Extending the Palladio Component Model using Profiles and Stereotypes

Max E. Kramer*, Zoya Durdik*, Michael Hauck†, Jörg Henss*, Martin Küster†,
Philipp Merkle* and Andreas Rentschler*
*Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
†FZI Research Center for Information Technology, Karlsruhe, Germany

***Abstract*—Extending metamodels to account for new concerns has a major influence on existing instances, transformations and tools. To minimize the impact on existing artefacts, various techniques for extending a metamodel are available, for example, decorators and annotations. The Palladio Component Model (PCM) is a metamodel for predicting quality of component-based software architectures. It is continuously extended in order to be applicable in originally unexpected domains and settings. Nevertheless, a common extension approach for the PCM and for the tools built on top of it is still missing. In this paper, we propose a lightweight extension approach for the PCM based on profiles and stereotypes to close this gap. Our approach is going to reduce the development effort for new PCM extensions by handling both the definition and use of extensions in a generic way. Due to a strict separation of the PCM, its extension domains, and the connections in between, the approach also increases the interoperability of PCM extensions.**

I. Introduction & Motivation

Domain-specific languages (DSLs), are designed for particular purposes. Their vocabulary is usually restricted to the terms used in their specific domains. On the one hand, this is their biggest strength: for a focused DSL, you can define the semantics more easily. Furthermore, narrow DSLs are easier to learn. On the other hand, being specific and concise is limiting, too. The expressiveness of a DSL might not be sufficient to cover all aspects of a problem that people would like to treat in the future. Additional DSLs need to be defined for the different viewpoints to cover the additional aspects. However, combining existing DSLs is difficult: First, the languages need to be combined on the abstract syntax level, making one DSL dependent on the other. Second, the tools that are generated for the concrete syntaxes need to be integrated. In general, this is a hard task.

A well-established approach to solve the problem of combining DSLs is to define a *core* metamodel with extensions designed as decorator models around it. Having a well-defined core, the conciseness and type-safety of the DSL are not corrupted. It is, however, the responsibility of the author of the extension to provide tool support that integrates the editors for the core DSL and the editors for the extension. The central concept is that the editors for the core model do not know about any decorator and can exist without any extension. Writing, maintaining and using additional editors for the extensions is a repetitive and time-consuming task. Here, we identify the need for a mechanism enabling general extensions of the DSL with a reusable set of tools for the definition of the extensions.

Another way to define extensions are annotations that seek to solve the problem by being very flexible. They allow to add arbitrary information to model elements. These additions must be interpreted at runtime by transformations processing the model. As type-safety is lacking, checking the syntax of any annotation is not possible at development time in EMF-based modeling environments, which is an unfortunate fact.

The abovementioned problems hold for the Palladio Component Model (PCM) [1] [2], too. While being a DSL for performance- and reliability prediction on the architecture level, a variety of additions to the core language have been introduced. Additions include information attached to connectors as performance completions, design patterns that are employed by the system or design decisions that have been made while designing the system structure or the deployment. All these additions have been defined outside the core of the PCM, raising the need for editors and adaptation of tools that process models with additions.

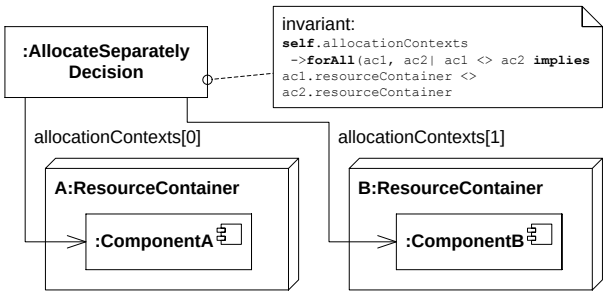

Fig. 1. Illustration of an architectural decision to allocate components separately applied to the deployment of PCM components (without stereotypes)

Fig. 1 exemplifies a design decision decorating a PCM allocation model. The element AllocateSeparatelyDecision indicates that the architect decided to deploy ComponentA and ComponentB on different hardware nodes A and B. The corresponding invariant formalises this decision through OCL.

Of course, the decision model that is attached to the PCM architecture model can be edited separately. The links to the PCM elements can be defined outside any PCM-related editor. But for an architecting process it is somewhat counterintuitive to have different model editors for the same information, which is architecture and architectural decisions. Therefore, we see the need for a generic mechanism that enables additions to the PCM without the need to re-generate any graphical or textual

Fig. 2. UML stereotype example (from [3], p. 663)

editor for PCM elements. The example above illustrates the situation we have in mind for extensions. A domain metamodel (decisions) exists independent of the PCM metamodel. The bridge between the two models, i.e. adding decision-related information to PCM elements, needs to be established by a profile-based extension mechanism.

For a generic extension mechanism, we identified five major requirements: It has to support *type-safe* extensions that can be created in a *non-invasive* way, so that the PCM does not have to be adapted to account for extensions. Furthermore, the mechanism has to be *lightweight* in the sense that extending the PCM has to be easier than creating a complete and separate metamodel. In addition, the mechanism has to be *flexible* and *intuitive* so that it is possible to create small as well as complex extensions using tools and notations that are familiar to many PCM developers. As a last requirement, extensions have to be *composable* so that new extensions can rely on existing ones.

