

Description and Simulation of Hardware/Software Systems with Java

Tommy Kuhn
University of Tübingen
Sand 13, Germany
Phone: ++49 7071 29 7 59 40
kuhn@fzi.de

Wolfgang Rosenstiel
University of Tübingen
Sand 13, Germany
Phone: ++49 7071 29 7 54 82
rosen@fzi.de

Udo Kebschull
University of Leipzig
Augustusplatz, Germany
Phone: ++49 341 9 73 22 08
kebschull@fzi.de

1. ABSTRACT

In this paper a newly developed object model is presented which allows the description of hardware/software systems in all its parts. An adaptation of the component model JavaBeans allows to combine different kinds of reuse in one unitary language. A model based design flow and some tools are presented and applied to a JPEG example.

1.1 Keywords

Object oriented hardware modeling, simulation, codesign.

2. INTRODUCTION

Recent years have brought a considerable rise in the complexity of integrated circuits which can only be handled by descriptions on higher and higher abstraction levels. The complexity is transferred from modeling to synthesis and simulation. Thus, rapid simulation and, above all, reuse concepts become of decisive importance and should be supported by an appropriate language.

By the transition towards higher levels of abstraction, the descriptions for software and hardware become closer and closer. Therefore it is reasonable to search for a single language for the description of hardware/software systems.

This contribution presents an approach, based on the usage of the object oriented programming language Java. The paper concentrates on the definition of an appropriate object model. The model differs considerably from other approaches with regard to its descriptive power. Normally, they are limited only to behavioral descriptions on the algorithmic level. In contrast, the presented approach is domain overlapping, and enables the description of hardware and hardware/software systems at varying levels of abstraction (e.g. register transfer, algorithmic and system level). Besides this, the integration of the component model JavaBeans enables graphical modeling and visualization

from different points of view: structural as well as behavioral point of view.

3. PREVIOUS WORK

The application of object oriented techniques to the field of hardware description is of course not new. The use of hardware description languages leads straight to object oriented VHDL [4,6]. The approaches are very interesting, but we do not consider VHDL as an appropriate language for the description of hardware/software systems. Scenic [3] uses C++ for algorithmic descriptions. [1,7] uses Java as a specification language. But they also only concentrate on behavioral descriptions on the algorithmic level.

4. THE OBJECT MODEL

Fig. 1 shows a schematic format of a typical hardware/software system. Both parts, software and hardware, make

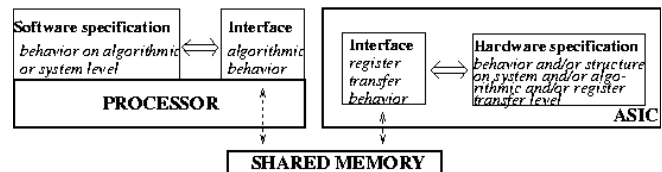


Figure 1. Structural view of a hardware/software system

up two structural components. These components are connected to one another. The software part is given in the form of a behavioral description, whose functionality is realized through execution on a processor. The hardware part is different. It is composed of structural blocks (primarily interface components) and behavioral parts. In this case the functionality of the behavior is realized by a circuit that results from the synthesis of the behavioral description, while the functionality of the interface components are already present in the circuits of a library and can be reused. So the total hardware/software system obviously is composed of structure and behavior on system level, algorithmic level, and register transfer level. The description covers all domains. The familiar approaches are not in a position to do this. In contrast to procedural language concepts object oriented languages can be adapted to cover all the domains. The problem is solved by an object model that is based on differing interpretations of objects.

4.1 Structural Interpretation

Structural interpretation interprets **objects as components** of the structural point of view. The abstraction level currently considered determines what is to be meant by a component.

Since objects communicate with one another via method calls, the **methods are interpreted as ports/connections**. Fig. 2 shows this interpretation, which should be designated as *structural interpretation* in the following.

