Identifying Semantically Cohesive Modules within the Palladio Meta-Model

Misha Strittmatter, Michael Langhammer
Institute for Data Structures and Program Organisation (IPD)
Karlsruhe Institute of Technology (KIT)
{strittmatter | langhammer}@kit.edu

Abstract: The Palladio Component Model (PCM) is a modeling language for component-based business information systems. Its primary and original focus lies on performance prediction. Its core meta-model is organized within a package structure within one meta-model.

Over time, the meta-model was extended, which allowed to model additional concerns. The extensions were made directly into the package structure, mostly without creating sub packages. Some cross-cutting concerns were placed inconsistently. This deteriorated the organizational structure of the meta-model, which negatively influences maintainability, understandability and extensibility.

To solve this, the meta-model should be restructured into meta-model modules. Within this paper, we identified concerns which are contained in the meta-model, to form a basis for the future modularization. This paper does not propose a definite modularization, but possible building blocks. What may be put as a single module or just a package within a module will be subject to future discussion.

1 Introduction

The Palladio meta-model is currently contained within one file and is subdivided into packages. This structure formed through the years, while the PCM was developed and extended. The top-level packages are partly aligned with the containment structure of the PCM submodels (e.g. repository, system). There are other top-level packages, which contain cross-cutting or general concepts (e.g. parameter, composition, entity). Further ones are concerned with quality dimensions (e.g. reliability, QoS Annotations). Some extensions distributed their new classes across packages, which were semantically fitting. This injecting as well as the scattering of new content is problematic, as some concepts cannot be easily associated with their concern.

Within the scope of the Palladio refactoring [SMRL13], the current all-enclosing structure is decomposed into a more modular one. The goal is to divide the Palladio meta-model into smaller, semantically cohesive, concern based meta-models (meta-model modules). From this, a small structural core can be formed. Further needed modeling constructs are regarded as extensions and can be loaded, if desired.

This modularization of the meta-model has several advantages. The meta-model becomes better maintainable and easier to extend and understand. When a Palladio model is created,

Proc. SOSP 2014, Nov. 26–28, 2014, Stuttgart, Germany

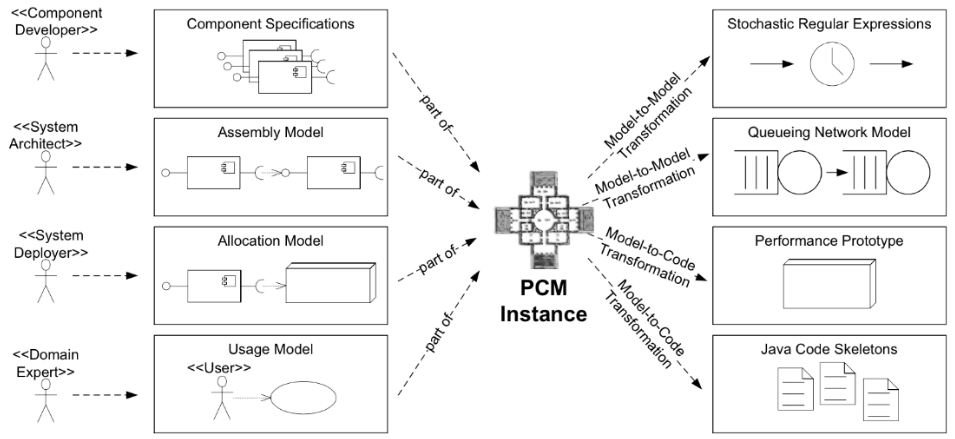

Figure 1: Simplified overview of the different PCM models that have to be modeled (from [RBB+11])

the model developer can choose the meta-model modules which are relevant to him and is not confused by the full complexity of content from all extensions. This is especially necessary, when the Palladio approach is extended to further quality attributes and even other domains than software engineering.

Within this paper, we identify concerns which are contained in the meta-model, to form a basis for the future modularization. This paper does not propose a definite modularization, but possible building blocks. What may be put as a single module or just a package within a module will be subject to future discussion.

This paper is structured as follows: In Section 2, we provide an introduction to the PCM and further literature references. In Section 3, the current structure of the Palladio meta-model is explained and illustrated. Section 4 describes how we approached the identification of the concerns, which are contained in the meta-model. Next, in Section 5, the semantically cohesive modules which were identified are presented and their interdependencies explained. Section 6 concludes the paper and states lessons learned. Finally, Section 7 outlines the next steps which are necessary for the modularization process.

2 Foundations

The PCM is a component model, that can be used to create a model of a software system and, amongst other, predict its performance without actually implementing the system. It has been introduced in various publications [BKR09, Koz08, RBB+11]. In order to predict the performance of a system users of the PCM have to model the system in six different models (see Figure 1).

First, a user has to create the Repository model. Within the Repository the components and interfaces as well as the provided and required roles between them have to be defined.

