UML-Based Development of Applications for Run-Time Reconfigurable Architectures

Thomas Beierlein\textsuperscript{1}, Dominik Fröhlich\textsuperscript{1,2}, and Bernd Steinbach\textsuperscript{2}

\textsuperscript{1} Hochschule Mittweida  
Dept. Information Technology and Electrotechnique  
Germany  
\{tb | dfroehli\}@htwm.de

\textsuperscript{2} Technische Universität Bergakademie Freiberg  
Institut of Computer Science  
Germany  
steinb@informatik.tu-freiberg.de

Abstract. The development of systems comprising of hardware and software components has been a complex and demanding task. To handle the informational and architectural complexity inherent to these systems novel development approaches supported by powerful tools are taken. In this paper we present an approach that is based on UML and the concepts of platform-based design, model-driven architecture, and co-design. We will introduce how these concepts can be used to direct the development and compilation of UML-models into readily executable applications. Further we will present a tool that implements the concept of model-driven compilation in order to automatically compile hardware software implementations from UML-models.

1 Introduction

It is a recent trend in the development of embedded computer systems to integrate classical microprocessors (uC) with reconfigurable logic resources, like field programmable gate arrays (FPGA). The uP executes the global control flow and those parts of the application that would require too much effort if implemented in hardware. Calculation-intensive parts of the application or critical I/O operations are performed by the FPGAs. In run-time reconfigurable architectures (RTT) the reconfigurable logic allows for the dynamic adaption of the implemented functionality to the executed algorithm, by reconfiguring its logic resources. Often these components are integrated on the same chip, together with sensors, analog circuits or even micromechanical components to systems-on-chip (SoC), in order to improve performance, power dissipation, size and cost of the system.

To better cope with the steadily growing informational and architectural complexity of such systems new approaches to their development are taken. A common trend in this area is toward the application of object-oriented (OO)
methodologies, languages, and tools for the development of embedded computer systems. However, considering OO in the context of embedded systems is a two-edged sword. From the design point of view, the application of OO design principles may enlarge the class of problems being tractable in the reality of systems engineering. But that does not come without cost. Typical OO specifications are hard to analyze and to implement in hardware, due to their dynamic features, such as dynamic message dispatch and the utilization of dynamic data-structures. Recent advances in design theory, architectural and system-level synthesis, and reconfigurable logic offer novel options to address these problems efficiently.

Our recent research focuses on the development of RR applications with the Unified Modeling Language (UML) [5]. Of course, the academic and industrial efforts to develop embedded systems with UML are not new. However, the most of these approaches have a strong focus on software. Common problems like, hardware/software partitioning, estimation, hardware/software synthesis, and verification are not sufficiently addressed [9]. In [7] Rosenstiel et al. propose a framework for the object-oriented specification and partitioning. This work was recently extended to support the design entry with UML. From the classes in the UML model they generate skeletons of classes and methods, which are completed manually and are then refined using traditional methods. This break in the tool-chains and paradigms is still very archetypal also to most software tool-chains for UML. Other approaches use UML for the specification of systems with SystemC [2], [10]. As with the forthcoming UML 2.0 academic and industrial groups have started to use this language also for the complete design and specification of hardware/software systems.

In this paper we present an approach that allows for the complete, precise and object-oriented specification of hardware/software systems with UML 2.0 and its automated compilation into a directly executable application comprising of hardware and software modules. Our approach supports model validation, hardware/software partitioning, and synthesis. In contrast to the related works, the whole development from specification to synthesis is performed directly on and driven by UML-models. The approach supports the hardware/software co-design of applications for RR architectures. The design of the application is captured as system-level UML-model. To increase reuse and re-targeting, also the employed design-, implementation-, and hardware-platforms are specified with UML. A UML model compiler, Mocca (MOdel Compiler for re-Confgurable Architectures), implements our methodology in that it automatically performs validation, platform mapping, and application synthesis. It is important to note that the compilation process is entirely driven by the models.

The rest of this paper is organized in three sections. In section 2 we give a brief overview of our methodology and introduce the main development activities and artifacts. Because platforms and models are key concepts of our approach, we discuss their meaning and relationships in in section 3 thoroughly. Finally, in section 4 this paper is concluded.
2 Development Methodology

Our development approach incorporates the specify-explore-refine paradigm of hardware-software co-design into the concept of Model Driven Architecture [3], [6]. For a detailed discussion of the development methodology we refer the interested reader to [1] and [8].