This paper presents a PCM extension approach that addresses these requirements through profiles and stereotypes as known from the Unified Modeling Language (UML) [3]. It takes ideas from decorator models, but brings it to a level where PCM models can be extended without re-generating editors.

The rest of the paper is structured as follows: Section II provides the foundations for our work. The main features and concepts of our approach are presented in detail in Section III. Section IV discusses the technical realization of our extension approach. Four different examples of PCM extensions that would profit from our approach are presented in Section V. The paper is completed with a discussion of related work in Section VI and conclusions in Section VII.

II. Foundations

In this section, we first give an introduction on the profile concept of the UML for extending the UML metamodel. We then explain an approach based on the Eclipse Modeling Framework (EMF) called EMF profiles. This approach adopts the UML profiles concept to DSLs based on EMF and appears to be suitable for providing a lightweight extension mechanism for PCM models.

A. UML Profiles

The UML provides a mechanism called *Profiles* for extending metaclasses to adapt them for different purposes [3]. By using the profile mechanism, user-defined UML extensions called *stereotypes* can be defined that carry specialized semantics. During the last years, several UML profiles have been standardized by the OMG and are included in common UML tools.

Fig. 2, taken from the UML standard [3], shows the UML element Interface extended by a stereotype Remote. With this extension, remote interfaces are metamodelled indicating an interface with specific properties for remote usage. With appropriate tooling, remote interfaces can appear as basic building blocks, such as conventional UML elements. Note that the UML stereotype element inherits from the UML class element. Thus, associations and attributes can be used to further specify the properties of the stereotype.

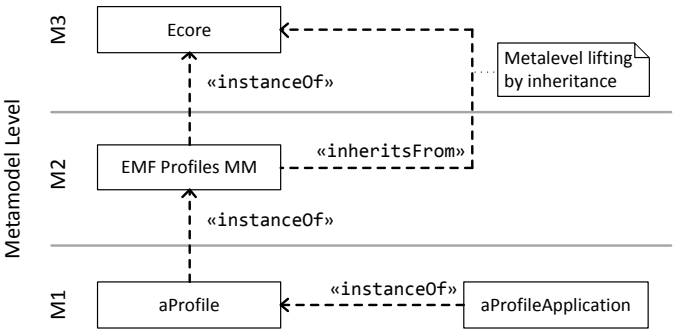

Fig. 3. EMF Profiles architecture (adapted from [5])

The UML *Profile* element is a special UML package used for collecting all UML stereotypes defined for a certain domain or platform (e.g. embedded systems, JEE, etc.).

B. EMF Profiles

To apply the profile mechanism to EMF-based metamodels, and to overcome issues with the UML profiles concept, Langer et al. developed an approach called EMF Profiles [4], [5].

Fig. 3 shows how EMF Profiles extend the metamodelling language Ecore with a profile mechanism. EMF Profiles provides a metamodel for defining profiles consisting of stereotypes. This metamodel inherits from Ecore metamodel elements. Aa profile inherits from Ecore EPackage, a stereotype inherits from EClass. A profile model is then defined on the same layer as a metamodel defined with Ecore (such as the PCM). The instance of a profile, i.e. a profile application, resides on the same layer as an instance of the metamodel.

EMF Profiles comes with a tool that supports the specification of profiles as well as the application of profiles in generated EMF and GMF editors. Besides adopting the concept of profiles from UML, this approach and the corresponding tooling facilitate the extension of EMF models by providing additional functionality. This includes the issue of storing profiles in a separate container (in UML, profile applications are directly included into the UML model), as well as a clean-up function to automatically delete stereotype instances referencing deleted model elements. As a stereotype can be used to extend multiple metaclasses, it provides a powerful mechanism for specifying extensions of EMF-based DSLs.

We decided to adopt EMF Profiles for implementing a lightweight extension mechanism for PCM models which is described in more detail in the following sections.

III. Concept

In this section, we explain the proposed PCM extension approach on a conceptual level. First, we outline the complete

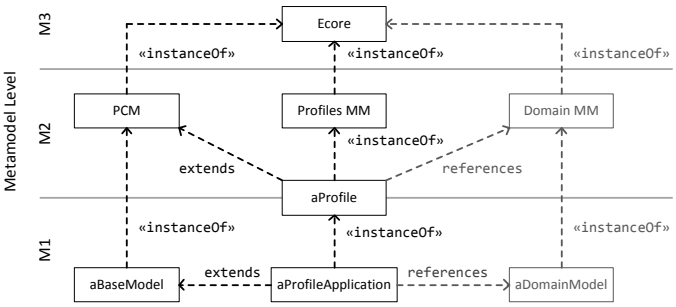

Fig. 4. Models, layers and relations of a profile application extending a PCM instance and referencing a domain model

approach and present all involved artefacts. Then, we explain in detail how an extension engineer defines an extension profile. Last, we switch to the process of applying an extension profile at runtime. Both extension steps are presented in a generic manner and illustrated using the PCM profile for design decisions as introduced before.

A. Overview

We propose a practical extension mechanism for Palladio to handle PCM extensions in a common way. In our vision developers of the Palladio Bench are enabled to extend the PCM by defining a profile that consists of stereotypes. In order to provide users of Palladio the possibility to use the extension profile, its developer has to register it using an Eclipse extension point. Afterwards users will directly be able to add information to models, because stereotypes provided in a profile can be applied in a generic way. Note that our approach only provides a way to add information to elements of the PCM. It does not address the use of this information nor is it concerned with extension possibilities for other parts of the Palladio Bench.

