REDUCING SOC DESIGN EFFORT BY ABSTRACTING COMMUNICATION DETAILS USING A ESL CENTRIC SERVICE BASED UML PROFILE

Millena Gomes, Cristiano Araújo, Adriano Sarmento, Ciro Ceissler, Josiane Bezerra, Denys Farias

Informatics Center (CIn) – Federal University of Pernambuco

ABSTRACT

This paper presents UML-ESL, an UML 2.0 profile that supports Electronic System Level for SoC design. The proposed UML profile supports hardware/software system modeling and partitioning at an abstraction level higher than current UML 2.0 based approaches. The proposed approach reduces the system modeling effort by abstracting its communication details. Designers are relieved from the tedious and error prone tasks of modeling interface, ports, channels and protocols. Using the proposed profile, the designer models communication as services and focus on the modeling of high level service features including synchronicity and parallelism, and also service performance constraints. The paper also highlights how the communication modeled using this profile can lead to a SoC communication infrastructure. The proposed profile was validated with the modeling of an image segmentation hardware/software system and the generation of a virtual platform that implements that system modeled in UML-ESL.

1. INTRODUCTION

Modern complex embedded systems include the use of SoCs (System-on-chip). These are highly integrated circuits composed of heterogeneous components like processors, memories, interconnection structures, and peripherals. The use of SoCs implies that parts of the functionality of the circuit are implemented as software applications running on the processors and the rest as hardware components. Communication between these parts can be synchronous or asynchronous and occur sequentially or in parallel. In order to implement hardware/software communication the designer must develop device drivers, and hardware elements, which consist of interrupt controllers, device controllers and hardware interfaces between those components and the interconnection structure.

The complexity of such systems resulted in the research of high level specification and modeling mechanisms. In particular, the use of higher abstraction levels for faster modeling and better simulation performance. Despite these approaches are commonly known as Electronic System Level [1], they differ in the language and abstraction levels supported. For instance, in the last years, several languages have been proposed as a solution for high abstraction levels implementation. Initial approaches have used programming languages, in particular C/C++, for the modeling of systems. Later hardware support libraries on top of these programming languages like SystemC were largely accepted and used to model and implement the modern systems, since it allows the hardware and software implementation. In recent years, UML 2.0 has gained space as an alternative to describe hardware/software systems due to its language independence and capacity to define domain specific profiles [3]. One example of the use of UML 2.0 for SoC design is the UML profile for SoC [2], which is tailored for the modeling of hardware.

Even using UML the abstraction level of current approaches is not high enough to ease design effort. Most of the proposed UML 2.0 profiles simply allow the modeling of hardware in UML. Details of the architecture, in particular communication, need to be provided. Designers still have to specify ports, interfaces, channels and protocols in order to model communication similar to its implementation, thus reducing the effort of model implementation. These are time consuming and error prone activities. So, the problem is how to define a mechanism for modeling hardware/software systems for implementation as SoCs, that abstracts communication details, provides means to the designer for high level configuration of the communication (ex: synchronicity and parallelism) and finally provides a path for the implementation of the system.

This paper presents a UML 2.0 profile, the UML-ESL, which supports hardware/software systems modeling by abstracting communication details. Its contribution is the definition of a UML profile that supports the hardware/software partitioning of the system, the modeling of communication at a high abstraction level using services calls, and the services classification (see Table 1). The paper also highlights how the proposed profile can be used for the generation of the communication infra-structure.

The rest of this paper is structured as follows. Related work is presented in section 2. The system level modeling is presented in section 3. The UML-ESL profile to enable system level modeling is described in detail in section 4. The results are discussed in section 5. Finally, some conclusions are given in section 6.

2. RELATED WORK

Aiming the improvement of UML model expression power for SoC design, some UML profiles and tools were defined. Mueller [4] proposed an UML 2.0 profile for SystemC based on the UML profile for SoC and a tool that generates SystemC code from UML diagrams. Muller also demonstrates how to model hardware and software together in a unique UML base environment. This profile supports ISS (Instruction Set Simulator)

encapsulation in UML diagrams and allows designers to model systems composed by elements implemented in C/C++ and SystemC. However, in Muller's approach UML designs are merely representations of SystemC models. Designers must still model communication in detail, including ports, interfaces and protocols

Riccobene [5] proposed a tool for hardware and software co-design using UML and SystemC. Riccobene`s tool employs an UML platform-independent model (PIM), that can be mapped onto a platform-specific model (PSM), which contains design and implementation details. The tool supports forward/reverse engineering to/from C++ and SystemC code generation. Using UML class diagrams designers are able to model the system as a set of modules. However, the communication must still be modeled by the use of ports, channels and interfaces. As before, communication details are still not abstracted in a level high enough.

