Automated Transformation of Descartes Modeling Language to Palladio Component Model

Jürgen Walter
University Würzburg Chair for Computer Science II
Am Hubland
D-97074 Würzburg, Germany
juergen.walter@uni-wuerzburg.de

Simon Eismann
University Würzburg Chair for Computer Science II
Am Hubland
D-97074 Würzburg, Germany
adrian.hildebrandt@stud-mail.uni-wuerzburg.de

Adrian Hildebrandt
University Würzburg Chair for Computer Science II
Am Hubland
D-97074 Würzburg, Germany
adrian.hildebrandt@stud-mail.uni-wuerzburg.de

ABSTRACT

Model-based performance predictions and reconfigurations enable optimizing resource efficiency while ensuring that Quality-of-Service demands are met in today's complex IT-systems. The Descartes Modeling Language (DML) and the Palladio Component Model (PCM) are two architectural performance modeling formalisms applied in this context. This paper compares DML to PCM concerning similarities, differences and semantic gaps. Based on this, we propose a mapping from DML to PCM for which we implemented a tool realizing an automated transformation.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling techniques*; D.2 [**Software Engineering**]

Keywords

Descartes modeling language, Palladio component model, Model-to-model transformation, Performance modeling, Performance prediction

1. INTRODUCTION

Todays IT service providers are driven by the pressure to improve the efficiency of their systems, e.g., by sharing resources, and to reduce their operating costs while ensuring Quality-of-Service demands. Model-based performance predictions and reconfigurations enable to detect and prevent performance problems and remain resource efficient at the same time. Existing formalisms range from analytical models (e.g. queueing networks) to more complex architectural performance models which capture architecture and resource landscape as well. Automated transformations from architectural to analytical models offer multiple was to analyze the system. DML and PCM are both architectural performance modeling formalisms to describe component-based systems. Both formalisms can be applied at design time and runtime scenarios. However, from the historic perspective, PCM initially has been designed for design time scenarios whereas DML targets reconfiguration at runtime. While the general approach is similar, the formalisms differ in a lot of details. For a practitioner it is a huge effort to understand those differences. Not knowing the different features at the beginning of a project, the current approach is to choose either PCM or DML. A transformation enables the use the strengths of both formalisms. In this paper, we explain a mapping from DML to PCM which we use for an automated model-to-model transformation.

2. COMPARISON OF DML TO PCM

The high-level architectures of DML and PCM —opposed in Figure 1— have a lot in common. Both include metamodels for resource environment, components and their behavior, deployment, and usage profile. From this high-level

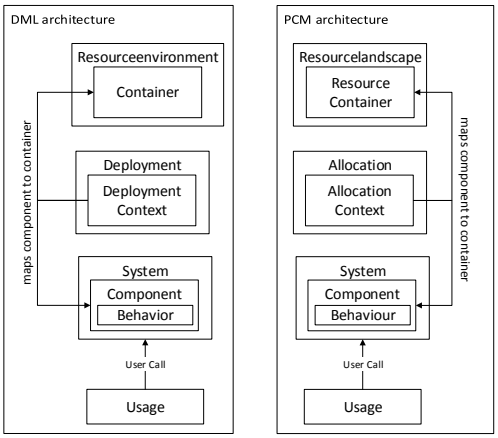

Figure 1: Comparison between DML and PCM Architectures

perspective, their submodels can be matched directly. However, on the sub-level there are the conceptual differences. According to Kounev et al. [2], the main advantages of DML compared to PCM can be summarized as follows:

- **Service Behavior Modeling** In contrast to PCM,

DML supports modeling multiple service behavior abstractions of different granularity for the same service. This allows for flexible performance predictions, ranging from quick bounds analysis to detailed model simulations.