In Fig. 4 we show how a profile and its application are related to other models. A profile is an instance of the profiles metamodel, which is based on the metamodel presented by Langer et al. [4]. This metamodel conforms to the Ecore metamodelling language used by EMF and the PCM. A profile instance (aProfile) extends the PCM by specifying which specific elements of the PCM can be extended using stereotypes. If necessary, a profile instance can connect instances of its stereotypes to a domain model instance (aDomainModel). It is optional to refer to such an instance of a metamodel representing the extension domain (aDomainMM), but to show full capabilities we depict the corresponding metamodel, model and relations in Fig. 4. When a PCM instance (aBaseModel) is extended, an instance of the profile (aProfileApplication) is created to represent the profile and stereotypes applications. Such a profile instance can refer to other models conforming to the PCM or to models representing the extension domain. We placed the profile application instance on the border of the first two model levels to highlight the two roles it is playing. On the one hand, the profile application is instantiated by profile applications in M1. On the other hand, the profile application itself is an instance of the profile metamodel in M2.

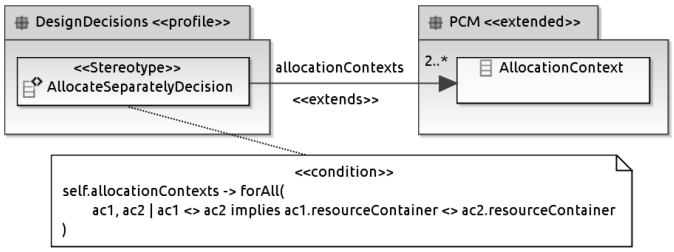

Fig. 5. Example design decision profile for the PCM showing the stereotype AllocateSeparatelyDecision and its context.

B. Extension Profile Definition

The first step in extending the PCM using our approach is the definition of an extension profile using the graphical profile editor. We are developing this profile editor based on the editor presented by Langer et al. [4]. As shown in Fig. 5, this editor is in turn based on the standard graphical editor for Ecore metamodels that is part of the EMF. For the design decision extension, for example, a developer has to create a profile that contains at least one stereotype for each possible decision type. In Fig. 5 we present how a part of this profile could look like. The profile contains everything that is necessary to enable users to specify the design decision mentioned in the introduction. This design decision states that two or more components have to be allocated to separate resource containers. The stereotype AllocateSeparatelyDecision of the DesignDecisions profile is linked to the metaclass AllocationContext of the PCM. This is done using a special extends relation. It states that this stereotype can only be applied to instances of the metaclass AllocationContext or to instances of its subclasses. Note that these semantics of extends relations between stereotypes and metaclasses are different from inheritance relationships among metaclasses. In our example, the extends relation is specified with a multiplicity of 2..*. Therefore, possible applications are restricted to cases where at least two entities have been selected. An additional restriction for applications of the AllocateSeparatelyDecision stereotype is specified using an OCL expression. This restriction is shown in the graphical profiles editor as an attached note. It states that the stereotype can only be applied to AllocationContexts that correspond to distinct resource containers.

With the profile editor it is also possible to define stereotype attributes and references as well as dependencies between stereotypes. These three possibilities are not used in our small design decision example and therefore only explained generically: Stereotype attributes are graphically specified in the same way metaclass attributes are defined using the graphical ecore editor. Such attributes have to be of primitive type and correspond to tagged values of UML profiles. References to complex-typed elements of additional domain models are graphically defined in the same way metaclass references are defined in EMF. Dependencies between stereotypes can be used to specify that a stereotype s1 can only be applied if another stereotype s2 has already been applied. These dependencies are graphically defined using a special depends on relation. We

introduce this new dependency concept for stereotypes with our approach as it is not part of UML or EMF Profiles.

The second and last step in defining a PCM profile is the registration of the profile. For this registration we use Eclipse's extension mechanism and define an extension point. This extension point requires extensions to provide a profile name and the path to the profile model itself. Every profile registered for the extension point is retrieved automatically by the Palladio Bench during start-up. Based on this, the application of profiles to regular PCM instances and all related tasks can be handled in a generic way as explained in the following subsection.

C. Profile Application

A profile registered for the extension point as described above is automatically available in the tree-based editor and in the graphical editor of Palladio. Whenever a user selects entities in an editor, stereotypes of extensions can be applied if the corresponding conditions are met. For this purpose, a menu is automatically created for each extension based on the information provided by the extension point registration and the profile itself. During runtime it is decided whether a stereotype can be applied with respect to the extends relations and OCL conditions defined in the profile. If a stereotype can be applied, the corresponding menu entry is activated.

During the application of a stereotype, a properties view showing the values of the stereotype application is opened. Using this view the user can enter values for attributes of primitive type. For complex-typed references elements of existing models can be selected. For each model and each applied profile the values for attributes and references of applied stereotypes are stored in a separate profile application file. This is an important difference to UML profiles and one of the reasons why parts of the Palladio bench can ignore profile applications if they are unaffected by them. The described possibility to edit stereotype application values is provided out-of-the-box and available for all profiles. An extension developer can also replace the standard view for editing stereotype applications with a graphical editor. Such a custom editor for stereotype applications could be used to conceal that elements are not part of the PCM.

