Software/Hardware Co-Design in the MuSE Environment

Matthias Deegener
Department of Computer Science
Technical University of Darmstadt
Alexanderstr. 10, 64283 Darmstadt, Germany

Sorin A. Huss
Department of Computer Science
Technical University of Darmstadt
Alexanderstr. 10, 64283 Darmstadt, Germany

Abstract

The research project MuSE \(^1\) provides an integrated software environment for the development of complex technical systems. MuSE supports the system engineer by its tools for the specification, simulation, and validation of the system’s behavior. MuSE also provides the structured archiving of all documents in a hypermedia database. This paper will focus on the specification and validation of the system’s behavior, especially on the concepts and tools for Software/Hardware Co-Design and Co-Simulation in the MuSE environment.

1 Introduction

The increasing complexity of technical systems that have to be developed in decreasing time periods requires a methodical approach to system development. Besides validation of the system, the documentation is very important. Due to increasing sizes of the projects, special methods for a multi-user development have to be considered.

In this paper we present an approach to system development, which consists of a hypermedia database system for development and documentation, different specification languages and a virtual reality user interface for system validation. Figure 1 depicts an overview of this architecture.

The goal of the MuSE project at TU Darmstadt is research in modelling and validation methods of complex technical systems. In order to separate modelling and validation processes there are two different domains in MuSE in which different user groups can

\(^1\)MuSE: “Multi-media technology for System Engineering” is sponsored by the “Deutsche Forschungsgemeinschaft” (German research foundation) under grant number He 1170/5-1,2

Figure 1: Architecture of the MuSE environment

work. Models are developed and tested in the modelling domain. Additional data, such as special requirements, milestone plans and simulations runs can be linked to the models. Of special importance is the multi-medial description of the design object that allows its visualization as well as an appropriate interaction with it.

The validation domain provides a documentation and simulation environment for users, who perform inspection tasks according to technical regulations. Changes in the model specification or requirements are not allowed. Classified data can be hidden to certain user groups, too. However, it is possible that the person performing validation generates simulation or examination protocols.

System simulation is an important procedural step for engineering tasks in both domains. Compared to the traditional development of single components it offers essential advantages. Thus, the development of a component of the system is no longer an individual process, but interacts with the development of the complete system. The design engineers will find their components in system context right from the start and they can detect possible restrictions easily. The probability of errors or incompatibilities is substantially lower following this methodology. In addition, the examination of the system in view of acceptance tests
may already take place during the design process.

Different approaches to the integration of system components to be implemented by means of software and hardware functions have meanwhile been introduced, but simulation is usually restricted to descriptions in each domain. The system environment that generates the actual stimuli for simulation is in general not considered. The use of e.g. SDL offers the possibility to map specifications into C or into other languages as well as the translation into VHDL[1]. However, the mixed software/hardware descriptions thus generated are separate partitions that cannot be composed jointly. If specified, the modelling of the environment is limited to the direct interface of a subsystem and is specified on the same abstraction level. In case of an application specific circuit, the interface consists usually of the components connected directly to this circuit.

In the proposed approach the complete technical system is consistently modelled and jointly simulated on several abstraction levels as well as in various specification languages.

2 Modelling of technical systems

Models generated in the MuSE environment reflect the architecture of real components. A system is viewed as a set of components working concurrently. These components are mapped to processes describing the behavior of the components. Interaction between components is modeled by process communication.

For the specification of the system behavior and its application environment a total of three dedicated description languages are supported in MuSE. The first language is State Event Logic[2], a concept for the description of parallel events that is based on first order predicate calculus. The second one is Sampλe [3], a functional language that was developed at TUDarmstadt. The standardized hardware description language VHDL is the third language used.

The first step of defining the system behavior consists of generating a reference specification in State Event Logic. The form of this specification allows an easy transformation to Prolog [4]. The use of Prolog supports the formulation of a large number of different inquiries of the system behavior. On the one hand, we can check on how the system reacts on certain input commands. On the other hand, we might ask questions about the system behavior in tricky situations. An assessment of the reference specification is thus achievable.