  - **Empirical Parameter Characterizations** PCM utilizes explicit specifications (i.e., explicit mathematical functions) to describe dependencies between model parameters. DML additionally supports characterization of parameter dependencies based on monitoring data that is collected at runtime.
  - **Empirical Behaviour** DML supports to model parameter characterizations that are dependent on the component assembly, i.e., flexible characterizations for different component instances of the same component type. In PCM, parameter characterizations are fixed for the surrounding component type. Differences between component instances are intended to be captured by explicit parameterizations. Thus, in run-time scenarios where representative monitoring data is available, only DML offers a convenient approach to make use of such monitoring data for parameter characterization.
  - **Indirect Parameter Dependencies** PCM supports modeling service behavior depending on service input parameters passed upon service invocation. However, the behavior of software components is often dependent on parameters that are not available as direct service input parameters. DML provides means to pass such influencing parameters to the service models whose behavior is influenced.

- **Resource Landscape** DML supports the modeling of complex multi-layered resource landscapes. Furthermore, it provides a template modeling mechanism that eases the re-use of resource specifications among several resource containers. This is particularly useful to model virtualization layers, to specify Virtual Machines (VMs) that stem from the same VM image.
- **Adaption Points** DML provides means to specify adaptation points as well as adaptation processes. There is no representation for this in PCM so far.

Moreover, there are semantic gaps. Some are introduced through the difference of American English used in DML and British English used in PCM. Furthermore, the wording differs at some points. For example, a DML `Container`, is a PCM `ResourceContainer`. `Deployment` corresponds to `Allocation` in PCM. `ServiceEffectSpecification` (SEFF) is named `FineGrainedBehavior` in DML. These gaps are a minor challenge for transformation, however necessary to understand it. More details about the formalisms can be found in the respective technical reports for DML [2] and PCM [3].

3. TRANSFORMATION

In the following we describe a mapping from DML to PCM. In this paper, we focus on service behavior and resource modeling. A more detailed view on the transformation is provided in [1].

3.1 Service Behavior Modeling

In DML, the services of a component can be expressed using three abstraction levels: fine-grained behavior, coarse-grained behavior and black-box behavior. Fine-grained behavior is the most detailed, describing external calls, internal resource consumption and control flow. Coarse-grained behavior describes external calls and internal resource consumption but does not model control flow. Black box behavior only describes how much time a component takes for the specified behavior. In contrast, PCM provides only a single level of behavior abstraction, the Service Effect Specification (SEFF), which is similar to the fine-grained behavior. For a transformation all three abstraction levels of behavior in DML need to be mapped to the single abstraction level of a `ResourceDemandingSEFF` in PCM. In case service descriptions are available on multiple levels, the most detailed behavior description shall be used for transformation. The transformation first searches for fine-grained behavior description. In case there is no fine-grained behavior we use the coarse-grained behavior. The black-box behavior is used when no other behavior description is available.

The current transformation is limited to explicit relationships, we do not transform empirical parameter dependencies to PCM. A preceding step would be the extraction of explicit StoEX from measured data which was out of scope. Besides that, the transformation of the fine-grained behavior to a `ResourceDemandingSEFF` is straight forward. For most elements in the fine-grained behavior there exists a similar concept in the SEFF of PCM.

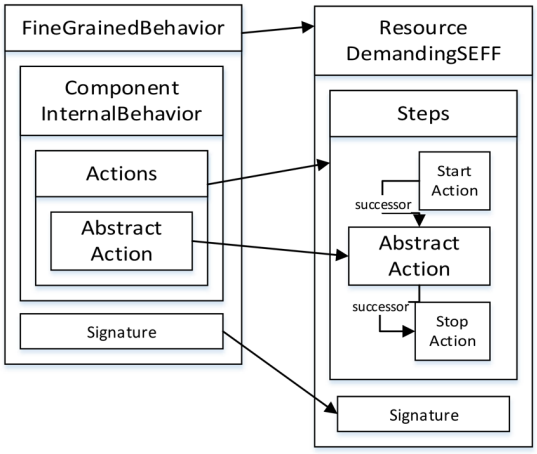

Figure 2: Fine Grained Behavior Transformation

A CoarseGrained-Behavior is also transformed into a ResourceDemandingSEFF. Each ExternalCallFrequency is transformed into one BranchAction with two branches of the type ProbabilisticBranchTransition. One branch contains nothing but the ExternalCall, while the other branch is empty. This way the probability of the ExternalCalls execution depends solely on the BranchProbabilities. The ExternalCallAction of DML uses the same meta-model to describe the ExternalCall as the ExternalCallFrequency and therefore the transformation yields identical behavior.