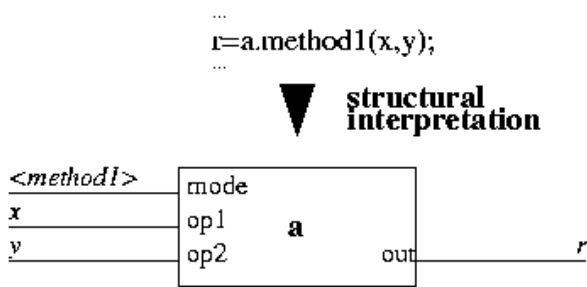


Figure 2. Example of the structural interpretation

The objects are called **hardware objects**. All parameters are primitive types which can be converted into ports of the bitvector type during synthesis. Structural interpretation allows hardware objects to be mapped on structural components. They can be utilized like regular objects when used within behavioral descriptions. Therefore, from the descriptive point of view, access to the structural components become simple method calls. Hereby access becomes transparent for the designer.

On the other hand, hardware object *a* of fig. 2 cannot be reused and inserted in just any structural description. Several reasons prevent this: first of all, structural descriptions require that it is possible for individual inputs to be entered independently of one another. Furthermore, structural components often have considerably more than a single output. A further problem arises from the standpoint of simulation. Method calls are sequential instructions. Therefore connections between hardware objects in structural descriptions through method calls are only temporarily available.

A solution to these problems is possible through an adaption of the JavaBean component model. A tool, java2struct, was implemented for this purpose. Fig. 3 shows this tool's function. A given class is expanded by so-called setters and

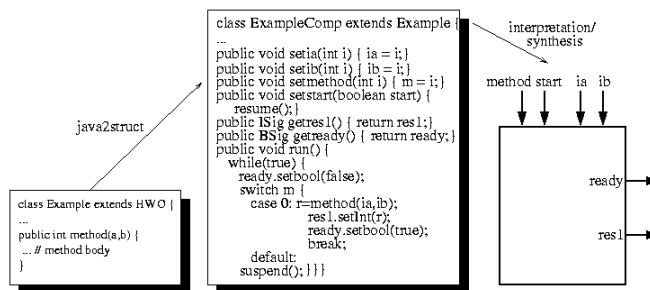


Figure 3. Automated Bean conversion with java2struct

getters through inheritance. Each input port is represented through a setter, and each output port through a getter. In behavioral descriptions the object can be addressed by the usual methods, and in structural descriptions by way of its

setters and getters. Java2struct also assures that an event is fired at each change in the return value of a getter. The output of java2struct meets the JavaBeans specification. The Beans are called **HardwareBeans**. They are set up in such a way that further processing with visual builder tools like IBM's Visual Age for Java is possible.

4.2 Behavioral Interpretation

The following interpretation called *behavioral interpretation* is completely different according to the interpretation of the preceding section. This is mandatory since structural interpretation pins down the binding of an object to an already synthesized component of a library. While this is desirable for structural descriptions, it is absolutely undesirable for behavioral descriptions. Specifically the creation of such a mapping is the job of synthesis. The behavioral interpretation no longer treats objects as components. The **objects are seen as connections** between components which are themselves represented by the methods. This interpretation is illustrated again in fig. 4.

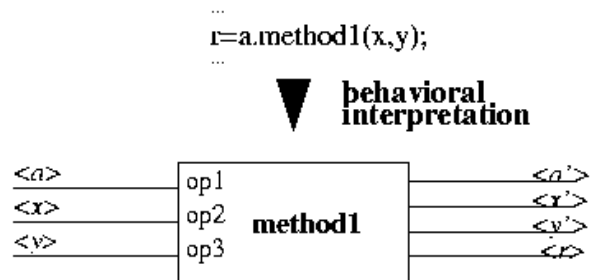


Figure 4. Example of the behavioral Interpretation

4.3 Model Based Design Flow

The total design flow, based on the introduced object model, is shown in fig. 5.