First, during system specification the system under development is specified by means of a platform independent model. This model comprises a use-case model and a design model and may contain other models as well. The term platform independent refers to the fact that the design model does not relate to an implementation platform. However, in our approach this model is based on a design platform which is explicitly specified in a design platform model. In our approach the application as well as the employed platforms used for design, implementation, and deployment are described with with UML and the MOCCA action language (MAL) [1].

The relationship between the relevant elements of the design model and the implementation platform is established in the platform mapping step. Result of this step is an implementation model and a deployment model of the user design. We explore and estimate the feasible implementation alternatives of the design on a given implementation platform. This platform comprises the types, type mappings, constraints and tools that may be used for the realization of user designs. Again, this platform is defined in an explicit implementation platform model. Also the available hardware is specified in an explicit deployment platform model.

Throughout the final synthesis step we manifest the identified platform mapping by automatically generating the hardware and software modules and interfaces. The result of this step is a platform dependent implementation of the user design which is finally compiled and synthesized into a ready-to-run application.

3 Platforms and Models

3.1 Overview

Figure 1 shows the important models and the platforms they describe and their relationships. The models below the horizontal line define the space of possible application architectures and characteristics. Above the line the models defining a concrete application, and hence representing a point in the application space are depicted.

---

3 The term platform is quite overloaded with meanings and definitions today. In the course of this paper we refer to the definition given in [4]: "A platform [...] is a layer of abstraction that hides the unnecessary details of the underlying implementation and yet carries enough information about the layers below to prevent design iterations."
3.2 Design Model and Design Platform Model

The design-model defines an executable and platform independent realization of the use-cases. It defines the structure and behavior of the system. System structure is defined in terms of UML classes and interfaces and their relationships, like dependency, generalization, and implementation. For concurrency specification we support active classes. System behavior is defined with operations, state-machines, and the MOCCA action language. Synchronization between concurrent control flows is supported with guarded operations. The model of computation is objects communicating through structured messages.

Each design is based on a design platform. A design platform is the set of types and constraints that may be used to define user designs. A design platform reflects the basic assumptions we make when constructing systems and makes them explicit in a separate model. Such design platforms can also be found when defining systems with programming languages, for instance. However these platform are usually implicit and captured in the definition of the language and utilized libraries. Making this platform explicit is a requirement for a platform independent, retargetable approach to compilation. Design platforms can and should be shared by different designs. The concrete content and representation of a design platform depends on the targeted application domain.

Design platforms are described in various ways, for instance in language reference manuals and library descriptions. In our approach we describe the design platform as UML-model that defines the available abstract data types. For each type we define its relationship to other types in terms of generalizations and dependencies, the supported operations, and constraints, like domains and distance vectors to other types. Please note that the definition of the basic types is in fact a must when developing systems with UML, because the few types used in the UML-spec are mainly intended for the specification of UML itself and are defined too loosely in order to be useful in real-world designs.

Example 1. Figure 2 shows a small portion of an example design platform model. It shows a number of types and their relationships. For the boolean type the operations defined on its instances are shown. These are the common logical operations and the allowed type casts. For the int the figure illustrates the constraints on this type, e.g. its domain and the distance to other types. Remarkable
is also the classifier-type that is the base type of all user classifiers in the design models based on this platform. A design platform may contain more types, for instance for basic input/output, access to actors/sensor, and system control. Also these types can be used like native types in user designs.

![Diagram](image)

**Fig. 2.** Example part of a Design Platform Model

The semantic of the design types and their operations is insulated from the semantic of the action language. Therefore the action language is not fixed to this approach but may be changed specifically to the application domain. The specification of the operations is inspired by Smalltalk. The MOCCA compiler uses the design platform model for various task during the compilation process. For instance during validation of user designs, for instance for type compatibility checks and automated insertion of type casts. This compiler does only make minimum assumptions about design types, what design models are considered valid it is entirely controlled by the design platform model and the UML well-formedness rules, of course.

### 3.3 Implementation Model and Implementation Platform Model

The implementation model defines how the design model is realized in terms of implementation classes, components, and artifacts. This model is created from the design model during the platform mapping phase, either manually, by the user, or (semi-) automatically by MOCCA. As the design model, the implementation model is based on an implementation platform, which is defined in a separate model. We define an implementation platform to be the set of types, type mappings, constraints and tools that may be used for the realization of user designs. Implementation platforms define all necessary information for high-level platform mapping, estimation, and synthesis. It is subject of this model to provide
efficient implementations for the design types of the design platform and model their integration in the compilation environment, in order to allow for automatic generation of directly synthesizable and executable hardware and software modules.