As it can be seen, current approaches fail to provide a mechanism for modeling hardware/software systems that abstracts communication details and is also suitable for SoC design implementation.

3. SYSTEM LEVEL MODELING

This paper proposes a high abstraction level mechanism for system modeling, where the system is modeled in the architectural domain, in which it is partitioned as computing elements that communicate through services. Each element is classified after the partitioning as hardware or software. One example of system modeling is shown in figure 1. It is composed of two modules *Skin* (software module), and *Mahalanobis* (hardware module), the later provides the *mahalanobisDistance()* service.

Services implement modules functionalities. They are provided to other modules in the system. Services are composed by input data, and a result that is made available for its caller after its execution. In this context a transaction is defined as the execution of a service, from its call until it returns a value. One example is shown in figure 2 where service 1 is called by both Module A and Module C.

Transactions are classified according to the categories proposed by Goniat [6] and summarized in Table 1. The first category is type and it can be classified as synchronous or asynchronous. In a synchronous transaction both the caller module and the service instance are blocked until the result is received by the caller module. That means that the service instance is not available for any other call. On the other hand, in an asynchronous transaction, services may return *service_non_available* responses that resume caller modules execution.

Table 1 Communication classification summary table

Characteristic	*Classification*	
Type	Synchronous	Asynchronous
Mode	Sequential	Parallel

The communication mode corresponds to the way or sequence that services are executed, and are classified as sequential or parallel. Figures 2 and 3 show examples of parallel and sequential communication respectively.

High level timing constraints may be applied to a service in the communication modeling. In this work the following constraints can be applied: the minimum, average or maximum execution time of each service.

4. UML PROFILE FOR ESL DESIGNS

This section presents UML-ESL, a UML 2.0 profile for ESL, which enables hardware/software systems modeling at system modeling abstraction level presented previous sections. Using UML-ESL the designer reduces effort to model hardware/software systems by abstracting communication details. It provides mechanisms for easing the specification of system partitioning in hardware and software, and the modeling of services including the specification of services constraints.

The system modeled in UML-ESL, contains a smaller number of classes compared with the same system modeled in the UML profile for SoC, since it abstracts the communication details. Partitioning information is presented in this model by the classification of the modules in *sw_module* and *hw_module* presented in those modules stereotypes.

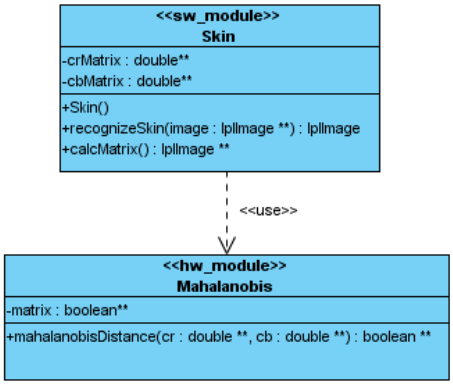

Figure 1 Class Diagram Example at System Level Modeling

The image segmentation of the Vmote design will be used to explain how to model systems architectures composed by hardware and software in a unique UML based environment. The image segmentation system example is composed by two modules, *Skin*, which is in charge of the acquisition of an image and its conversion to a pixel matrix, and *Mahalanobis*, which is responsible for the execution of the mahalanobis distance segmentation algorithm. The following sections will present how to model the system architecture, communication and design constraints using the UML-ESL profile.

4.1 Architectural Modeling

System architecture is modeled using class diagrams, which are responsible for the hierarchical modules representation and their structures, represented by its attributes and services. In the class diagram the designer partitions the system in hardware and software parts. In order to support hardware/software partitioning, UML-ESL provides stereotypes *hw_module*, used to identify modules that have to be implemented as hardware

components, and *sw_module*, used to identify modules that have to be implemented as software applications. Figure 1 presents an example of the partitioning of part of the VMote system. The *Skin* class is defined as a software module while the *Mahalanobis* class is defined as a hardware module.

