A Performance Model Management Repository Based on the Palladio Component Model

Alexandru Danciu, Andreas Brunnert
fortiss GmbH
Guerickestr. 25
80805 München, Germany
{danciu, brunnert}@fortiss.org

Helmut Krcmar
Technical University of Munich (TUM)
Boltzmannstr. 3
85748 Garching, Germany
krcmar@in.tum.de

ABSTRACT

Applying performance models to evaluate component-based enterprise applications in practice is becoming more and more difficult with an increasing organizational complexity. Components can be governed by different organizational units, are subject to a continuous shift between life cycle phases, and exhibit diverging release levels. Creating and maintaining performance models based on the composition of these components can, therefore, represent an elaborate task. A Performance Model Management Repository (PMMR) supports managing performance models in complex enterprise environments. This paper presents the implementation of a PMMR based on the Palladio Component Model (PCM). The PCM meta-model is extended to enable managing and maintaining multiple versions of components and their interfaces. Furthermore, resource demand specifications derived from different hardware environments are integrated into the meta-model. The Palladio-Bench is extended for persisting PMMR elements to EMFStore.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: measurement techniques, modeling techniques

Keywords

Performance Model Repository, Performance Evaluation, Palladio Component Model, Enterprise Application, Component-based Performance Model

1. INTRODUCTION

Managing and maintaining performance models of component-based enterprise applications is elaborate and influences the adoption rate of performance models in the industry. Modeling components and their relationships becomes increasingly difficult due to constant evolution in terms of architecture, governance and life cycle [4]. Complex system of systems architectures imply an increased number of components and dependencies. Individual components can be under the control of different teams within one or more organizations. Components can adhere to diverging release cycles or be situated in different life cycle phases. As a consequence, performance models need to support a concurrent access of multiple teams and users. The concept of the unique identity of a component is replaced by the existence of multiple versions of a component at the same time. Also, resource demands of components specified in performance models need to support their representation for different hardware environments. The Performance Model Management Repository (PMMR) proposed in [3] aims at providing a solution for these challenges. The envisioned solution uses the Palladio Component Model (PCM) as meta-model employing the Palladio-Bench as PMMR client and EMFStore[1] as PMMR server. However, the current implementation of PCM doesn't support the core features of the PMMR. PCM model artifacts are stored in flat files making a concurrent access difficult. PCM repository components and interfaces don't provide any versioning mechanism. Resource demands associated with repository components are specified for a specific hardware environment. This work presents extensions to the PCM meta-model and the Palladio-Bench required by a PMMR.

2. EXTENSIONS TO THE PALLADIO COMPONENT MODEL

Extensions to the PCM meta-model consist of classes added to the underlying Eclipse Modeling Framework (EMF)[2] model. All extensions are grouped in a new subpackage called *pmmr*. Thus, existing PCM models and editors don't need to be adapted. This work is based to the PCM version 3.4.1.

2.1 Versioning of Components and Interfaces

The evolution and versioning of software systems constitutes the focus of software configuration management (SCM). In SCM the term *version* is used on different abstraction levels. Version control systems such as Subversion[3] or EMFStore record changes to software artifacts or model elements and produce implicit versions [5]. In contrast, explicit versions of software artifacts are defined as part of a release management [8]. With regard to the nature of a change, interface and implementation versioning are distinguished [2]. Interface versioning reflects changes in the interaction of

[1]http://www.eclipse.org/emfstore/
[2]http://www.eclipse.org/modeling/emf/
[3]https://subversion.apache.org

components, while implementation versioning reflects changes in the source code of a component. A PMMR requires the existence of explicit versions to support specifying and maintaining different releases of components and interfaces simultaneously within an enterprise, as depicted in Figure 1.

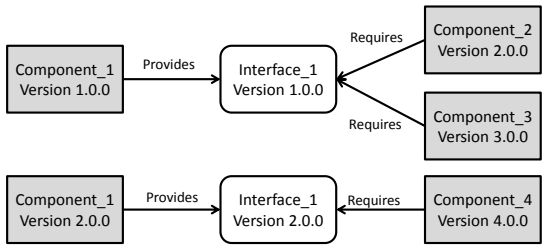

Figure 1: Example for versioning of components and interfaces (adapted from [8])

The extensions applied to the PCM meta-model for supporting the versioning of components and interfaces in a PMMR are depicted in Figure 2. The contents of a PMMR are subordinated to an *Application* element. Component and interface version specifications are organized in a hierarchical structure of *Package* elements. Packages represent a set of related version specifications and can be defined arbitrarily by PMMR users. Thus, version specifications can be grouped by any criteria, such as release, component name or organizational unit. *ComponentVersion* and *InterfaceVersion* represent the specification of a version and inherit common attributes from the abstract class *VersionSpec*. Version specifications contain information such as a version identifier, references to previous or subsequent versions and references to branched or merged versions. The usage of generic types enforces that these relationships can exist only between elements of the same class.