In the second model, the System model, the composition of the components has to be defined. Moreover, provided and required roles of the whole system have to be defined.

The third model is the Service Effect Specification (SEFF) model. A SEFF, which is a behavior model, has to be specified for each method provided by a component. It contains control flow elements like resource demands and internal call actions as well as external call actions. These provided information is used by PCM to predict the performance of the system in a later step.

The fourth model is the so called Resource Environment Model. It specifies the resources which are available for the system, i.e. different servers, their computing and memory power as well as their network connection have to be defined.

The fifth model is the Allocation model. It specifies how components (of the system) are deployed on the resources.

The last model is the Usage model. The Usage model describes the number or arrival rate of users who are using the software system.

These six models act as input values for the PCM allowing to predict the performance of the system. The results of the prediction are the response time of the system and methods and resource utilization. The results can be visualized in, for example, time charts or pie diagrams.

The Palladio bench, which is the implementation of the PCM approach, is build using the Eclipse Modeling Framework (EMF) [SBPM08]. To allow performance prediction and code generation, model-to-model transformations and model-to-code generation are used.

3 Current Package and Dependency Structure

In this Section we give a brief overview about composition of the current Palladio Component meta-model. Currently, the meta-model is divided in several hierarchical packages. Table 1 gives an overview about the packages. Generally, the packages are divided according to the six models presented in Section 2. The Allocation package, for example, contains the classes that are used in the allocation model.

The Palladio meta-model also has dependencies to the Identifier, the Units and the Stochastic Expression (StoEx) meta-model. The Identifier meta-model only includes the class Identifier, which has the unmodifiable attribute ID. During the creation of an instance a (UUID) is created and assigned to the ID attribute. Hence, Identifier, and all of its subclasses, do have a UUID to identify the elements globally. The Units meta-model contains classes that specify different Units that are used in a SEFF. The StoEx meta-model includes classes to create stochastical expressions in a SEFF. Therefore, the StoEx model uses classes that allows the creation of new random variables.

Figure 2 shows the current package structure and the dependencies between the packages within the Palladio meta-model.

Package	**Explanation**
pcm	root-package that contains all other packages
core	contains the entity and composition packages
entity	contains all core classes that are shared between other packages
composition	contains all classes that are used in composite models
repository	contains all classes that are used in the repository model
resourceenvironment	contains the classes to specify the resource environment
system	contains the System class that represents the system model
allocation	contains classes that are necessary for an allocation model
usagemodel	contains classes that are necessary to create a usage model
subsystem	contains a class to compose sub-systems of a system
resourcetype	contains classes to specify resource types
protocol	contains an abstract class as a stub, such that interfaces could be extended by protocols
parameter	contains the specification for variable usage
reliability	contains classes for reliability prediction
seff	contains all classes that are necessary to model a SEFF
seff_performance	sub-package of SEFF that contains performance specific classes
seff_reliability	sub-package of SEFF that contains reliability specific classes
qosannotations	enables the annotation of quality information
qos_reliability	sub-package of qosannotations that contains classes to that are necessary for reliability annotations
qos_performance	sub-package of qosannotations that contains classes to that are necessary for performance annotations

