OBJECT-ORIENTED CO-DESIGN FOR RUN-TIME RECONFIGURABLE ARCHITECTURES WITH UML

T. Beierlein \(^1\), D. Fröhlich \(^2\), B. Steinbach \(^2\)

\(^1\) Germany, Mittweida, University of Applied Science, Labor Embedded Control
\(^2\) Germany, Freiberg, Freiberg University of Mining and Technology, Institute of Computer Science

In this paper we present an object-oriented approach and a development environment for the system-level design of run-time reconfigurable computer systems. We use the Unified Modeling Language for the specification, modeling, and visualization throughout all phases of development. We show how object-oriented applications can be implemented and executed on custom computing machines comprising run-time reconfigurable logic resources and how the problem of hardware platform mapping can be solved in such an environment.

1. Introduction

With the increasing efficiency and availability of run-time reconfigurable (RTR) computer architectures and devices, RTR systems become important candidates for the acceleration of a broad range of problems. However, years after its inception, reconfigurable computing still faces a wide gap between the capabilities offered by the hardware and the availability of mature methodologies and tools for the development of their targeted application domains. For reconfigurable computing to move out of the laboratories into general purpose computing, time and cost for the design, implementation, and deployment of its applications must become comparable to software development.

HW/SW co-design for object-oriented (OO) systems became a topic of interest to the research community during the last few years. 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. Recent advances in design theory, architectural and system-level synthesis, and reconfigurable logic offer novel options to address these problems efficiently.
We developed a process and a supporting tool for the object-oriented system-level specification, design and implementation of applications for RTR-architectures. In our approach both, the application under development and the reconfigurable target platform are specified with models. The models are described with the Unified Modelling Language (UML) and an action language. A dedicated UML model compiler, MOCCA (MOdel Compiler for reConfigurable Architectures), implements our methodology in that it automatically performs system validation, platform mapping and application synthesis.

The research on development methodologies for reconfigurable computer systems is still quite new. In [2] Chata and Vemuri presented the SPARCS design environment and a design flow. However, they do not address system-level specification, but high-level behavior specifications. Eisenring and Platzner [3] are developing an implementation framework for RTR systems, which consists of a development methodology and a design representation. The system specification is done at the task-level. The framework allows for partitioning, estimation, and synthesis.

The rest of this paper is structured as follows: In section 2 we will start with a brief introduction of our UML-based co-design approach. Due to its particular importance to any HW/SW co-design approach we discuss hardware platform mapping more thoroughly in section 3. Finally this paper is concluded. For examples, and a detailed description of our development environment we refer the interested reader to [1].

2. UML-Based Co-Design Approach

2.1. Motivation

Today it is common practice to use software programming languages, like Java, C, or C++, or variants of programming languages to capture system-level specifications, e.g. SystemC [8]. Executable co-specifications are commonly used as basis for system validation and synthesis. However, as stated in [1], these languages have a strong focus on implementation. This renders them actually insufficient for the purpose of system-level specification.

Due to these problems there has been a growing interest in using UML as co-design. In UML systems are specified by means of models rather than code. This may improve design quality and tractability of complex systems. The expressive power and generality of UML allows for the specification of the important different system aspects in a consistent and comprehensive manner.
to the user. Also, the extensibility of UML makes the language adoptable to different application domains.

As with UML 2.0 the formerly crude capabilities for the detailed modelling of behavior improved significantly. The UML action semantics enable the complete and precise behavior specification. Action languages provide a syntax for this semantic which is suitable to the targeted application domains.

### 2.2. Activities and Artifacts

Figure 1 shows the basic activities and artifacts of our development approach. The illustrated approach incorporates the general methodology of hardware-software co-design into the concept of Model Driven Architecture (MDA) [4][7]. It is a particular goal of MDA to separate the application from the platform, comprising of hardware, operating systems, compilers etc., used for its implementation. This not only supports the re-targeting of the application but can also simplify the mapping of a given application to a particular platform.

Fig. 1. UML-Based Co-Design - Activities and Artifacts

In our approach the system and the target platform are defined by means of independent models, described with UML and a dedicated action language. The system is specified by means of a platform independent model (PIM) of its structure, function, and behavior. The target platform model (TPM) defines the services provided by the implementation target for system execution.

Given a PIM of the application we proceed by transforming it to a platform specific model (PSM). The PSM is defined by mapping the PIM to the TPM.
Initially, we fix the hardware specific parameters of the implementation by mapping the objects of the application to the target architecture. Finally, if the implementation requires additional operating system support, e.g. for inter-object synchronization and communication, the according elements of the PIM are bound to the resources provided by the respective target platform devices.

The final step of the development process is synthesis. Given a platform specific model of our application we proceed by transforming it into an implementation model which is finally synthesized into a ready-to-run implementation. The synthesis of software and hardware modules is parameterized by the implementation language specific patterns and rules which are incorporated into the synthesis tools and the target platform model.

The supported UML sub-set for modelling the system function includes use-cases and collaborations, although these elements are not considered for synthesis, as you would expect. For structural modelling we support the relevant UML classifiers, like classes, interfaces, primitive data-types, artifacts, components, and nodes, and their various relationships. For detailed behavior modelling we support UML state-machines, activities, and actions. The MOCCA action language (MAL) provides a syntax for the UML actions. This language allows for the detailed specification of control flow, arithmetical/logical/relational operations, instance and link manipulation, and time calculation. The model of computation is object communicating through structured messages.

3. Hardware Platform Mapping

3.1. Activities and Artifacts

Platform mapping is the transformation of a PIM of the application into a PSM by relating each element of the PIM to the resources of the target platform that will be used for its implementation. Platform mapping involves hardware platform mapping and software platform mapping. Due to space limitations and its particular importance we will concentrate on the discussion of the former.