When performing analysis, simulations or model transformations are executed in Palladio that may request extension information for input models using an API for our generic extension mechanism. The extension information obtained can be kept separate or can be used to create augmented versions of models in a pre-processing step. If a workflow in the Palladio Bench does not explicitly request information for a certain profile it operates as if the profile is not applied. On the other hand, Palladio can be easily extended by different types of analyses, without having to change PCM core functionality (e.g. by using various eclipse extension points or workflow engine extensions). Providing such analyses is out of scope of this paper, but the proposed Palladio extension can be used to facilitate such analysis without changing the PCM core (and the PCM core metamodel).

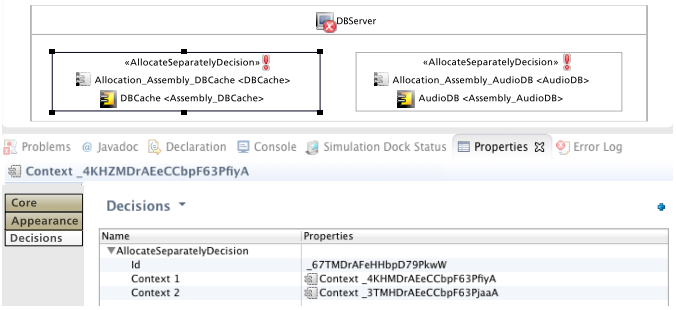

Fig. 6. Graphical editor showing a profile application and a validation error

IV. Implementation

The technical realization of the extension approach presented in this paper is going to reuse existing infrastructure from EMF and EMF Profiles. In this section we outline which steps are necessary to implement our approach as three artefacts: A graphical profile editor, an integration into Palladio's tree-based and graphical editors and an API for persisting and retrieving extension information.

A. Extension Profile Definition

The profiles editor of EMF Profiles provides the basic functionality needed to define the proposed PCM profiles. It is possible to define new stereotypes and to specify to which PCM elements can be applied. Furthermore, primitive-type attributes and complex-typed references can be added to stereotypes. Support for additional OCL constraints restricting the application of stereotypes is missing and has to be implemented for our approach. This could be done by integrating functionality of existing OCL-editors in order to provide syntax-highlighting, auto-completion and syntax checking. To ease the definition of big profiles, support for profile subpackages and inheritance among stereotypes could be added. EMF Profiles currently forces profile developers to choose either a graphical profile editor or a tree-based profile editor. In the future it would be helpful if a single profile can be edited with both editors as it is the case for Ecore metamodels.

B. Profile Application

Currently EMF Profiles provides only restricted tool-support for the application of profiles and stereotypes using existing tree-based and graphical model editors. It is possible to apply stereotypes to a model once a profile has been explicitly loaded by a user. Our idea is to perform an automatic retrieval of registered and applicable profiles based on the current modelling context. This can be done by realizing a profile registry mechanism using an Eclipse extension point. To incorporate support for profiles into the existing PCM graphical editors some additional effort is necessary. Stereotyped entities have to show applied stereotypes in guillemets. Validation of stereotype constraints has to be done while adding and editing stereotype applications. Fig. 6 shows for our design decision example how the integration into a graphical editor and the result of a failed validation may look like.

Setting values for attributes or references and the persistence of stereotype applications is not yet fully supported in EMF Profiles. Therefore, an editor for stereotype applications has to be implemented for our approach. In addition, the type, multiplicity and general OCL constraints defined in registered profiles have to be evaluated in a modified version of the stereotype application hook. As mentioned above, it is currently possible to restrict stereotype applications in the profile editor of EMF Profiles with respect to the type and number of elements to be extended. In the integration for the tree-based and graphical model editor, however, only the type restrictions are evaluated. For our approach, it is crucial that correct multiplicities are ensured. In cases where the lower bound of the multiplicity of an extends relationship is greater than one, it has to be ensured that the corresponding stereotype can only be applied to the desired number of elements. Upper bounds of extends relationships have to be checked similarly. At the moment it is not possible to set the values for attributes and references of stereotype applications in a dedicated view. For our approach, a property view that can be used to enter attribute values and to select referenced elements as described in the previous section has to be implemented. Moreover, a clean-up mechanism has to be introduced to remove dangling stereotype applications caused by model element deletions.

A last important area for the implementation of our extension approach is the persistence, validation and retrieval of stereotype application information. In EMF Profiles stereotype applications have to be explicitly persisted in publicly visible XMI-files. For our approach, stereotype application files have to be automatically validated and stored. Furthermore, these files should be hidden in the package explorer of the Palladio Bench as long as nothing contrary is specified by the extension or by plug-ins depending on it. This can be achieved by using a specific PCM project nature or project type. Last but not least, the implementation of our approach has to provide an API for querying and retrieving the stereotype application information in analyses, simulations and transformations.

V. Example Extensions

A number of examples from existing Palladio decorator models can serve as illustration of both the usefulness and applicability of the EMF profile-based extension mechanism that we introduced so far. The examples come from the performance engineering domain, such as the middleware completion V-C, and the specification of measurements V-D. Others are aligned with the use of PCM as an architecting tool: The recording of design decisions and patterns V-B, and finally the definition of security-related information to PCM elements V-A, which we will discuss in the following.