Table 1: Current packages of the Palladio Component meta-model

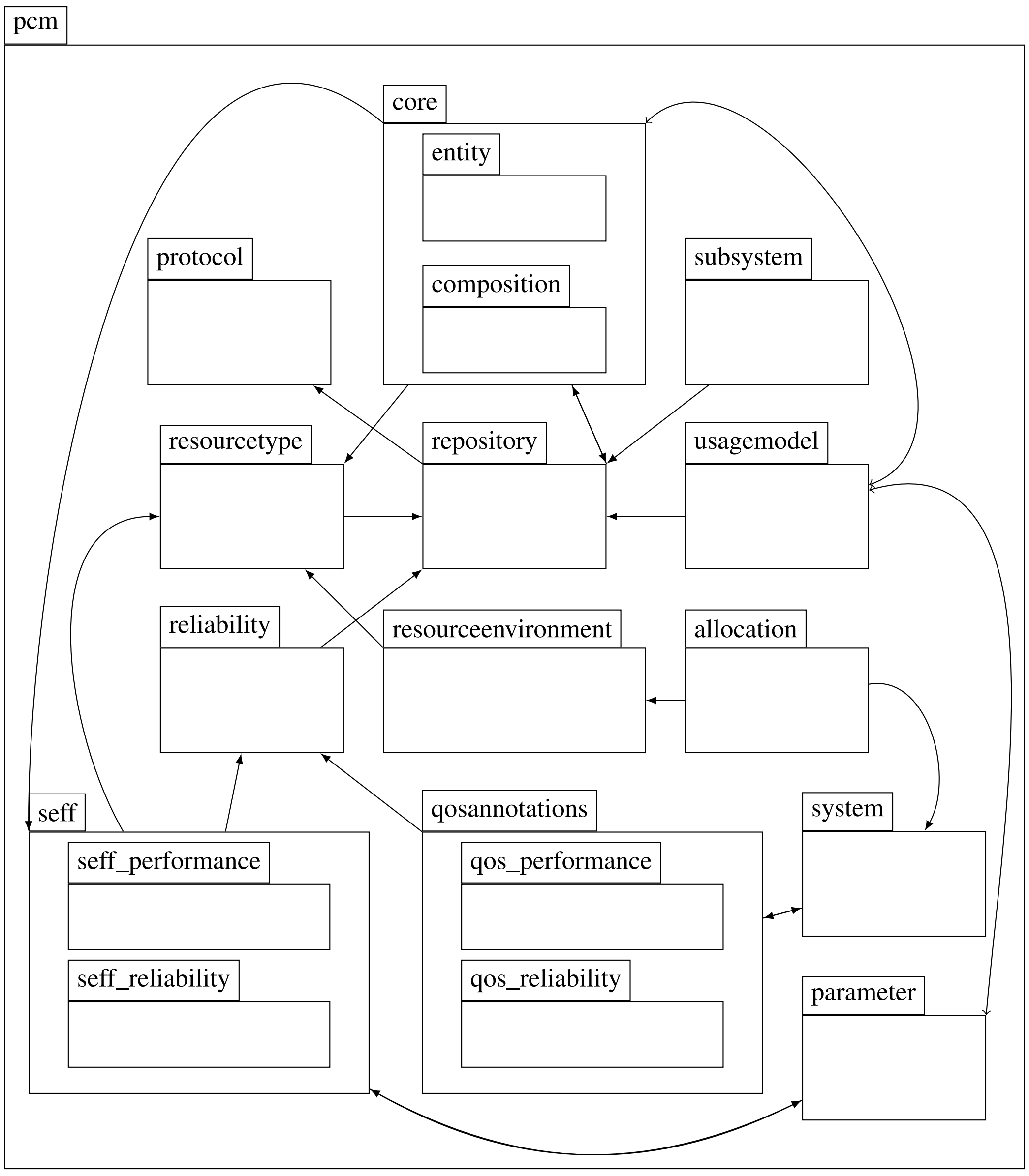

Figure 2: Current package structure of the Palladio Component meta-model. An arrow between two packages implies a dependency from one package to the other package. In order to simplify the Figure we assume that the arrows are transitive. For example, qosannotations depends on reliability and repository. However, since reliability already depends on repository we omit the arrow from qosannotations to repository. We also count dependencies from and to sub-packages as dependencies from and to the parent package. Finally, we omited dependencies to the entity package.

4 Identification Procedure

The content of this paper originates from a thorough, longer running inspection of the meta-model. Beforehand, we outlined the concerns and their dependencies from an outside perspective (similar to the view of an experienced Palladio user). It was the goal of the inspection to confirm, refine and correct this top-down classification with technical bottom-up information. We examined, how the package structure, relates to the concerns and arranged all classes to their belonging concerns. We also sought for new concerns.Because besides the top-down visible concerns, the invisible infrastructure of the meta-model (abstract class hierarchy) and hidden features can only be seen when looking into the meta-model. We considered how the concerns could be internally structured (e.g. by subpackages or subconcerns) and which concerns should rather be fused with others. Further, we inspected the dependencies of every class and if they confirmed to our idea of dependencies between concerns.

5 Concerns of the Palladio Meta-Model

Within this section, we will describe the concerns, which we identified in the Palladio meta-model. We will explain, what is the purpose of the concern and what are the main concepts. How do they originate within the package structure and why they were assembled like this. Outgoing dependencies are shortly explained. Incoming dependencies which currently exist in the meta-model and are problematic are discussed. Such dependencies could be of the type which go from a core concern to an enhancing concern. Please note that this is in no way an explanation of the meta-model in detail. Not every class and individual dependency is explicitly explained, but only the ones which are important for the concerns from an outside perspective. For additional details, please consult the tech report [RBB$^+$11].

An overview of the core concerns is shown in Figure 3. Dependencies to the entity concern are not shown for simplicity, as almost every concern is dependent on it.

5.1 Entity

The entity concern contains a simple class hierarchy to provide IDs and names to subclasses. The content of this concern stems mostly from the entity package. The interface dependent part has been factored out. The exception is the identifier class, which carries an ID and makes all subclasses uniquely identifiable. It is contained within its own minimal meta-model. It could be beneficial to locate this whole concern into that meta-model, as names are also used in further meta-models.

This concern has no outgoing dependencies. However, most classes are dependent on this concern, as IDs and names are widely used. So, we will not explicitly state these dependencies everywhere.