For detailed simulation and a first step towards an implementation we transform parts of the specifications (processes) to the functional language Sampλe. This specification is efficient enough to be used as the base for our virtual reality (VR) simulation, because real time simulation requirements have to be met.

An attempt is made in MuSE to provide a universal concept for the parallel description on all abstraction levels. The extension of Sampλe by a process concept [5] makes it possible that, just as in VHDL, design objects can be composed of parallel virtual processes. According to the 'top-down' design methodology the components are initially described in an abstract manner. With this functionality a prototype can partially be tested. In order to be able to consider timing restrictions, we exploit VHDL on the same abstraction level.

2.1 Partitioning into HW and SW components

By means of the process concept outlined above, which is available both in Sampλe and VHDL, we then use the thus partitioned specification for the generation of virtual processes in VHDL.

A consideration of more details requires further partitioning. Especially if the goal is implementation, the behavioral description has eventually to be mapped to a structural description. For this purpose synthesis tools can be used. This requires additional information, since the so called "high-level" synthesis is not generally applicable yet at this moment.

In a target architecture that consists of microprocessors or DSPs and of one or more ASICs, the software to be executed on the processors plays an important role in partitioning. The software can be divided into several independent processes. Mainly due to timing restrictions parts of the software have to be replaced by an ASIC in order to cope with performance specs. In the approach introduced in the following the test of partition variants is supported. By means of the flexibility in composing a system, implementations can be optimized according to cost and timing constraints. In addition, an even more flexible mapping of functions to software and hardware can be achieved by means of an inclusion of techniques that produce special microprocessors and a dedicated design environment [6].

Based on the CSP concept of C.A.R. Hoare [7], we define a class of events, that are based on communication events and denote them timed communication events. They are represented by a tuple c.v.t, where c is the name of the communication channel, v is the value of the message forwarded and t is the time, when
the value is valid. This triple can also be interpreted as a pair \((c, s)\) of a pair \((v, t)\), where on the communication channel \(c\) a signal \(s\) is transferred. This signal \(s\) consists of the value \(v\) and the time \(t\).

We define functions which extract channel, message and time values:

\[
\text{channel}(c, v, t) = c, \quad \text{message}(c, v, t) = v, \quad \text{time}(c, v, t) = t.
\]

and

\[
\text{channel}(c, s) = c, \quad \text{signal}(c, s) = s, \quad \text{message}(v, t) = v, \quad \text{time}(v, t) = t.
\]

This definition provides an interface for interprocess communication. The communication structure is depicted in Figure 2.

![Diagram of communicating processes](image)

**Figure 2: Communicating processes**

As an example consider a simple process DNOT, which inverts and delays the input value.

\[
\text{DNOT} = \mu x. \ (\text{in}=x \rightarrow (\text{if message}(x)=0 \text{ then } \text{out}=1 \text{ time}(x)+\Delta t) \text{ else out}=0 \text{ (time}(x)+\Delta t) \rightarrow X),
\]

where \(\alpha \text{in}(\text{DNOT})=\alpha \text{out}(\text{DNOT})=\{0.1R, 1.1R\} \).

In this example we use a transport delay model, where each value at the input will appear at the output after \(\Delta t\) time units.

Actually we work on a partitioning tool for the implementation of this process concept. This tool is a sort of backend for the MuSE environment. Based on the process concept, processes can be generated, refined and combined. Hierarchical design is a main objective of this tool. We have defined an appropriate data structure that describes the communication ports as well as the timing behavior and other restrictions of the processes. In Figure 2 a snapshot of the tool is presented. For each process a certain delay time is given, which may be viewed as its operation time. Additionally one could define detailed timing information, e.g. pin to pin timing, or restrictions in memory usage or area consumption. All of these data can be used to check whether a generated implementation meets the specification.