A. Security

In the EMERGENT project, initial work has been carried out in order to extend Palladio by an analysis of security and privacy attributes of a software architecture [6]. The approach aims at supporting the software architect or Quality of Service (QoS) analyst in specifying such attributes on the architecture level. By analysing the architecture, it helps answering questions related to security and privacy issues, such as “Is it possible for an attacker to get access to a certain kind of data?” A prototypical implementation of the approach has been developed using a decorator metamodel. This metamodel facilitates the specification of security-related annotations to a PCM model. The PCM model and the annotations are then transformed into an analysis model that can be solved by a protocol checker using predefined rules.

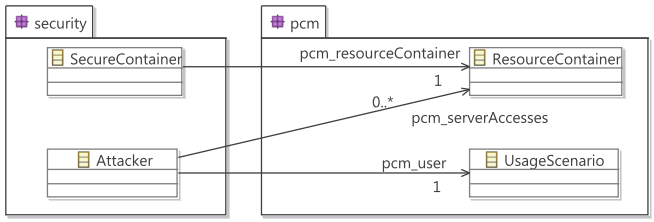

Fig. 7. Security PCM annotations using a decorator model

Fig. 7 shows an excerpt of the security annotation metamodel and the referenced PCM metamodel entities. In the figure, two security-related elements are shown that decorate the PCM metamodel. A SecureContainer can be used to indicate that a PCM ResourceContainer meets certain requirements for secure operation. In addition, the Attacker element can be used to model malicious users in the system. The approach uses the PCM UsageScenario to indicate such users, hence the according reference. For the Attacker element, references to PCM ResourceContainers can be provided to indicate on which servers the user has access to.

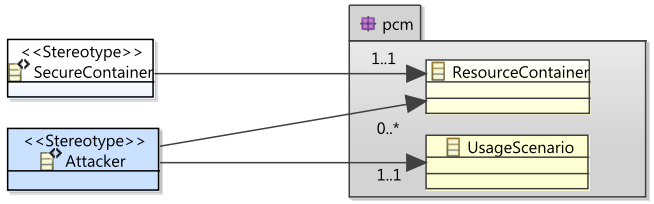

Fig. 8. Security PCM stereotypes using the EMF Profiles approach

Fig. 8 shows how the security extensions can be implemented by using the EMF Profiles approach. The elements SecureContainer and Attacker are defined as stereotypes residing in a security profile. In this example, we only use a subset of the Palladio security extensions. However, the remaining security extension elements can be modelled as stereotypes in a similar way. The example shows that, regarding the metamodel, the existing decorator approach can be replaced by using the EMF Profiles approach. By using common tooling as described in Section IV, modelling security attributes can be included in existing PCM editors. We plan to convert the complete existing security decorator model into a profile model and integrate the profile model into the existing security analysis toolchain, where the model is transformed into an analysis model which

is checked with predefined security rules. If common PCM editors support the profiles mechanism, no custom editor for the security extensions has to be implemented Instead of implementing a custom editor for the security extensions, common PCM editors supporting profiles could be reused, leading to decreased development efforts needed for providing editor support for the security extension.

B. Design Patterns and Rationale

Design patterns are a subclass of design decisions, which can influence one or multiple architectural elements, such as components, provided and required roles, and connectors. In architectural diagrams, design patterns are commonly represented with *pattern roles* and *role connectors* [7]. For example, a Model-View-Controller (MVC) pattern [8] consists of three roles (*Model, View*, and *Controller*), and three connectors between them (*View-Model, Controller-View*, and *Controller-Model*). In the PCM this pattern would be represented through three component instances (*AssemblyContext* in PCM) and three connectors between them (*AssemblyConnector* and *OperationProvidedRole* in PCM). Consequently, the rationale for the existence of these PCM elements would be the decision to apply the MVC pattern. This decision shall be explicitly visible in the model to prevent accidental modification or deletion of the participating components and connectors.

We have implemented a metamodel to capture pattern design decisions for the PCM. An excerpt of it can be seen on Fig. 9 (for full version refer to [9]). The two levels of instantiation required for the design patterns can be seen. The meta-level defines that a pattern consists of roles and connectors between roles. First, the general pattern gets instantiated with a *PatternType* element, for example, MVC or Observer [8]. Such a type instance specifies the number of and the names for the roles and connectors, e.g. Observer pattern type has two roles (*Observant* and *Observee*), and one connector between them.

Second, a modelled system can contain multiple patterns of a certain pattern type. Therefore, a pattern gets instantiated with a *PatternApplication* element referencing the *PatternType*. At this step, pattern roles and role connectors are mapped to the components and component connectors in the PCM model.

Such a metamodel realisation has several drawbacks. First, the two instantiation levels increase the complexity and reduce the comprehensibility of the metamodel. Second, the pattern decoration model references multiple PCM elements, and evolution of the PCM may ripple though multiple elements of the pattern metamodel causing significant maintenance effort of the tool-chain. Finally, when working with the PCM models, which components are annotated with the pattern information, the information on these annotations is not visible unless the related plugins are installed. This might lead to undesired changes, e.g. deletion of a component that was part of a pattern.