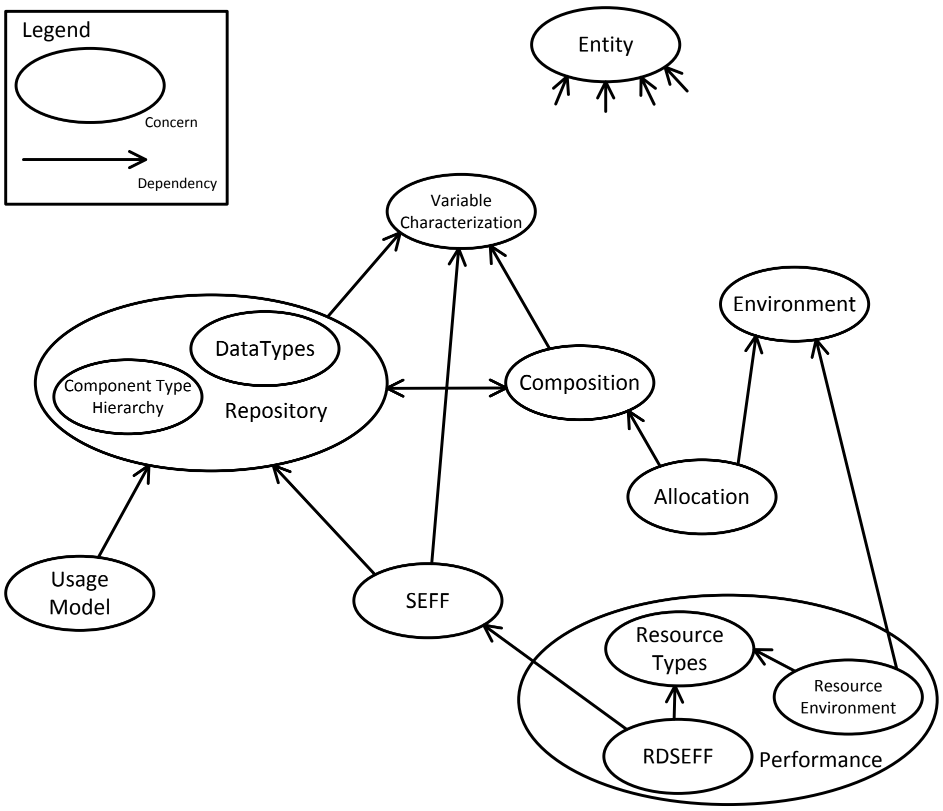

Figure 3: Main Concerns

5.2 Variable Characterization

The variable characterization concern provides capabilities to specify properties of variables (e.g. bytesize, structure, value), These variables are mostly input and output parameters of calls as well as in component parameter specification. This concern is currently known as the parameter package. However, as the parameter is a concept and meta-class from the repository, we chose variable characterization as a name. As the concepts, which are contained in the concern, are used in different places, we did not merge it, but left it as an independent concern.

The main concepts of this concern are: The VariableUsage holds the name of a variable and an arbitrary number of characterizations. The VariableCharacterisation holds type and value of the characterization. The VariableCharacterisationType declares which characterization should be specified: bytesize, structure, value and so on.

There are no outgoing dependencies, but to the StoEx meta-model, as random variables are used.

5.3 Repository

The repository concern covers the main concepts, which users know from the conventional repository model within the PCM. Within a repository model, interfaces, components and their relations can be specified. The repository may also contain custom data types, which are meta-modeled in a subconcern. Another subconcern provides modeling constructs for the component type hierarchy (Provides- and CompletesTypes). Further, the content of the repository concern could be subdivided into an abstract part which contains the abstract class hierarchy and a concrete part, which assigns semantics to the abstract classes. Such a further division could foster reuse in other domains and constrain the direction of dependencies. The classes of the repository constraints originate from the repository, subsystem and entity packages. However, some specific content of the repository package was factored out into other concerns.

From the outside view, it is clear what the main concepts of the repository concern are: components, interfaces and their relations. However, from the inside view, there is an elaborate class hierarchy for classifying different types of components. Further main concepts are: roles and a specialization of the abstract type hierarchy to use operation list signature.

The repository concern is dependent on the composition concern, as it describes how composed elements of the repository are constructed. The variable characterization is also referenced to enable component parameters. There are some problematic outgoing references, which should be reversed. Namely to the reliability and QoS annotations concerns. To form a clean architecture description language, also the dependency to the behavior of components (SEFF) should be reversed, so that the repository is not dependent on behavior but behavior extends or annotates content of the repository concern.

DataTypes This subconcern of the repository concern provides modeling capabilities for custom data types. These data types can then be used in definitions of signature lists for OperationInterfaces. It originates completely from the repository package. Its main concepts are: the PrimitiveDataType (such as integer, char, string), the CompositeDataType (similar to struct concept of object oriented programming) and the CollectionDataType (a set containing multiple instances of the same data type). There are no outgoing dependencies.