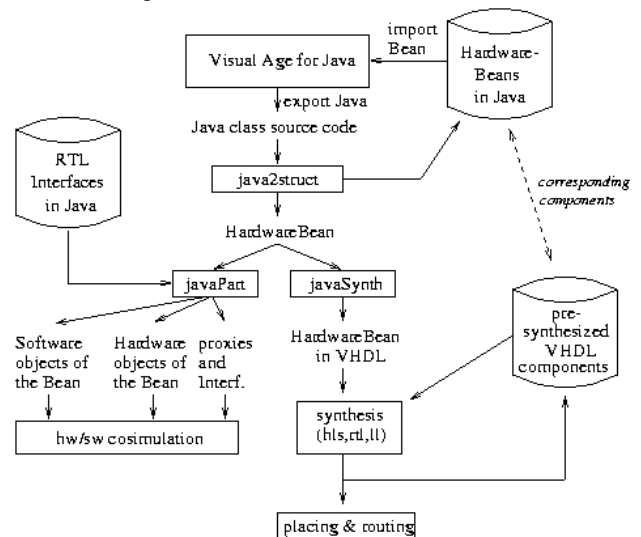


Figure 5. Design flow

A hardware object that is to be processed is transformed into

a HardwareBean through java2struct and stored in a Bean library. It's then available for later reuse with Visual Age for Java. The inner life of HardwareBeans can be made of ordinary software objects as well as of other already reused HardwareBeans of this library. There are now two alternatives regarding the further processing of the HardwareBean.

The first alternative is the tool javaSynth. It creates the ports of a VHDL component using the setters and getters. The inner life of this component comes about through the conversion of objects held in the HardwareBean. The conversion is done regarding to the different interpretations. All held HardwareBeans are treated according to structural interpretation, and taken from a VHDL library and instantiated as already synthesized components. After its synthesis, the total produced component is assigned to this library as well. So for every HardwareBean in the Bean library there is a corresponding synthesized component in the VHDL library. All held software objects are mapped onto connections and their methods are converted to VHDL components/processes and synthesized using a commercial high level synthesis system. Some limitations in the Java language must be fulfilled in order to enable the mapping from Java to VHDL processes. For example it's not allowed to use dynamic data structures, floating point operations and so on. But all object oriented techniques like inheritance, polymorphism, multithreading etc. are possible anymore.

The second alternative consists of the treating of the inner life of the HardwareBean produced by java2struct as a hardware/software system. The objects are not converted into VHDL. Instead, they are kept unchanged and used for simulation/prototyping. The software objects make up the software part of the system, while the hardware objects make up the hardware part. This will be further explained in the following section.

5. SIMULATION

The javaPart tool was implemented in order to simulate hardware/software systems. The software objects of the HardwareBeans make up the software part of the system, while the hardware objects serve the simulation of the hardware part. Fig. 6 provides an overview.

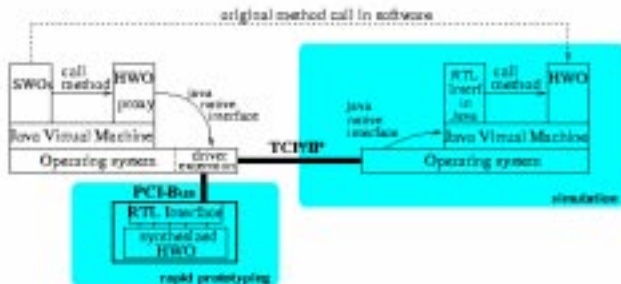


Figure 6. Hardware/Software cosimulation and prototyping

Each accessing of a software object to a hardware object is mapped to a proxy object. This represents the interface between hard- and software. If the proxy object determines

that the hardware, for example in form of a PCI card, is available, the call will be sent on to the real hardware. If it is not available, the HardwareBean existing in software will be addressed. In the latter case the simulation serves the functional verification of the hardware/software system and the creation of profile data in the hardware/software interface represented by the proxy object. The proxy object is also a HardwareBean. Along with it there exists a synthesized component in the VHDL library. A synthesis with javaSynth leads to an inclusion of these interface component. In it, a communication protocol is implemented, one which also makes possible the concurrent execution of the hardware part. Callbacks from hardware to software are also possible (see also [2]).