These drawbacks of the current implementation could be resolved through the proposed generic PCM profile mechanism. The double instantiation would not be required anymore, and the metamodels will be decoupled, thus allowing for an independent co-evolution of the metamodels. Finally, the

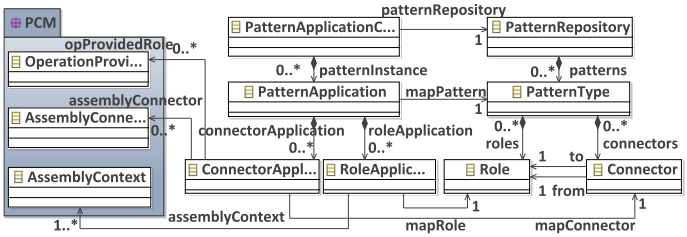

Fig. 9. Excerpt of the pattern metamodel: Decoration of the PCM

annotation information (although in a brief way) would be visible also without additional plugins, thus reducing the risk of unintended design decision violation.

C. Performance Completions

Performance completions, initially proposed by Woodside et al. [10], is a technique to programmatically refine an architecture model, to have it better reflect performance influences arising from technological choices. Palladio offers a rich set of performance completions, implemented as in-place model transformations.

Completion transformations can be parameterised via mark models. A mark model is an external configuration model which decorates the architectural main model. For some completions in Palladio, parameter configurations are forming a feature-tree, reflecting the tree-like structure of options and possible combinations thereof. For instance, in order to predict the effect of middleware communication, Palladio provides several middleware-specific completions, each with its own set of configurable options.

As of today, Palladio's *Chilies Project* is one possible way for configuring completion transformations to be run as a preprocessing step prior to performance analysis. In Chilies, each transformation needs to be triggered manually. Elements are annotated outside of the graphical PCM editors, in a tree-based feature configuration editor.

Other completions, like the event-based middleware completion [11], introduce several new concepts by extending the Palladio metamodel with new classes. At the moment, extending PCM is done invasively, and new elements are further hard-coded into the GMF editors.

At the bottom line, current integration of completions is hard to handle because mark models are not standardised in a manner that fosters integration into the editors. Also, completions which extend the core metamodel with new concepts need to modify it, since there is no method to extend the component model and the graphical editors non-invasively.

With a PCM profile mechanism in Palladio, we can define a standardised framework for completion integration, which helps us to solve the aforementioned shortcomings. We are gaining the following benefits: Better integration into the editors, the chance to non-invasively add new concepts to the core, as well as the automatic integration into Palladio's workflow.

A profile mechanism lets Palladio users annotate model elements directly within the editor, in a uniform manner. This can be realised with a stereotype which is capable of

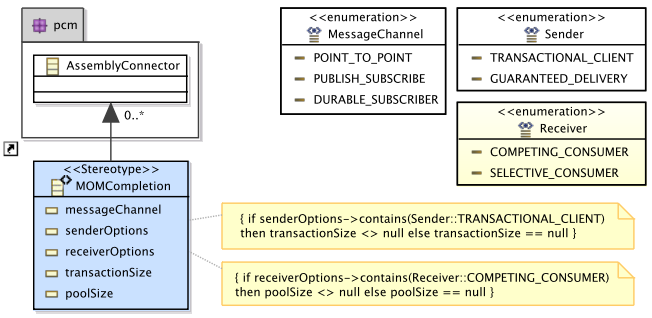

Fig. 10. Connector completion for message-oriented middleware

specialising a PCM element, displaying an icon as a distinct visual feature in the editors, and listing the stereotype's attributes context-sensitively in the properties view. Still, a lot of completions exist, whose parameter space is modelled as a feature tree. In the past, these completions had to be configured in an external feature editor. As an alternative, when registering a new stereotype for these completions, a custom properties view tab can provide an editable view for such feature configurations. With the help of OCL, however, even 3-layered feature trees, as used by the message-oriented middleware (MOM) completion, can be mapped onto a set of tagged values (depicted in Fig. 10).

As a second benefit, Palladio's core models can be extended with further concepts, without polluting the core models. Existing concepts, like, in the case of the event-based middleware completion, events, event groups, event connectors etc., but also future macro concepts, can be factored out of PCM's core model, keeping it clean and concise.

A third benefit is that each completion transformation can be obliged to specify at registration time a list of profiles it processes. This enables Palladio to automatically trigger a completion, depending on the profiles which are found to be applied to a given model. If execution is automatically triggered, manual approaches like that of Chilies would become obsolete.

D. ProbeSpecification

Another promising field of application is to attach measuring information to PCM models, which can then be used by the simulation to decide where to take measurements to satisfy the performance analyst's information need. When, for instance, the response time of a system call is of interest, the corresponding EntryLevelSystemCall could be annotated to indicate that the time span between call and return shall be recorded throughout a simulation run. Enabling the performance analyst to define measurement points manually avoids wasting resources due to unwanted measurements and renders highly specific measurement settings possible.

With the idea in mind of specifying measurement points manually, the *ProbeSpecification* metamodel (PS-MM) and a corresponding framework has been developed. While the metamodel allows for expressing measurement points, the framework provides the measuring infrastructure to Palladio simulators. The framework is already employed in SimuCom as

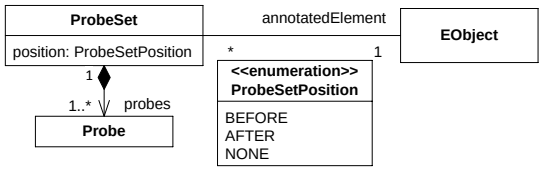

Fig. 11. Current ProbeSpecification metamodel (excerpt)

well as EventSim [12], whereas the metamodel has never been adopted. One of the major reasons is the lack of tool support to handle PS-MM instances. Namely, automatically generated EMF tree editors do not provide the ease of use we desire, which is mainly due to the separate PS-MM editor forcing the modeller to constantly switch between Palladio editors and the PS-MM editor when specifying measurement points. An integration with Palladio's graphical GMF editors does not exist at all and failed so far due to limited development capacities. As a result from the absent PS-MM support by Palladio simulators, measurement points are currently hard-coded in the simulation code, hence bringing along all drawbacks for such a predefined setting as discussed earlier. Adopting EMF profiles, we believe to benefit from a reduced development effort for integrating the PS-MM with Palladio's EMF and GMF editors.