Component Type Hierarchy This subconcern of the repository concern contains the classes to specify the component type hierarchy. The whole concern was contained in the repository package. ProvidesComponentTypes are used to declare mandatory provided roles. CompletesComponentTypes are used to constrict required roles. These role constraints have to be fulfilled for a conforms relation to hold. These component types can then be substituted by their conforming components within an composed structure. There are no outgoing dependencies (except to the parent concern), nor problematic incoming ones.

5.4 Composition

The composition concerns defines how composed structures can be constructed: components are put into their context within a composed structure, which results in an assembly context. The roles of the assembly contexts are connected by assembly connectors. The roles of the outer structure are connected to the roles of assembly connectors by delegation connectors. This concern stems solely from the composition package and will therefor be known to Palladio users as the composed structure or system diagram. This concern is dependent on the repository concern, as components and their roles are referenced. There is also a dependency to variable characterization, as component parameters can be set in assembly contexts.

5.5 SEFF

The SEFF concern contains all classes that are necessary to model the behavior of a component method. For example, the behavior can be modeled in terms of internal actions, external actions, loops and branches. Unlike the current PCM meta-model, the SEFF does not allow the modeling of performance or reliability. This will be realized by additional extensions to the SEFF. Another difference to the current meta-model will be that a SEFF is no longer contained within BasicComponents. This will decouple the repository from the SEFF. In addition, this will have the positive effect that concurrent changes on SEFF and the Repository model will no longer overwrite the other change, if saved. Since SEFFs are contained in a BasicComponent the SEFF concern has a outgoing dependency to the repository.

5.6 Usage

The usage concern contains all classes that are necessary to create a usage model. It matches the current usagemodel package in the current Palladio meta-model. The usage only has a dependency to the Repository concern. In particular it needs the OperationSignature and the OperationProvidedRole that are used in the EntryLevelSystemCall.

5.7 Environment

The environment concern contains all classes that are necessary to model the environment of the system in terms of hardware environment and their linking between one another. The concern consists of the performance independent part of the resourceenvironment package from the current Palladio meta-model. Hence, it will contain the classes Environment and Liniking. The environment concern has no outgoing dependencies.

5.8 Allocation

The Allocation concern contains all classes that are necessary to create an Allocation model. Like the usage model, it matches its former PCM package, the allocation. The main concepts are Allocation and AllocationContext. The AllocationContext matches which component is deployed on which hardware resource. Hence, the allocation concern has dependencies to the composition and the environment concerns.

5.9 Performance

The Performance concern enables the modeling of performance information. It has several sub-concerns. Each of the sub-concerns extend other concerns in order to enable the performance annotations that can be used for performance prediction in a later step.

RDSEFF The RDSEFF (Resource Demanding Service Effect Specification) contains all classes to enable the modeling of a SEFF that interacts with resources. It contains all classes from the seff_performance package in the current Palladio meta-model. The main concepts of the RDSEFF are the ResourceDemandingSEFF class itself as well as ResourceCall and ResourceDemandingBehavior. It has outgoing dependencies to the SEFF and Resource Environment and an incoming dependency from the Infrastructure concern.

Resource Environment The Resource environment is a sub-concern of the performance concern. It contains classes that extends the environment model to enable the performance modeling of hardware resources. These resources are used for the performance prediction in a later step. The main concepts of the resource environment are the ResourceEnvironment, which extends environment, and the ResourceContainer. The resource environment has outgoing dependencies to the environment and to the Resource Type.

Resource Type repository The resource type concern is a sub-concern of the performance model. The resource type contains the concrete resources, e.g. CPU and HDDs that can be used in the resource environment. Since the resource type repository only contains resources that are used in the resource environment it does not have any outgoing dependencies.