The internal behavior of the processes intermediately is specified using Sample. Sample as a functional language offers some facilities, which are useful for prototyping and testing. Functions of higher order are such an feature. Using functions as arguments for other functions is a feature that is useful for prototyping, but has no representation in VHDL. Ignoring these features for an easy transformation from Sample to VHDL will deprive the functional language and hence no advantages would be left. For that reason we allow such constructs, but also we have defined an Sample subset, which allows an automated transformation to VHDL or even C.

After testing the prototype with different higher order functions, only a few relevant ones will remain to be implemented. The designer can manually translate these higher order functions in conditional statements. The main advantage is, that the designer can stay in "his" language for prototyping and for implementation. This is similar to synthesizing from VHDL, where different abstraction levels can be described, but only a subset of the language syntax may be synthesized.

The partitioning tool is aimed to different applications. The first one is generation of Sample process code for rapid prototyping.

The second application is generation of VHDL code. C is supported, because there are, for almost every standard processor, good compilers available. With the GNU-C-compiler environment, even a dedicated compiler for application specific microprocessors may be produced [6].

The reuse of designs is the fourth application. By means of this hierarchical design approach, parts of the design tree can easily be reused in future projects. For the reuse of designs, which where implemented without our partitioning tool, we will provide an input filter for VHDL and Sample descriptions.

The last application of this tool is the generation of a dedicated simulation backplane for the system to be designed.

Once the processes are defined/loaded into the partitioning tool the hierarchical design can be manipulated by the user. The ability to move processes between different hierarchical levels and also generate new process instances and hierarchical levels is
main feature of the tool. The communication channels are automatically adapted. With the reconfiguration of processes the timing behavior is updated and the tool is therefore able to indicate timing constraint violations.

Figure 3: Partitioning Tool

The processes are graphically represented by circles, and communication ports by rectangles. The communication channels are shown as directed links as depicted in Figure 3.

With increasing complexity of the system, the number of links on certain hierarchical levels between two processes might be too large for a meaningful graphical representation. Therefore we use representation links, which are displayed as one link, but act as a link bundle. The contents of these links can easily be inspected.

With a sequential description in Assembler, derived from a C program and additional timing assertions or a VHDL model, the timing behavior for program execution for a specific standard microprocessor described in the proposed data structure can be evaluated.

2.2 Design environment

In order to simplify the validation of large systems, where several designers and validators are involved, a hypermedia system is exploited in $\textsc{MuSE}$. This system is based on the co-operative author’s system SEPIA [8]. There is a distinction between two groups of users: modellers and validators. In Figure 4 one can see several levels of this hypermedia environment. In the upper left part of Figure 4 there is a selection window for the user group. By selecting “Modellierung” (modelling) or “Validierung” (validation) one gets different views of the same hyper-network. On the right here is a partitioning variant shown. The various basic components of the example described in Section 4 are further subdivided into subcomponents. The components “Fahrverhalten” (driving behavior) and “Fahrzeuggebung” (driving environment) are displayed by groups of subcomponents, whereas the partitioning of the depicted object is visualized by means of links.

3 Validation by co-simulation

Validation of system models is achieved by the simulation of the complete model. In the past, special simulators were used for the individual parts. Even algorithms (software) were tested on a workstation only and not, in general, on the emulated processors. In addition to the separated simulation of algorithms and hardware, prototype related simulators are used in industry. In such dedicated simulators the software is tested on a special prototype. This is a processor board with standard processors or digital signal processors (DSP), frequently denoted as 'single board computer'.

