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**]

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. *Descartes Modeling Language* (DML) [5] and *Palladio Component Model* PCM [1, 6]. 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. For a practitioner, it is a huge effort to understand the differences between these formalisms. In this paper, we explain a mapping from DML to PCM which has been implemented as an automated model-to-model transformation.The transformation offers the following advantages:

- **Reusability** Analysis approaches and tooling (simulation, graphical editors, design space exploration) of PCM can be reused, even if some information about the model structure is lost after transformation.
- **Comparability** The ability to quickly transform models enables a better comparison between models and may cause vice-versa stimulation of formalism and tool development.
- **Flexibility** The transformation offers the flexibility to switch formalisms easily. PCM practitioners can explore DML case studies within familiar tooling.

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

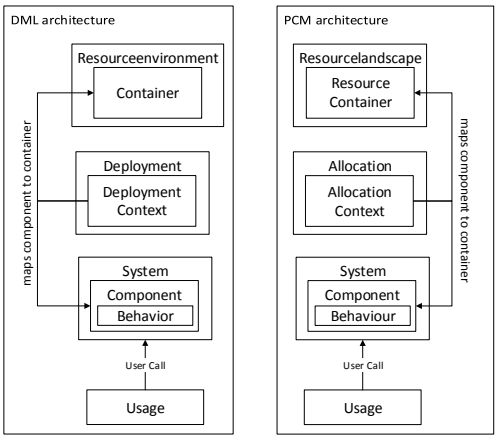

Figure 1: Comparison between DML and PCM Architectures

perspective, their submodels can be matched directly. How-

ever, on the sub-level there are the conceptual differences. According to Kounev et al. [5], the main advantages of DML compared to PCM can be summarized as follows:

- **Service Abstractions** In contrast to PCM, DML supports modeling multiple service behavior abstractions of different granularity for the same service. This allows for building models having constrained input data. Furthermore, it offers flexible performance predictions, ranging from quick bounds analysis to detailed model simulations.
- **Empirical Parameter Characterizations** DML supports characterization of dependencies between model parameters empirically using monitoring data that is collected at run-time (i.e., time series data) and does not require an extraction of explicit specifications (i.e., mathematical functions) like in PCM.
- **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 runtime 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 that stem from the same image.
- **Adaptation Points** DML provides means to specify adaptation points as well as adaptation processes [3, 4]. There is no representation for this in PCM so far.

3. TRANSFORMATION

In the following we describe a mapping from DML to PCM, focusing on service behavior and resource modeling. A more detailed mapping is available in [2].

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). 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.

For DML and PCM, the description of service behavior in includes parametric dependencies. DML offers to store parametric dependencies explicitly using stochastic expressions or implicitly using empirical parameter dependencies from observations (i.e., time series data). Currently, we do not transform empirical parameter dependencies to a explicit parameter dependencies (which is the only description PCM offers). Adding a transformation for empirical descriptions would include an automated extraction of explicit dependencies from measured time series data. Besides parametric dependencies, the transformation of the fine-grained behavior to a `ResourceDemandingSEFF` is straight forward. For most elements in the fine-grained behavior exists a similar concept in the `SEFF` of PCM.