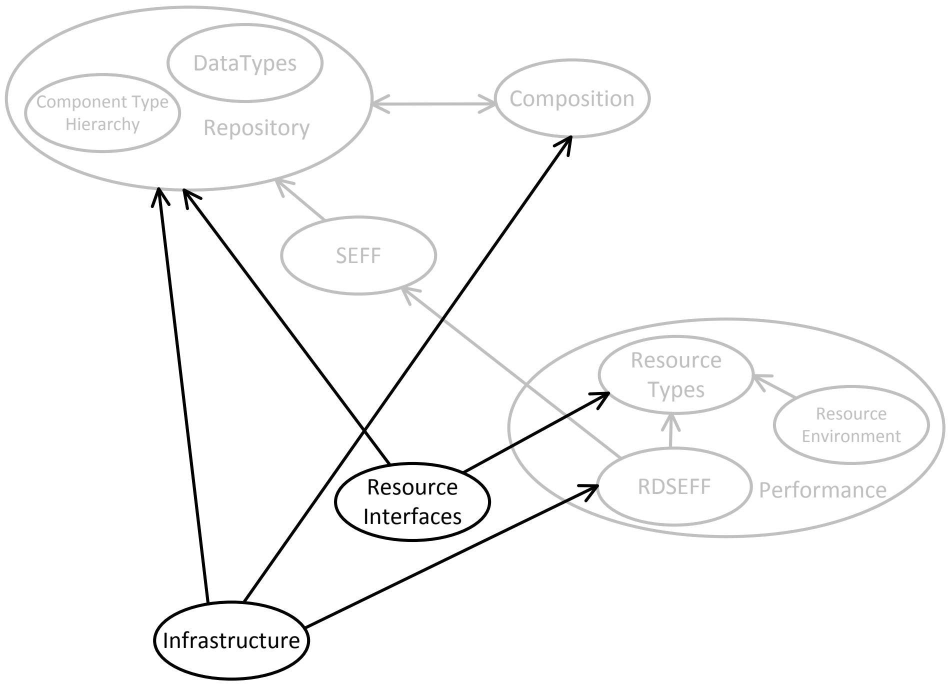

Figure 4: Performance Specific Extending Concerns

5.10 Resource Interface

The resource interface [Hau09] concern extends the resource environment and allows a more detailed performance predictions based on a resource model. The main concepts of the resource interface are ResourceInterface and ResourceSignature. The resource interface has an outgoing dependency to the resource environment. Since the class ResourceSignature uses Parameters from the repository it also has an outgoing dependency to the repository (see Figure 4).

5.11 Infrastructure

The infrastructure concern provides capabilities to model infrastructure software components (e.g. middleware) [Hau09]. Its dependencies are shown in Figure 4. Calls to such infrastructure components have no return values and can be made directly from InternalActions. The infrastructure components therefore provide capabilities to model complex resource demands. The classes infrastructure concern originate in the repository, composition and SEFF package. This subdivision should also be kept in this concern as subconcerns or packages.

The main concepts of this concern are: InfrastructureSignatures, -Interfaces, -Roles, -Connectors and -Calls. Normal and infrastructure Components are distinguished by an attribute in the ImplementsComponentType, this should be replaced by a construct of more annotating nature to be able to factor it out of the repository concern.

The InfrastructureCall has an incoming dependency from SEFF, as it is contained within AbstractInternalControlFlowActions, this should be reversed. Expected outgoing references are therefore: to the repository, because of the inheritance from Interface, Roles, Signature and the annotation of component. There is also a reference to composition, because of the inheritance of the connectors and another reference to SEFF because of the introduction of the InfrastructureCall.

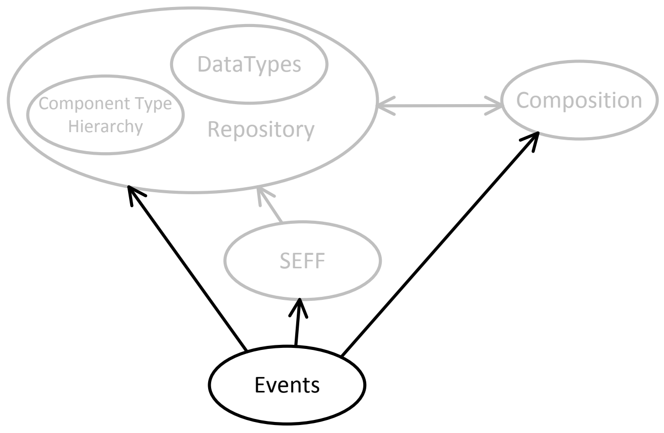

Figure 5: Event Communication Concern

5.12 Events

Currently the Event classes are part of the repository package [Rat13]. For the modularized meta model we propose a own concern for the Event extension of the PCM (see Figure 5. This concern itself will be separated in the three sub-concerns Events-Repository, Events-Composition and Events-SEFF. The Events-Repository contains all classes to model an

Event based system in the repository model, e.g. source and sink roles. The Events-Composition contains the classes that are necessary to compose the events modeled in the repository model. Finally, the Events-SEFF enable the modeling of event based Service Effect Specifications.

The Events-Repository sub-concern has dependency to the repository concern. The Events-Composition sub-concern has dependencies to the Events-Repository and to the composition. The Events-SEFF sub-concern has a dependencies to the Events-Repository and to the SEFF concern.