A clear disadvantage of the prototype-based simulators is the fact that partitioning in software/hardware already has to be completed in advance. In case that problems with timing constraints arise later, one has to start all over right from the beginning, or to accept restrictions. By linking simulation of hardware and software components one can avoid the implementation of such a prototype. By means of the proposed concept it is possible to change the partitioning at any time. Several cooperating simulators are used for validation in accordance with the $\textsc{MuSE}$ approach. The interactive graphics input and output act as the
validation interface. The graphical interface is developed according to the model requirements, but it is independent of its functional behavior. The interface is implemented by means of a visualization and interaction toolkit in C++. The processes describing the functionality of the system have defined interfaces, part of which are used for the visualization and interaction. By defining the multi-media and functional process interfaces, the exchange of parts is very simple. Thereby the processes are assigned to single simulators and can thus be simulated in a cooperative manner. No additional interfaces to interactive visualization are provided at this level. If the designer or validator wishes more detailed simulation results, he has to leave system simulation and use special simulators that are usually available. But the simulation can be supported by the database that stores the simulation runs. Thus, a component can be examined with the debugger available for the description language used, whereas the stimuli can result from the preceding interactive system simulation.

3.1 Conversion of data structures

Most simulators usually do not support co-simulation. The commercial VHDL simulator, which we use, is conceived as an independent simulator and offers no interface to other simulators. Since interchangeability of the components plays an important role in the MuSE project, interface definitions and interface access are defined and implemented by the 'remote procedure call' method. This results in a C or C++ interface for the VHDL simulator, which takes over the data transfer from and to the simulator and controls the simulation run. A suitable control is essential for commercial simulators, since they are used in 'black-box' mode.

In addition, data and data types on both sides need a definite representation. Therefore, VHDL data types are converted to C++ classes, which results in two advantages in comparison to implementation through function calls in C. First, VHDL simulation can be started automatically by allocating values to class-defined variables. In doing so, the system partition modelled with VHDL can be treated like a C++ object. Second, C++ is an object-oriented language, resulting in the fact that an object oriented design for the complete system can be built using object oriented techniques on top of VHDL [9].

An object oriented design can also serve as a basis for the construction according to a 'bottom up' design process. The conceptual advantages of object oriented languages are especially suited to support the reuse of objects.

One problem that arises when exercising interfaces of the components during VHDL simulation is that VHDL data types are not available in C++ directly. Therefore, individual C++ classes are defined for the 3 main groups of data types of VHDL. More detailed information about linking C++ descriptions to modules coded in VHDL can be found in [10].

3.2 Linking simulation tasks

There are two different concepts how to run simulation, namely the time driven and the event driven approach. By means of parallel processing of components during simulation the operation concepts of the individual simulators have to be coordinated in order to achieve co-simulation. The CSP model exploited for partitioning purposes serves again as a basis. Using the outlined communication concept the event driven simulation is appropriate. The VHDL simulator used works according to this principle.

The implementation of a system with parallel event simulation requires special synchronization mechanisms. A simpler method of synchronization, that is also used in our example, is the time driven equitemporal simulation. The processes are no longer initialized through changes in any of the input signals but through a central clock only. Thereby all modules are synchronized.

In the VHDL simulator event simulation remains unchanged, whereas synchronization takes place through conditions of the kind 'will complete in specified time'.

There are many possibilities to link a HW simulator with a C program. Therefore, many VHDL simulators offer a C interface. However, this interface has a limited functionality and is aimed for the use of component models coded in C. In our implementation the principle applied there is reversed. That means, that a VHDL simulator, which possibly calls C procedures, is not the simulation manager. It is just one part of a heterogeneous simulation environment. That is why it is important to be able to control the VHDL simulator and to synchronize it to other parallel simulation processes of the environment.

In order to exploit commercially available simulators, we therefore work exclusively via the built-in I/O interfaces of these tools. Therefore, a parallel operation of several VHDL simulation tasks is a major advantage of our approach. Consequently, a design which consists of standard processors and ASICs might be tested at first without any regard to time behavior by means of a C-software simulation on a
workstation. After completion of partitioning of the functionality into software modules for the processors and hardware functions (ASICs), the performance can be validated by means of VHDL descriptions of both ASICs and processors, which run the software modules.