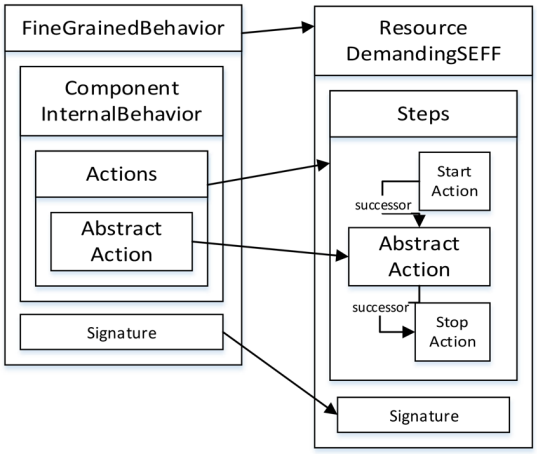

Figure 2: Fine-Grained Behavior Transformation

The coarse-grained behavior abstracts control flow and specifies only the frequency of external calls and resource demands. It has no direct representation in PCM. While low-level elements are similar, the high level integration differs. At first, DML's `CoarseGrainedBehavior` anchor element is transformed into a `ResourceDemandingSEFF`. Then, each external call, modeled as `ExternalCallFrequency`, has to be replaced by multiple PCM elements. It is transformed into one `BranchAction` with two branches of the type `ProbabilisticBranchTransition`. One branch contains the `ExternalCall`, while the other branch is empty. The resulting probability of the `ExternalCalls` execution depends solely on the `BranchProbabilities`. Through this, we can remodel DML behavior using PCM elements. On the lower level, the `ExternalCall` of DML corresponds to the `ExternalCallAction` in PCM. Altogether, the transformation of coarse-grained description yields identical behavior.

The black-box behavior in DML is based on the idea of very

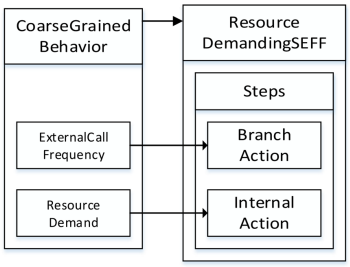

Figure 3: Coarse-Grained Behavior Transformation

limited observations and has no knowledge about resources used internally. While `BlackBoxBehavior` does not require a resource specification, this is necessary for the SEFF. Hence, the PCM `ResourceContainer` which hosts the component requires a `ResourceSpecification` with `DELAY` resource type without DML representation. To avoid an explosion of the number of newly introduced resources, the transformation creates a single shared delay resource per host which is reused for multiple service behavior descriptions.

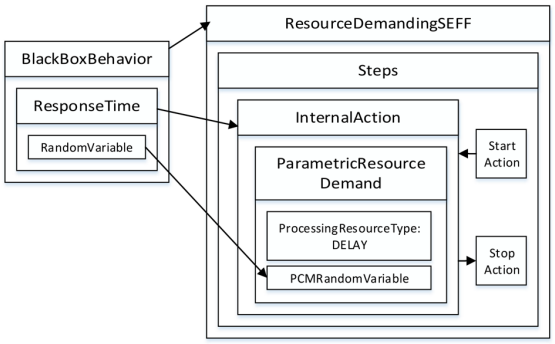

Figure 4: Black-Box Behavior Transformation

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 of software sublayers (e.g., virtual machines). The transformed PCM model only holds a mapping to the hardware on which the virtual machine is deployed. 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.

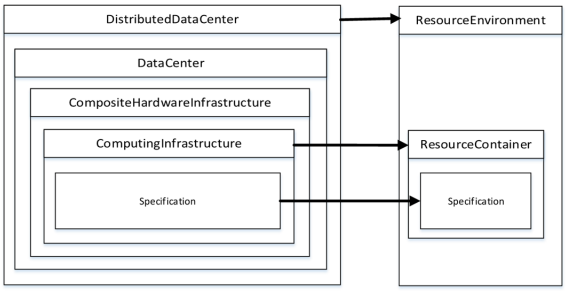

Figure 5: Resource Landscape Transformation

4. CONCLUSIONS

This paper presents a short overview of 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. The ability to quickly transform models enables a better comparison between models and may cause vice-versa stimulation of formalism and tool development. Further, the transformation offers flexibility and reuse of existing tooling. 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] S. Becker, H. Koziolek, and R. Reussner. The Palladio component model for Model-Driven Performance Prediction. *Elsevier Journal of Systems and Software (JSS)*, 82(1):3–22, 2009.

[2] A. Hildebrandt. Automated Transformation of Descartes Modeling Language to Palladio Component Models. Bachelor Thesis, University of Würzburg, 2015.

[3] N. Huber, A. van Hoorn, A. Koziolek, F. Brosig, and S. Kounev. S/T/A: Meta-Modeling Run-Time Adaptation in Component-Based System Architectures. In *Proceedings of the 9th IEEE International Conference on e-Business Engineering (ICEBE 2012)*, pages 70–77. IEEE, September 2012.

[4] N. Huber, J. Walter, M. Bär, and S. Kounev. Model-based Autonomic and Performance-aware System Adaptation in Heterogeneous Resource Environments: A Case Study. 2015.

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

[6] 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.