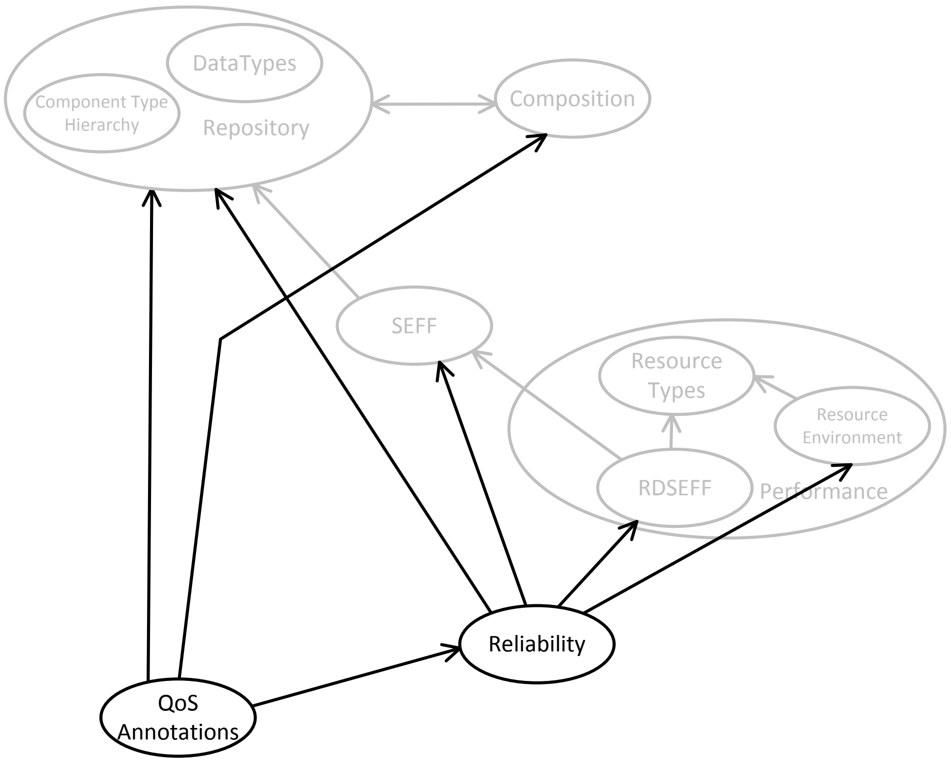

Figure 6: Reliability and QoS Annotations Concerns

5.13 Reliability

The reliability concern enriches the PCM by structure and information which is needed to model and analyze reliability of software systems [Bro12]. Its dependencies are shown in Figure 6. With this extension, InternalActions, hardware resources and external services are allowed to have a chance to fail. This concern stems from the reliability, seff_reliability and resource environment packages. Which should also be reflected in its subconcerns or package structure.

The main concepts of the reliability concern are: The abstract class FailureType and its subclasses: SoftwareInduced-, ResourceTimeout-, HardwareInduced- and NetworkInducedFailureType. The FailureOccurenceDescription class contains a feature probability and a reference to a FailureType. FailureOccurenceDescriptions can be annotated to system external required interfaces or be contained in internal actions within SEFFs. The RecoveryAction represents a recovery point within a SEFF. A RecoveryAction contains multiple RecoveryActionBehaviors, which are RDSEFFs. The ResourceFailureProbabilityAnnotation is a new class which extracts the hard-coded failure rate from resource containers and LinkingResources.

This concern was retroactively and intrusively added to the meta-model. It is cleanly contained within its own packages. However, it has many incoming dependencies, from packages which are more general. These dependencies should be reversed to decouple these packages and their concerns. The reliability concern will then be an annotating extension. These problematic incoming dependencies are:

- A repository contains FailureTypes. These should be contained within an own container in the reliability concern.
- Signature references failure types.
- The ResourceType concern is dependent, as there are bidirectional relations to HardwareInducedFailureType and NetworkInducedFailureType.
- The SEFF concern is dependent on InternalFailureOccurenceDescription, as they are referenced by internal actions.
- Passive resources reference ResourceTimeoutFailureType (which is not consistent with the type level).

All in all, the reliability concern will be dependent on the following concerns: repository, ResourceType, SEFF and RDSEFF. However, it should be reconsidered, if the extension of the resource type concern is really necessary. The resource types are statically instantiated and do not change during modeling. Can a reliability engineer really model something using these relation, which changes the result of the reliability analysis? If the information provided by these relations is purely conceptual, they could be omitted.

5.14 QoS Annotations

This concern it used to apply Quality of Service (QoS) annotations to system external services or components which are not yet fully specified (e.g. Provides- or CompletesTypes). Its dependencies are shown in Figure 6. These annotations are not a general specification but context dependent. I.e. the resource environment is fixed as well as further dependencies and background or concurrent usage overhead. This concern contains two small subconcerns for performance and reliability specific annotations. In the scope of the refactoring they may become subpackages of this concern or subpackages of the reliability and performance concern.

The main concepts are: The QosAnnotations class is just a container, which contains all annotation specifications. SpecifiedQoSAnnotation is an abstract super class for all QoS annotations. It references the role and signature, for which an annotation should be applied. SpecifiedOutputParameterAbstraction provides a variable characterization for a return value. Within the subconcerns, SpecifiedQoSAnnotation is specialized to provide annotations of execution time and failure probability and failure type.

Systems contain QoSAnnotations, this should be removed. A QoSAnnotation should reference a repository. The remainder are outgoing references which are legitimate due to their annotating nature. These dependencies reference the following concerns: repository, composition, variable characterization. Further, the reliability subconcern depends on the reliability concern, as the FailureOccurenceDescriptor is used.