4 Application example

![Diagram](image)

Figure 5: Simulation environment for the example

We present a Rear Wheel Steering (RWS) of an off-road vehicle as an example to demonstrate the proposed approach. This subsystem was selected because it combines components originating from different engineering domains: electronic components and a hydraulic subsystem. This RWS is designed as a part of an small truck (MB Unimog). Therefore it has different functions in order to allow for different degrees of the vehicle’s maneuverability. These functions are selected by a switch. It offers four positions: off, reduced turning radius, extremely reduced radius, diagonal motion. The functions are implemented using the distinct steering algorithms. Rear wheel angles between -20° and 20° are achievable and are set up by means of a hydraulic rear wheel steering. For security reasons the RWS is disabled and blocked at speeds over 50 km/h. All sensors are duplicated as to detect possible faults.

If the complete system is to be simulated, we have to build models for both, the vehicle and its environment. This means that the RWS is validated by “driving” the vehicle within the simulation environment. Handling of error conditions and disturbed input data may also be evaluated during the simulated operation of the vehicle.

The development is aimed at the RWS, but all other parts of the vehicle have to be specified as well. By means of a graphical interface this vehicle can be operated. The hydraulic part is represented by input-output relations, whereas the electronic component will be modelled in VHDL supporting its concurrent operation. Therefore concurrent partitions have to be defined. The system and its environment, which may be viewed as a test-bench to the RWS to be developed, consists of three main sections, as depicted in Figure 5:

1. Vehicle and environment model (C++)
2. Software parts (C, C++, ), which model the sensors, hydraulic parts and steering algorithms of the RWS
3. Hardware component (VHDL) of the RWS.

![Diagram](image)

Figure 6: The drivers cabin of the Unimog in virtual reality (VR)

Figure 7: Validation tool interface

The graphical interface offers all important steering functions of the vehicle. Display instruments and out-
puts of the RWS are represented as outlined in Figure 5.

The signals and data types being defined within the entity of the HW component are transformed using the methods outlined in the third section. Now we can combine the software description and the hardware model and simulate them jointly, because the VHDL model consists of the components of its description and an underlying VHDL simulator. This technique is completely different to previous approaches, e.g. [11]. Therefore, we achieve a co-simulation of interacting mixed-systems descriptions.

As described in the third section of this paper a time driven simulation is used. Here the simulator clock rate is 50 Hz. Inside the VHDL simulator an event driven simulation with shorter time intervals is executed. For most input/output values there are no problems with clock rates, but for example the speedometer has an asynchronous output: its frequency rises with increasing speed. In addition, the outputs for the hydraulic valves have a 160 Hz clock rate. At the beginning of each simulation interval the actual frequency of the functional block is transferred to or from the simulator, respectively.

Two graphical interfaces are provided for this application: the window based user interface shown in Figure 8 and the VR interface, which is depicted in Figure 6.

At first, a traditional X/Motif user interface offers all the basic functionality to control acceleration, breaks, steering, and the steering mode of the rear axis of the vehicle (Figure 8). It is possible to perform all the necessary actions to explore the different steering modes and its behavior at different speeds.
In Figure 7 the processes of the top hierarchy, the communication lines between the processes and the central clock are depicted. In order to monitor the exchanged data, a system developer or test engineer can click on such a link. A popup-window shows the data that is exchanged while the simulation is running. With the modification of this data the impact on the simulated system can be observed immediately. This is an important feature for testing of fault handling systems.

For detailed simulation of electronic parts, the VHDL description can be exercised by means of a commercially available simulator. As described before, this simulator is embedded in the environment. Internal information of the components is accessible by the simulator only. As depicted in Figure 8, design analysis tools (e.g. Synopsys Waves) can jointly be used within the MusE environment. In the background of Figure 8 one can see the hypermedia system, via which the simulation is started. The graphical user interface and also the validation tool are active, while the simulator also provides output for the waveform analyzer. Internal conditions of the component under test and internal values, that are not transferred to the validation tool, can thus be examined.

References