In general we define a implementation platform on the basis of abstract instruction sets and execution characteristics. The term instruction may be the native instruction of microprocessors, the operations directly implementable in hardware, but also high-level operations of programming languages. For each processing element in the hardware platform an implementation platform is modeled, however an implementation platform model can be shared by processing elements. To illustrate the content of implementation platform models, we introduce it with two examples for a C/C++ platform.

Example 2. Figure 3 shows a part of an implementation platform model. The example depicts some of the design types from figure 2 and the implementation types being used for the realization of the design types. Because we can allow for stacked and nested implementation platforms, we can also find such type mappings between different implementation types. During automated platform mapping MOCCA creates one or more implementation types for the types of the user design not belonging to the design platform and establishes realization-dependencies between them.

Fig. 3. Example Implementation Platform Model - Type Mappings

The implementation types and the operations they implement are precharacterized by their quality-of-service (QoS). The QoS-characteristic of the type itself is an area value that defines the memory footprint of the instances of this type, or the number of gates in case the implementation platform is deployed on
reconfigurable hardware. The QoS of operations implemented by the types, may define the latency, power dissipation, or even abstract cost. Also the types and their operations specify information to control the utilized generator, like name-mappings and implementation language patterns for the utilization of primitive operators etc. It is important to emphasize that the straightforward definition of the type-mappings was made possible by the explicit formulation of the design platform.

The QoS-characteristics as well as the control information for the hardware/software generation is represented with the common UML extension mechanisms. The format of this information depends on the concrete requirements of the generator, mappers, and estimators of the implementation platform. In order to allow for a flexible and straightforward re-targeting the according MOCCA components and their relationships are also defined in the implementation platform model.

Example 3. Figure 4 continues the specification of the implementation platform model in figure 3. In this part of the model the MOCCA-compiler components for the C/C++ implementation platform are specified.

![Diagram of Compiler Components](image)

Fig. 4. Example Implementation Platform Model - Compiler Components

It is important to note, that the component specification is used by MOCCA in order to adapt itself to the concrete target platform. During the compilation MOCCA dynamically links modeled implementation platform specific components into its extensible architecture. Users may implement new compiler (back-end) components on their own, or specialize existing components, in order to adapt the compiler to their concrete requirements. To link the compiler into existing tool-chains an implementation platform model may also specify interpreter-components. These can be used for instance to invoke programming language compilers, logic synthesis tools etc. transparently during the compilation process.
3.4 Deployment Model and Hardware Platform Model

The implementation model of a system defines the realization of the design model, and the same applies for the implementation platform model. The actual deployment of the implementation model components on the nodes of a target hardware architecture is defined by the deployment model. In this model deployment relationships between (processing) nodes and the artifacts manifesting the system (e.g. executables, hardware configurations, libraries, tables etc.) are defined. Again, this model may be defined (semi-) manually or entirely automated by MOCCA.

The deployment model is defined in terms of an underlying hardware platform, that is defined in a hardware platform model. We define the hardware platform model as the set of abstractions of hardware nodes, their communication channels and constraints that may be used for deployment and execution of design (platform) model implementations.

In contrast to other approaches we do not specify the micro-architecture of the hardware itself. Instead we characterize the hardware components and the resource services they provide sufficiently well in order to enable a fast design space exploration based on sufficient estimates of the implementation characteristic of the explored implementations.

4 Conclusions

In this paper we presented a novel approach to the development and compilation of Uml-models for reconfigurable hardware architectures. The presented approach is based on platforms and their Uml-models. The platforms are used throughout the each phase of development, from design to the final implementation and deployment. We have shown that platform models serve two main purposes. First, carefully developed platform models make the basic assumptions and restrictions of a dedicated platform explicit to the developer. This may improve the quality, comprehensibility, and portability of system developed with this approach. Second, platform models are a very useful tool to direct the automated transformation of system specifications into final products. Dedicated model compilers use platform models to perform automated validation, platform mapping, synthesis, and optimizations on user models in order to create directly executable applications from system-level specifications.

As an example for such a model-compiler we have presented MOCCA. This compiler implements the presented approach. Currently we are validating and evaluating our development approach on several real-world designs. This includes the development of performance critical applications of embedded systems, like encoders/decoders for multimedia-streams. The first results of the evaluation are very promising. Especially the good support for the system-level exploration of design alternatives at the Uml-level and the immediate translatability into an executable implementation with hardware and software is very appealing to users.
References