6. EXAMPLE

The applicability and productivity of the introduced concepts should be demonstrated with a JPEG codec. An object-oriented JPEG codec was described in Java for this purpose. To present the source code here would be to much. All essential information is shown more compactly in the UML collaboration diagram [5] in fig. 7.

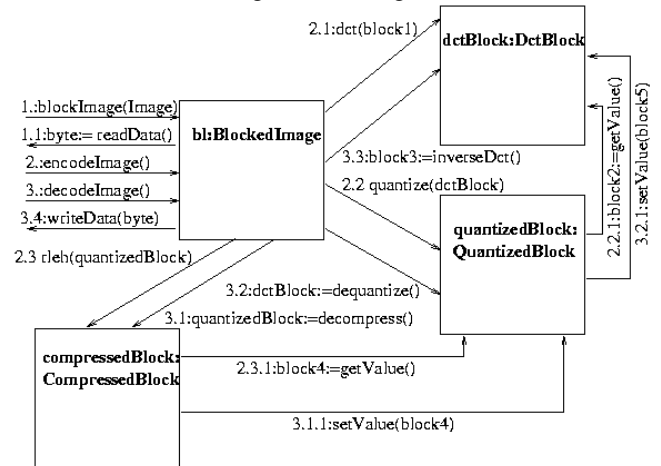


Figure 7. UML collaboration diagram of a JPEG codec

The boxes are objects and arrows represent method calls. Each method call is equipped with a leading sequence number, in order to express the chronological succession of the calls. A class diagram would show that the classes DctBlock, QuantizedBlock, and CompressedBlock have a using-relationship with BlockedImage, that is, they are used within BlockedImage.

The meaning and possibilities of the object models' utilization will be demonstrated in three scenarios.

First of all, all objects are interpreted as software objects. As the surrounding class BlockedImage will be handled according to the design flow. Its methods are expanded through java2struct in order to include the necessary getters, setters, and events. A HardwareBean is developed which is stored in the Bean library. From now on, it can be utilized in other behavioral and structural descriptions at the system level. The developed HardwareBean fulfills all requirements

to be utilized within Visual Age for Java. This HardwareBean is now converted into a VHDL description through javaSynth. The conversion orients itself according to the introduced object model: all objects are converted to connections, and all methods to components. Although this paper will not go into details of this conversion, it should be viewed schematically. The application of the behavioral interpretation to the individual objects leads to fig. 8

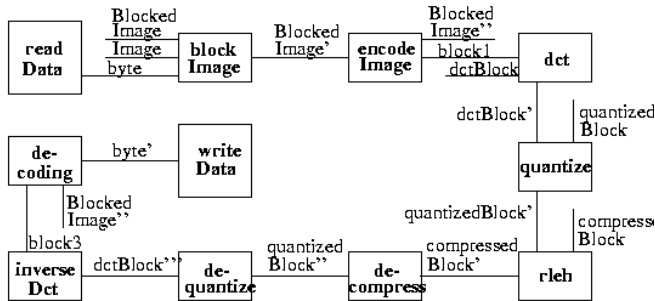


Figure 8. JPEG after behavioral interpretation

Their division into separate blocks is of a merely logical nature. The conversion into VHDL comes about through an in-lining of the blocks in a joint VHDL process. Hereby the total optimization potential is maintained; the objects reuseability is, however, limited to the reuse of their descriptions. The developed VHDL description is synthesized through a high level synthesis and stored in the VHDL library.