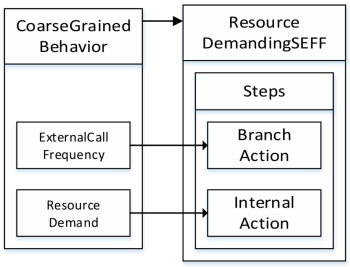

Figure 3: Coarse Grained Behavior Transformation

While black box behavior of DML does not require a resource specification, this is necessary for SEFF specification. For the PCM model this means: The `ResourceContainer` which hosts the component requires an artificial delay `ResourceType`. The transformation creates only one shared delay resource per host which is reused.

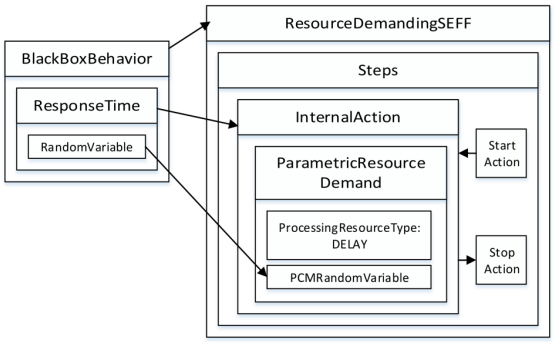

Figure 4: Black Box Behavior Transformation

3.2 Resource Landscape and Deployment

In contrast to PCM, the resource landscape model of DML uses hierarchical resource containers. Consequently, hierarchical information gets lost at the transformation, as depicted in Figure 5. Besides hardware resources, DML enables to model virtual resources which again have no representation in the PCM formalism. The transformed model loses the information whether a component is deployed on a virtual machine or not. The transformed PCM model only holds a mapping to the VM-holding hardware.

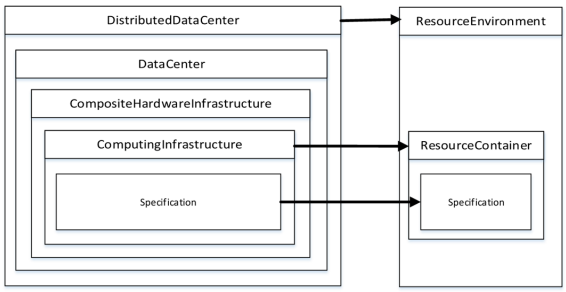

Figure 5: Resource Landscape Transformation

The `DeploymentContext` maps an `AssemblyContext` (instantiation of a software component) to a hardware `Container`. The container can be either a `ComputingInfrastructure` or a `RuntimeEnvironment`. In case a component is mapped to a `ComputingInfrastructure` the mapping is identical. If the `DeploymentContext` contains a reference to a virtual resource specified as `RuntimeEnvironment`, the mapping has to be adapted to the `ComputingInfrastructure` the original virtual resource is part of.

4. CONCLUSIONS

This paper presents an automated model-to-model transformation of DML to PCM. While the majority of DML can be transformed preserving equal behavior, some information stored in DML models have no representation in PCM. Compared to a manual transformation an automated transformation is more convenient, faster and less error prone. The ability to quickly transform models enables a better comparison between models and may cause vice-versa stimulation of formalism and tool development. Ideas for future work are a transformation from PCM to DML and the investigation of a shared core functionality of both formalisms. The transformation code is available at https://se3.informatik.uni-wuerzburg.de/descartes/dml2pcm.

5. REFERENCES

[1] A. Hildebrandt. Automated transformation of descartes modeling language to palladio component models. BachelorThesis, University of Würzburg, Am Hubland, Informatikgebäude, 97074 Würzburg, Germany, 2015.

[2] S. Kounev, F. Brosig, and N. Huber. The Descartes Modeling Language. Technical report, Department of Computer Science, University of Wuerzburg, October 2014.

[3] R. Reussner, S. Becker, E. Burger, J. Happe, M. Hauck, A. Koziolek, H. Koziolek, K. Krogmann, and M. Kuperberg. The Palladio Component Model. Technical report, KIT, Fakultät für Informatik, Karlsruhe, 2011.