6 Conclusion

During the process of inspecting the meta-model, it became clear to us, that containment of multiple concerns in one package and the distribution of concerns over multiple packages, degrade the understandability of the meta-model. The initial meta-model might have been conceptually well formed. However, later modifications and extensions worsened that design.

Another problem, which stems partly from later modifications and extensions, is that dependencies cannot be properly constrained in the package structure. This makes the meta-model more difficult to maintain, as one has to regard unexpected dependencies. E.g. when extending concerns are hard coded into core concepts, changing the extension has to be done carefully. Also it makes a modularization quite difficult.

Further, the lacking modularization makes the application domain of the meta-model very specific. As the PCM was initially designed for performance modeling and analysis, the quality constructs were hard coded into the foundational constructs. If the current Palladio meta-model is used or extended for other purposes and domains, domain unrelated features would decrease the comprehensibly for the developers.

To solve these problems, we need to modularize and refactor the meta-model and we need to set up conventions for future extensions. In the scope of the modularization, hard-coded parts will be factored out into modules. We hope, that the concerns, which we identified, and their dependencies will provide a good basis for future deliberations. The modularization will also bring some more control over the dependencies, as creating a reference between two meta-models, which are not yet dependent in that direction, requires explicit loading of the referenced meta-model. So, extra consideration will be provoked, when inserting such a new dependency.

Conventions will help to ensure, that most extensions will be made externally and not intrusively. These conventions should also propose and restrict the extension method (e.g. aspect-oriented annotating [JHS+14], profiles [KDH+12]). It may also be beneficial for the understandability of the extensions, that their package structure reflects which concerns of the Palladio meta-model are extended.

7 Continuation of the Migration Process

The immediate next steps will be to discuss and settle for a modularization of the meta-model. We hope that the concerns identified in this paper will provide a solid base for future deliberations. Next, the meta-model will be refactored.

However, to use the modularized meta-model in future releases of the Palladio Bench, further steps have to be performed: A backwards transformation has to be implement, which transforms model instances of the modular meta-model into instances of the classic meta-model. This will enable all the tools to remain functional. The tools which are actively maintained can then be ported to operate on the modular in their own time. Another important step towards release is the implementation of new graphical editors.

The implementation of a forward transformation which transforms instances of the classic meta-model to the modular one, is not immediately mandatory for a release. However, providing such a transformation is nevertheless important, as it migrates legacy model instances.

8 Acknowledgments

This work was partially supported by the Helmholtz Association of German Research Centers.

References

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82:3–22, 2009.

[Bro12] Franz Brosch. *Integrated Software Architecture-Based Reliability Prediction for IT Systems*. PhD thesis, Institut für Programmstrukturen und Datenorganisation (IPD), Karlsruher Institut für Technologie, Karlsruhe, Germany, June 2012.

[Hau09] Michael Hauck. Extending Performance-Oriented Resource Modelling in the Palladio Component Model. Diploma thesis, University of Karlsruhe (TH), Germany, February 2009.

[JHS+14] Reiner Jung, Robert Heinrich, Eric Schmieders, Misha Strittmatter, and Wilhelm Hasselbring. A Method for Aspect-oriented Meta-Model Evolution. In *Proceedings of the*

2Nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling, VAO '14, pages 19:19–19:22, New York, NY, USA, July 2014. ACM.

[KDH$^+$12] 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 Steffen Becker, Jens Happe, Anne Koziolek, and Ralf Reussner, editors, *Palladio Days 2012 Proceedings (appeared as technical report)*, Karlsruhe Reports in Informatics ; 2012,21, pages 7–15, Karlsruhe, 2012. KIT, Faculty of Informatics.

[Koz08] Heiko Koziolek. *Parameter Dependencies for Reusable Performance Specifications of Software Components*. PhD thesis, University of Oldenburg, 2008.

[Rat13] Christoph Rathfelder. *Modelling Event-Based Interactions in Component-Based Architectures for Quantitative System Evaluation*, volume 10 of *The Karlsruhe Series on Software Design and Quality*. KIT Scientific Publishing, Karlsruhe, Germany, 2013.

[RBB$^+$11] Ralf Reussner, Steffen Becker, Erik Burger, Jens Happe, Michael Hauck, Anne Koziolek, Heiko Koziolek, Klaus Krogmann, and Michael Kuperberg. The Palladio Component Model. Technical report, KIT, Fakultät für Informatik, Karlsruhe, 2011.

[SBPM08] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework*. Eclipse series. Addison-Wesley Longman, Amsterdam, second revised edition, December 2008.

[SMRL13] Misha Strittmatter, Philipp Merkle, Andreas Rentschler, and Michael Langhammer. Towards a Modular Palladio Component Model. In Steffen Becker, Wilhelm Hasselbring, André van Hoorn, and Ralf Reussner, editors, *Proceedings of the Symposium on Software Performance: Joint Kieker/Palladio Days, KPDAYS '13*, volume 1083, pages 49–58. CEUR Workshop Proceedings, November 2013.