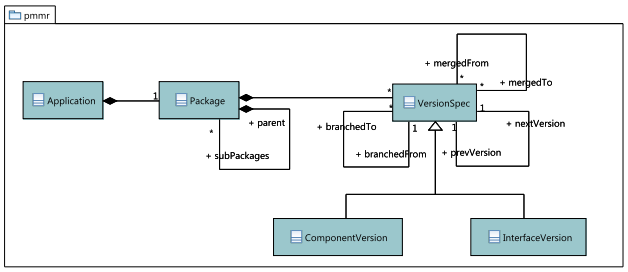

Figure 2: PCM meta-model extensions for managing component and interface versions

2.2 Central Repository for Components

The management of PCM components constitutes the main functionality of a PMMR and its implementation is highly influenced by technical limitations imposed by the PCM meta-model, the Palladio-Bench and EMFStore. Components are represented in PCM as *BasicComponent* elements and interfaces as *OperationInterface* elements which are both contained in a *Repository*. As a result, these elements cannot be contained by any other class in an EMF model. However, standard editors generated by EMF only support creating and displaying these elements when a containment relationship exists. Therefore, the existing container has to be reused to enable proper editing components and interfaces.

Figure 3 depicts the extensions (represented as colored boxes) added to existing PCM meta-model elements (represented as white boxes). *ComponentVersion* and *InterfaceVersion* extend the existing *BasicComponent* and *OperationInterface* elements. Therefore, these elements combine both the implementation and the versioning of components and interfaces. Because components and interfaces can be subsumed by merging multiple versions to one entity or divided by branching of one version into multiple entities, the identity of components and interfaces can change over time. The meta-model extension does, therefore, not support a version-independent entity representing the identity of components or interfaces.

The *Package* element extends the existing *Repository* element. Components and interface implementations can, thus, be organized in a hierarchical package structure avoiding storing them in one large collection.

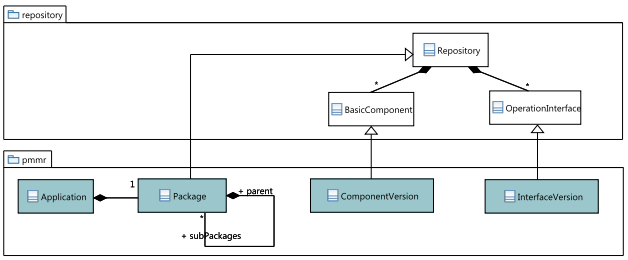

Figure 3: PCM meta-model extensions for managing components and interfaces in a PMMR

PMMR contents are persisted in EMFStore and are organized in projects. Each project consists of an *Application* element and its contained elements. The Palladio-Bench is extended to support connecting to EMFStore for checking out local copies of projects. A user management for restricting the access to the PMMR is also provided. Changes applied to the contents of local projects are tracked and can be committed to the repository server by users.

Another implication of the existing containment relationship between *BasicComponent* and *Repository* is that a component instance can be contained by only one repository instance. Thus, the same component instance managed by a PMMR cannot be added to any other local PCM repository models for conducting simulations, which is contrary to the basic idea of a PMMR. One solution for this would be to implement a wrapper class which is added to each repository instance and passes attribute values between the caller and the component instance. However, due to the containment relationship between a component and its subordinated *ServiceEffectSpecification* elements, the serialization of the wrapper object is not performed correctly. Another drawback to this approach is that all methods of the generated wrapper implementation have to be manually overwritten. Our proposed solution to support adding a component instance to multiple repository models, is to create object clones and keeping them synchronized with the PMMR. Updates to a component instance are propagated to all affected PCM repository models. Editing is, however, only supported within the PMMR project.

2.3 Handling Resource Demands

Components managed by a PMMR can specify resource demands measured in a specific hardware environment. Users of the PMMR must, however, be able to use these components for any performance evaluations which specify a different hardware environment. Therefore, resource demands of components stored in a PMMR are specified relative to a hardware benchmark score. This enables setting heterogeneous hardware environments in relation and to convert resource demands. During the check-out of a component version, resource demands are converted with respect to a target hardware environment.