Hardware platform mapping starts with the PIM or a partial version of the PSM. The transformation of the PIM to the PSM is done by iteratively changing the model structure and behavior such that it can be implemented on the given target architecture. The design space is explored by the iterative application of feasible transformations to the model. Each time the computation of a complete mapping is finished, we estimate its quality with a cost function. Finally the best candidate is chosen for implementation. The
cost function is based on the general resource model (GRM) as specified in [5].

3.2. Target Hardware Architecture

The class of hardware architectures targeted by our approach are heterogeneous custom computing machines consist of a microprocessor (master) that is accompanied by reconfigurable logic resources (slaves). The master executes the overall control flow and the functionality with a small gain/cost-ratio with respect to its hardware implementation. Functionality with high gain/cost is implemented with the slaves.

The functionality of each hardware device is modelled in the TPM by means of processing resource services which comprise an abstract operation set, the available operation implementations and their QoS (quality of service) characteristics, like latency, area, power consumption, and usage constraints [5]. Moreover, for each operation, implementation language specific rules and patterns are given [1]. Other resource services, like communication, storage, or logic, are modelled similarly. Architectural differences and communication facilities are captured by means of abstract communication operations.

In our system-level approach we build upon 3rd generation programming languages and architectural synthesis tools to implement the software and hardware modules of the application respectively. For example, for hardware modules, MOCCA currently supports behavioral VHDL and Java for the Xilinx Forge compiler [9]. On the software side we support Java and C++.

3.3. Mapping of Objects to Reconfigurable Logic

We based partitioning on objects, mainly for two reasons. First, objects are a fundamental source of concurrency. Each object can be seen as an abstract machine providing services that may be requested by other objects by means of messages. All objects of a system execute concurrently; synchronization is reached through the messages. This object-based model of computation maps well to CCM-architectures comprising of multiple, possibly reconfigurable, processing elements. Second, objects encapsulate data and behavior to process the data. During partitioning of objects, both function and data are partitioned, which can reduce communication effort.

Objects are bound to logic resources and instantiated at run-time. Binding is performed on the base of complete configuration contexts per device, whereat each configuration context may contain multiple objects. The management of the logic resources and objects is performed dynamically by a
light weight run-time environment service. The Binding of objects to logic resources is done on demand, driven by the overall control flow of the system.

There are many design options to interface objects, like shared memory, memory or I/O mapped registers, or even full-fledged communication networks, all of which may be found in the same architecture concurrently. In our model we abstract from these differences by means of abstract communication operators. The interfaces are automatically generated by device-type and implementation-language specific back-ends\(^1\).

### 3.4. Mapping Algorithms

Given the described implementation and execution framework one can think of many different mapping algorithms. The feasibility of an algorithm depends on its specific tradeoff between the quality of the computed solution and the effort to compute this solution, which not only depends on the mapping algorithm itself but also on its input - a PIM or a partial solution of a PSM, and the TPM. We do not describe a single algorithm here, but outline the algorithmic framework in which we implement such algorithms.

The figure illustrates the algorithmic framework for mapping algorithms. First, the system is profiled by estimating the execution probability and frequency of each realized behavior, by computing a execution profile of the system. Second, we compute candidate mappings for each realized model element. The number and type of mappings pre-computed for a specific model element is a function of the expected utilization of this model element. For instance, for model elements the number of allocation candidates pre-computed for a model element increases with the execution probability and frequency of this element. Thereby, we account for different implementation options of each element. Moreover, this allows us to adjust the trade-off between compilation time and the quality of

---

\(^1\)MOCCA currently supports the fully automated generation of memory mapped register interfaces. The external interface of each object contains registers for the exchange of message parameters, synchronization, and a type-ID register by which the dynamic message dispatch is controlled.
the computed mapping. The search space is reduced by hierarchically nesting of interdependent mappings, whereby the selection of a mapping enables the selection of mappings which depend on it.

In the third step, the mapping candidates are preestimated. The local QoS of each candidate mapping is estimated by means of the QoS-characteristics of the used resource services as modelled in the TPM.

Next, the actual mapping is performed. We start by computing a complete PSM by selecting and applying a sufficient set of candidate mappings to the PIM or a partial PSM. This initial solution is then progressively refined such that the quality of the overall partitioning is optimized. For this we iteratively select sets of mappings from the set of available mapping candidates, estimate the effect that their actual application to the model would have, decide if the mappings shall be applied, and if so apply the mappings. The strategy for the selection and application of mapping sets is subject to the concrete mapping algorithm. For instance a simulated annealing algorithm will typically select candidate mappings for only a small number of cohesive model elements and apply those mappings which improve the current solution. A branch-and-bound algorithm may apply the mappings in a deterministic order and apply only the best. Online-estimation is based on the preestimated QoS of the mapping candidates. The overall number of iterations is fixed, bound by the time, or a predetermined QoS that must be reached.

5. Conclusions

In this paper we presented an approach and a supporting environment for the object-oriented system-level development of applications for run-time reconfigurable computer systems. To overcome some of the drawbacks of approaches we use the universal syntactic and semantic framework provided by the Unified Modeling Language. This approach enables a seamless and comprehensible transition of the system specification to a ready-to-run application, without the need to change the applied paradigms, methods, or language in between.

In the second part of this paper we presented an approach how hardware platform mapping can be accomplished in co-design approaches based on object-oriented specifications in UML. We have proposed an implementation and execution framework for objects realized in reconfigurable hardware. The presented approach has been implemented in the MOCCA-compiler. Currently we are working on the experimental evaluation of the quality of the designs generated by this tool.
References