The PS-MM was designed as a decorator model as illustrated in Fig. 11. ProbeSets subsume one or more Probes to form a unit which can then be mounted at a measurement location. While Probes can be seen as devices capable of measuring different quantities (e.g. the current time instant or the queue length of a processor), a ProbeSet can be seen as the application of measuring devices at a specific location (e.g. an EntryLevelSystemCall). Not shown in Fig. 11 is the concept of Calculators, which serve to calculate performance metrics out of measurements produced by Probes. Calculating response times, for example, involves two timestamps representing a service's invocation time and its return time. The reference annotatedElement reveals the decorator nature of the metamodel. Each ProbeSet decorates exactly one EObject, which can be an arbitrary PCM modelling-element, but also any other modelling-element from an arbitrary EMF meta-model.

A major drawback accompanying this design is the missing type safety for ProbeSet applications. That is, ProbeSets may be applied to any instance of an EMF metaclass even if the application makes no sense at all. OCL constraints could mitigate this problem, but introduce additional complexity.

To address these shortcomings, we are planning to refactor the PS-MM towards the presented profiles approach. Since the ProbeSpecification targets not only Palladio, but should also be applicable in similar contexts, *generic profiles* [5]—a part of EMF profiles—seem well-suited and is therefore planned to be supported by Palladio profiles. Generic profiles take into account that profiles are sometimes created with the intention to apply them to a broader range of metamodels. Thus, stereotypes in a generic profile may not refer to concrete metaclasses, but instead extend *generic types*. Each generic type represents a role which needs to be bound to a concrete metaclass once applying the profile to a specific metamodel. This is illustrated

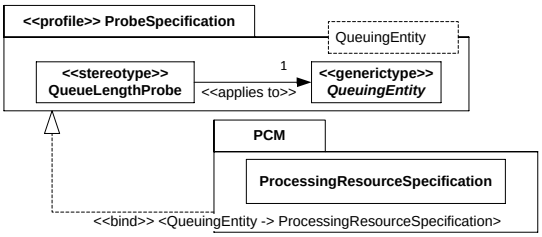

Fig. 12. Generic profiles example

by the example shown in Fig. 12. In the example, the PCM metamodel and the PS-MM are unaware of each other until the binding maps the abstract role of a QueuingEntity to the PCM metaclass ProcessingResourceSpecification.

The challenges of this approach are as follows. First, it has to be decided what PS-MM metaclasses will serve as stereotypes. Deciding for ProbeSets would closely resemble the current metamodel, but bears some disadvantages compared to using Probes, which also seems more natural. Second, although not explicitly foreseen in EMF profiles, the PS-MM will likely need to reference the corresponding profile since Calculators residing in the PS-MM refer to their ProbeSets or Probes. Third, SimuCom's Xpand transformations need to be extended and TraversalListeners are required for EventSim (cf. [12]).

VI. Related Work

The autosar metamodel [13], used in the automotive domain, supports annotations on multiple levels. For each Identifiable entity it is possible to add multi-language textual information as annotations. Each of these annotations is identified by an arbitrary string given the Origin of the information. Furthermore, each autosar PortPrototype can have multiple specialised and typed annotations to provide domain specific information. In the autosar metamodel, annotations are realised with dedicated containment relations.

Another annotation mechanism can be found at the heart of the Ecore metamodel [14]. It allows to add EAnnotations to every instance of EModelElement, i.e. basically any Ecore element can be annotated. An EAnnotation has an identifier and a key value map to store arbitrary values. This mechanism is especially used for storing metamodel documentation. The EAnnotations are stored as containment relations.

In the Service Architecture Meta-Model (SAMM) [15] developed in the Q-ImPrESS project, annotations are used to add QoS specifications, like resource demands, to model elements. Annotations in the SAMM are stored in a detached model using a decorator approach, i.e. each annotation has references to the annotated element. While initial tool support for specifying annotations exists, no tooling is available for adding new annotations. In addition, the implementation is strongly connected to SAMM concepts and thus not reusable.

The Ontology Annotation Metamodel (OAM) [16] was developed in order to derive ontologies from models based on annotated information. This way ontology concepts such as classes or object and data properties are assigned to an existing model. The implementation is based on the Atlas Model Weaver (AMW) and uses a dedicated model based on the AMW mapping model to store link elements that map between annotations and model elements.

VII. Conclusions

In this paper, we presented an approach for extending the PCM in a standardized way. The proposed extension mechanism uses profiles consisting of stereotypes to define extensions for the PCM. As a result, extensions for Palladio can be specified in a uniform format, which makes it possible to handle them generically. In addition to the extension format, we presented a mechanism to register extensions and we introduced a possibility to integrate them into Palladio's graphical editors. The envisioned approach strictly separates base models and extension information. As a result, the Palladio Bench can be unaware of future extensions. In order to provide a solid foundation for our approach, we also discussed practical challenges for an implementation based on EMF Profiles.