Each component version managed by a PMMR can specify resource demands for several *ProcessingResourceType* objects, such as central processing unit (CPU) and hard disk drive (HDD). The Palladio-Bench provides a predefined set of instances of this class which are used for the specification of resource demands. As a result, a PMMR should not manage own instances, otherwise a mapping to the predefined resource types would be necessary during each check-in and check-out. Instead, the predefined instances are imported and reused.

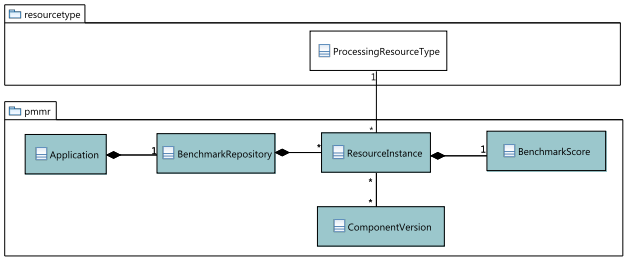

Figure 4: PCM meta-model extensions for managing resource demands

Figure 4 depicts the extensions applied to the PCM meta-model for handling resource demands from different hardware environments. The PMMR contains a set of *ResourceInstance* elements representing specific hardware resources, of a certain processing resource type such as CPU, which are manufactured by a specific hardware vendor. Each resource instance contains a *BenchmarkScore* element. A benchmark score specifies the performance achieved by a specific hardware resource while executing a specific task. *ResourceInstance* objects are contained in the *BenchmarkRepository*. During check-in of a new *ComponentVersion* object, the user has to specify resource instances describing the hardware environment where resource demands were collected. When checking out a component to a local PCM repository model the benchmark scores are used to convert resource demands with respect to a specified target hardware environment.

3. RELATED WORK

A number of approaches propose means for versioning model artifacts. Existing approaches address both models in general [1] and domain specific models. An approach for versioning of Unified Modeling Language (UML) models is proposed by Murta et al. [6]. Roshandel et al. [7] propose an approach for managing the evolution of software architectures. However, none of the approaches address requirements arising from the management of performance models.

The *Performance Knowledge Base (PKB)* proposed by Woodside et al. [9] is closely related to the PMMR. The PKB intends to store knowledge resulting from performance evaluations over time and over system versions. Instead of managing performance models, the PKB is intended to store measurements and prediction results. According to the authors, such a PKB should be able to construct performance models on demand.

4. CONCLUSION AND FUTURE WORK

In this work, we have presented extensions to the PCM meta-model required for implementing the PMMR proposed in [3]. These extensions enable managing versions of components and interfaces in a central repository server and handling resource demands for different hardware environments.

The proposed implementation of a PMMR employs several existing methods, techniques and tools which were already evaluated for similar purposes. Future work will focus on evaluating the integrated solution in an experimental setup observing the interaction with users to validate the feasibility of the approach. Additionally, the integration of the PMMR with approaches producing or consuming PCM models will be addressed.

5. REFERENCES

[1] K. Altmanninger, M. Seidl, and M. Wimmer. A survey on model versioning approaches. *International Journal of Web Information Systems*, 5(3):271–304, 2009.

[2] V. Andrikopoulos, S. Benbernou, and M. Papazoglou. On the evolution of services. *IEEE Transactions on Software Engineering*, 38(3):609–628, May 2012.

[3] A. Brunnert, A. Danciu, and H. Krcmar. Towards a performance model management repository for component-based enterprise applications. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, pages 321–324, New York, NY, USA, 2015. ACM.

[4] A. Brunnert, C. Vögele, A. Danciu, M. Pfaff, M. Mayer, and H. Krcmar. Performance management work. *Business & Information Systems Engineering*, 6(3):177–179, 2014.

[5] M. Kögel. *Operation-based Model Evolution.* PhD thesis, Technische Universität München, 2011.

[6] L. Murta, C. Corrêa, J. a. G. Prudêncio, and C. Werner. Towards odyssey-vcs 2: Improvements over a uml-based version control system. In *Proceedings of the 2008 International Workshop on Comparison and Versioning of Software Models*, CVSM '08, pages 25–30, New York, NY, USA, 2008. ACM.

[7] R. Roshandel, A. V. D. Hoek, M. Mikic-Rakic, and N. Medvidovic. Mae—a system model and environment for managing architectural evolution. *ACM Transactions on Software Engineering and Methodology*, 13(2):240–276, Apr. 2004.

[8] A. Stuckenholz. Component evolution and versioning state of the art. *SIGSOFT Software Engineering Notes*, 30(1):7–, Jan. 2005.

[9] M. Woodside, G. Franks, and D. C. Petriu. The future of software performance engineering. In *Future of Software Engineering (FOSE)*, pages 171–187, Minneapolis, MN, USA, 2007.