In the second scenario, the domain overlapping description is to be demonstrated through a mixture of behavior and structure. For this purpose, the design flow described in the first will first of all be passed through with the class DctBlock. A HardwareBean and a circuit are created. Because the transformation into a HardwareBean only carries out an extension of methods, this HardwareBean can be employed in the JPEG description just as the original can.javaSynth recognizes, in the conversion to VHDL, that it is dealing with a HardwareBean, and therefore handles this object according to structural interpretation. The object is therefore no longer converted to a connection, but rather to a component. Methods, on the other hand, are converted to connections. The resulting structure is shown in fig. 9.

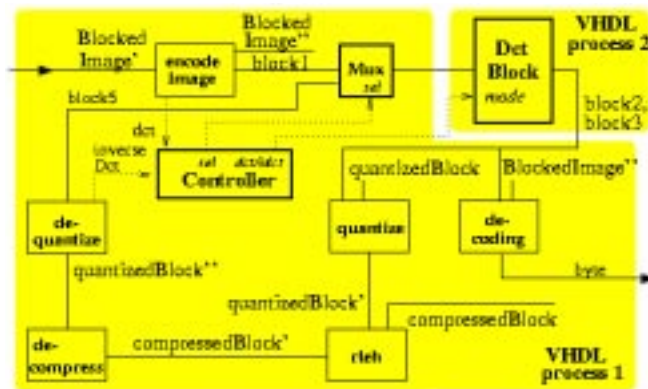


Figure 9. JPEG after mixed interpretations

This time DctBlock represents a physical component. During synthesis DctBlock is treated as a black box, since it already exists in a synthesized form in the VHDL library. With this, optimization potential is lost, but reuse is extended through the reuse of already synthesized components. For the user this is completely transparent. He merely uses objects from different libraries, whose handling are by no means distinguishable.

A look at fig. 5 shows that, towards the end of scenario 2, an employment of javaPart is also possible. In distinction to javaSynth, javaPart interprets the HardwareBean *DctBlock* included in the description as the hardware partition of a hardware/software system. The other objects are not synthesized in order to realize their behavior, but rather interpreted as software partitions of such a system.

7. CONCLUSION

The paper concentrates on the domain overlapping aspects of object-oriented modeling and simulation of hardware and hardware/software systems. A model based on different object interpretations was presented. The model allows the description and combination of behavioral and structural descriptions. The software component model JavaBeans was expanded to the field of hardware description and integrated within a model based design flow. Structural and behavioral descriptions can be done graphically with standard software tools and different kinds of reuse (IP reuse of descriptions and reuse of pre-synthesized components) are combined and supported within one unitary language and on a high level of abstraction. The border between hardware and software becomes transparent for the designer. A cosimulation system allows simulation and prototyping of hardware/software systems specified in Java.

8. REFERENCES

- [1] Helaihel, R., and Olukotun, K. Java as a Specification language for Hardware-Software Systems. In *Proc. ICCAD'97*
- [2] Kuhn, T., and Rosenstiel, W. Java Based Modeling and Simulation of Digital Systems on Register Transfer Level. In *Int. Workshop on System Design Automation*, Dresden, 1998.
- [3] Liao, S., et. al. An Efficient Implementation of Reactivity for Modeling Hardware in Scenic Design Environment. In *Proc. of the 34th DAC*, 1997.
- [4] Nebel, W., and Schumacher, G. Object-Oriented Hardware Modelling - Where to apply and what are the objects? In *Proc. of the Euro-Dac '96 with Euro-VHDL*.
- [5] Rational Software Corporation. URL: <http://www.rational.com/uml>
- [6] Swamy, S., and Molin, A., and Covnot, B. OO-VHDL: Object-Oriented Extensions to VHDL. *IEEE Computer*, October, 1995
- [7] Young, J.S., et al. Design and Specification of Embedded Systems in Java Using Successive, Formal Refinement. In *Proc. of the DAC'98*, 70-75