Using four existing PCM extensions, we illustrated that current extensions would benefit from a consistent extension mechanism using profiles. The example of a security extension showed that the used decorator approach could easily be replaced by a profile in order to benefit from generic tooling. With an extension for design patterns and related rationale, we demonstrated how extension metamodels can be simplified if the relation to the PCM is factored out into a profile. The third example of performance completions exemplified how our extension approach could render concepts of extension domains more explicit. Finally, an extension example for specifying measurements illustrated a use case for generic profiles, which are supposed to be applicable to a broader range of metamodels.

The approach that we presented in this paper meets the requirements that we identified upfront: its profiles are *type-safe* and *non-invasive* as they can be defined, registered and applied based on the type of existing PCM elements without changing the PCM. The Palladio Bench can be unaware of extensions, because extended model instances appear to the user and the Palladio Bench as ordinary models as long as they do not explicitly ask for extension information. Furthermore, the approach is *lightweight* because it provides specific concepts for stereotype applications so that it becomes unnecessary to explicitly model similar application mechanisms in extension metamodels. In addition, the approach is *flexible* because it supports profiles of different complexity: profiles that add only simple-typed information as well as profiles that refer to extension metamodels or involve application conditions formulated using the Object Constraint Language (OCL). We are convinced that the approach we presented is *intuitive* because profiles are graphically defined using a notation that is adopted from UML profiles. Profiles can be based on other profiles but they can also be applied independently of each other. Therefore, the approach supports *composable* extensions.

In future work, we are going to implement the proposed approach in the Palladio Bench and we plan to evaluate it using some of the example extensions presented in this paper.

ACKNOWLEDGEMENTS

We are grateful to Anne Koziolek and Erik Burger for co-developing this approach and thank Benjamin Klatt, Klaus Krogmann and Qais Noorshams for their comments on drafts. Many thanks go to the anonymous reviewers for their feedback.

REFERENCES

[1] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *Journal of Systems and Software*, vol. 82, no. 1, pp. 3 – 22, 2009.

[2] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, and M. Kuperberg, "The Palladio Component Model," Karlsruhe, Tech. Rep., 2011.

[3] Object Management Group (OMG), "OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.1," http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/, August 2011.

[4] P. Langer, K. Wieland, M. Wimmer, and J. Cabot, "From UML Profiles to EMF Profiles and Beyond," in *Objects, Models, Components, Patterns*, ser. Lecture Notes in Computer Science, J. Bishop and A. Vallecillo, Eds. Springer Berlin / Heidelberg, 2011, vol. 6705, pp. 52–67.

[5] ——, "EMF Profiles: A Lightweight Extension Approach for EMF Models," *Journal of Object Technology*, vol. 11, pp. 1–29, 2012.

[6] EMERGENT Project, "Deliverable D.Q1.G1.3.2 / M24: Beschreibung von Anforderungen und Architektur für Qualitätssicherungssysteme in föderierten Cloud-Umgebungen," http://www.software-cluster.com/en/projects/joint-projects/emergent-en, 2012.

[7] M. Elaasar, L. C. Briand, and Y. Labiche, "A Metamodeling Approach to Pattern Specification," in *9th Int. Conf. on Model Driven Eng. Lang. and Syst.*, ser. MoDELS'06, 2006, pp. 484–498.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, Amsterdam, 1995, no. 1.

[9] Z. Durdik and R. Reussner, "Position Paper: Approach for Architectural Design and Modelling with Documented Design Decisions (ADMD3)," in *Proc. of the 8th Int. ACM SIGSOFT Conf. on Quality of Software Architectures*, ser. QoSA '12. New York, NY, USA: ACM, 2012, pp. 49–54.

[10] M. Woodside, D. Petriu, and K. Siddiqui, "Performance-related Completions for Software Specifications," in *Int. Conf. on Software Engineering (ICSE)*, 2002.

[11] B. Klatt, C. Rathfelder, and S. Kounev, "Integration of event-based communication in the palladio software quality prediction framework," in *Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS (QoSA-ISARCS 2011)*. New York, NY, USA: ACM, 2011, pp. 43–52.

[12] P. Merkle and J. Henss, "EventSim – an event-driven Palladio software architecture simulator," in *Palladio Days 2011 Proceedings*, ser. Karlsruhe Reports in Informatics ; 2011,32, S. Becker, J. Happe, and R. Reussner, Eds. Karlsruhe: KIT, Fakultät für Informatik, 2011, pp. 15–22.

[13] AUTomotive Open System ARchitecture (AUTOSAR), "AUTOSAR Software Component Template," October 2011.

[14] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*, 2nd ed., ser. Eclipse series. Addison-Wesley Longman, Amsterdam, Dec. 2008.

[15] Q-ImPrESS Project Consortium, "Project Deliverable D3.1 Prediction Model Specification," http://www.q-impress.eu/wordpress/wp-content/uploads/2009/05/D3.1-Prediction_model_specification_v20.pdf, 2009.

[16] G. Hillairet, "AMW Use Case - Metamodel Annotation for Ontology Design," http://www.eclipse.org/gmt/amw/usecases/oamusecase/.