Design Space Exploration through Interactive Model Mappings for UML-based Specifications

Tim Schattkowsky[1], Achim Rettberg[2], Rainer Dömer[3]
[1] Paderborn University, Germany
[2] C-LAB, Germany
[3] University of Cal. at Irvine, USA

1 Introduction

In previous work [3], we have introduced the Abstract Execution Platform (AEP) as an UML-based design approach for the design of embedded hardware or software systems. This approach is largely based on UML 2.0 Activities as a data and control flow oriented model of computation. The resulting specifications have to be transformed to platform specific code, e.g., HDL for a specific FPGA family. During this transformation, the functional system description is mapped to the available resources, e.g., logical functions within LUTs, memories and existing functional units (like multipliers).

The design space for an AEP solution is determined by the possible mappings to available resources and the application of general patterns like multiplexing. Thus, we introduce an interactive mapping approach based on the application of graph transformations to enable design space exploration in the context of our model-based AEP design approach.

2 Interactive Model Mappings in the Design Process

A complete abstract AEP system specification is already executable as a software system. Such a specification also conforms to the AEP SoC Profile [1] if it does not employ any constructs not allowed by the SoC profile. However, this is just the starting point for refining the specification to contain explicit information about how the employed constructs have to be mapped to different possible implementations.

In our approach, design alternatives are the results of the application of different chains of model transformations leading from a functional system specification to a final platform specific design (see Figure 1). Starting with a complete AEP-based system specification, implementation alternatives of the defined functional blocks can be interactively selected based on libraries of available model transformations.

In the transformation libraries, the mappings between function blocks and specific implementation methods are defined as graph transformation rules. For such rules, the left-hand side defines the functional block to be matched while the right-hand side defines a correspondence either consisting solely of AEP language elements or new language elements that are interpreted by the platform specific model compiler

employed to create the final synthesizable HDL. For this, the original behavior model is implicitly transformed to a canonical form of an extended UML Activity where all expressions are completely expanded. This enables the matching of single language elements such as multiplications while maintaining the original UML syntax for Activities at the model level. The graph transformation rules are based on the graph transformation approach presented in [2].

AEP profiles define syntactic subsets of the AEP for application in a more specialized domain like SoC modeling. Syntactically, these are just UML profiles. The transformation libraries in our approach are defined in the context of the AEP or an AEP profile. The resulting metamodel effectively forms the type graph for the transformations in such a library. For our approach, both left-hand side and right-hand side have to conform to the same metamodel. Thus, the libraries contain model-to-model transformation resulting in a model conforming to this metamodel.

A particular model may conform to several nested AEP profiles, e.g., the SoC profile and a platform specific profile. As a result, transformations from a more abstract library, e.g., the AEP-level transformations applied to transform the Base Design to Design C in our example, may cause a model to be elevated to a more abstract level of design. Thus, the selection of the considered libraries during the interactive mapping process already determines the level of abstraction of the final design alternative.

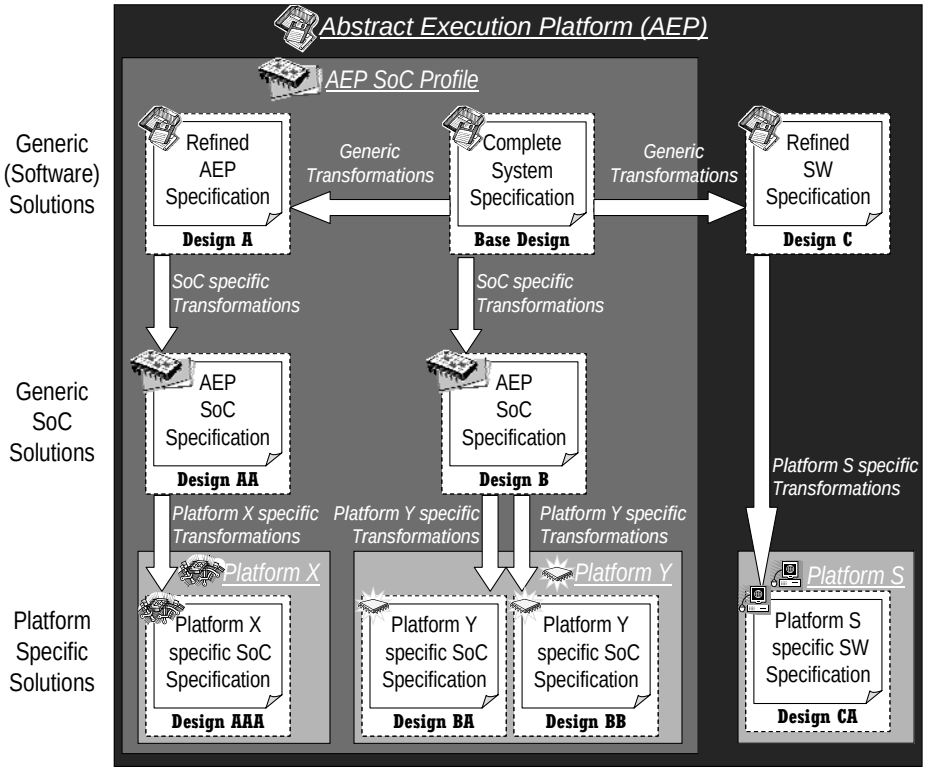

Figure 1: Interactive Model Mappings in AEP-based SoC Design

For the exploration of SoC design alternatives, transformation libraries at three different levels of abstraction are employed. First, there is a generic transformation library containing model transformations introducing AEP-level refinement like the explicit unrolling of loops or inlining of functions. These concepts are generally applicable to both hardware and software systems, as the result of these transformations can be expressed as a modified AEP specification enforcing a certain implementation style on a construct. Thus, it enables the specification of the handling of the usual time/space tradeoffs and generic optimizations. These transformations are employed during the generic mapping process, where the designer can apply these transformations directly to the original functional specifications. The resulting specifications are still valid generic AEP specifications at roughly the same level of abstraction. As a second step, the design may apply transformations belonging to the SoC profile to enforce hardware specific design decisions like the use of multiplexing, replication and pipelining. Finally, platform specific resources can be exploited through a platform specific transformation library containing transformation rules for mapping certain resources like predefined functional units (e.g. multipliers), different memories or logic functions implemented in a LUT of an FPGA.

However, even for the same functional block, the mapping to available platform resources has often to be done at the instance level to cope with limited resources. Thus, especially for the SoC profile, the original specification can be automatically transformed into a single instance level model. This essentially enables the application of the same transformation rules at the instance level.

The final result of the interactive mapping process is a set of design alternatives described as chains of transformations from the abstract Base Design. Tools can maintain these design alternatives by saving the actual matches leading to the application of the transformation rules. Thus, all design alternatives can be managed and considered in later stages of the design flow, e.g., simulation. Furthermore, consistency between the evolving Base Design and the already defined design alternatives can be preserved while the specification evolves.

References

[1] Tim Schattkowsky, Jan Hendrik Hausmann, Gregor Engels: Using UML Activities for System-on-Chip Design and Synthesis. In Proc. ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2006), Genova, Italy, October 2006, LNCS, Springer, 2006 (to appear).

[2] Tim Schattkowsky, Wolfgang Müller: Transformation of UML StateMachines for Direct Execution. In Proc. 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), 2005.

[3] Tim Schattkowsky, Wolfgang Müller, and Achim Rettberg: A Generic Model Execution Platform for the Design of Hardware and Software. In G. Martin, W. Müller (eds.): UML for SoC Design. Kluwer, 2005.