Each module is formed by attributes and operations, represented by the functions implemented by the module. The configuration parameters are identified by the parameters in the constructor. These can be configured by the designer. The modules of the example showed in do not have any parameter. In the UML-ESL profile, services are identified as the operations that are visible to external modules, i.e. public functions. Private functions are not considered services in this approach, they are simply internal operations.

Class diagrams are also employed to specify interactions between modules. The "use" arrow connecting two modules indicates that one module uses services provided by the other.

4.1.1. Architectural Implementation

The hardware and software modules of the system can be mapped into a base platform composed by processor(s), bus and memory. The hardware and software modules can be implemented as hardware elements and applications running in a processor, respectively. The communication among them is implemented by the bus and extra components integrated into the platform, as wrappers and drivers to allow the communication among the platform elements.

4.2 Communication Modeling

Communication among modules and transaction parameters are modeled in sequence diagrams. This diagram is used to represent interactions among modules, through messages that call their services. Services calls, including their response, will be called transactions in this paper.

In the sequence diagram the designer is able to classify transactions according to the criteria defined in Table 1. The type, synchronous or asynchronous, is selected by the type of the arrow used to identify the service call. Figure 2 shows the difference between the representation of synchronous messages, represented by filled arrowheads (service 1), and asynchronous ones, represented by stick arrowheads (service 2). In Figure 3 the *mahalanobisDistance*() service is an example of synchronous service.

Transactions execution sequence is modeled in the sequence diagram by the order in which they appear considering the vertical axis, which is the time representation. In the sequential diagram the designer can define the transaction mode to be sequential or parallel. The communication mode used is represented by the frame *namebox* presented in the right top of the frame. The parallel combination fragment is drawn using a frame with the text "par" in the frame's namebox, as showed in figure 2. Each operand in the frame represents a thread of execution performed in parallel.

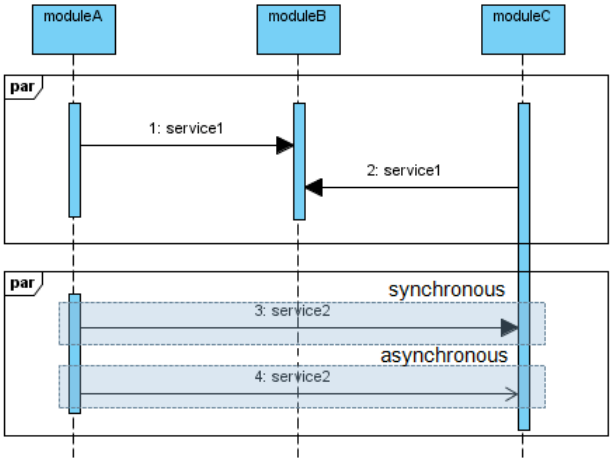

Figure 2 Parallel communication representation example

The sequential combination fragment is represented similar to parallel communication by using the text "seq" in the frame's namebox, as showed in Figure 3. As it can be seen, the transaction between the *Skin* and *Mahalanobis* modules is made by the *mahalanobisDistance* service. This service is executed sequentially to the image matrix calculation (*calcMatrix* service). Notice that the use of "seq" frames is optional, services sequential execution may be modeled just by inserting service calls without any other extra information.

4.3 Communication Constraints

Sequence diagrams are also used to specify communication constraints. The user is able to define maximum, minimum or average time for each service. Constraints are represented in the diagrams as comments associated to the transactions. As showed in Figure 3 the comment indicates a maximum transaction time constraint of 1ms for the *mahalanobisDistance* service from the *Mahalanobis* hardware module. Constraints are identified by reserved words defined by the profile: *maximum_time*, *minimum_time*, *average_time*, and *data_size*.

4.3.1. Communication Implementation

Service transactions and constraints can be converted to the communication architecture shown in figure 4. Transactions are composed of a request and a response packet. The request packet is composed by control information (identification of the service that will be executed and the data size) and data. The response packet is composed by the data generated by the service and a status. On the software side transactions are implemented as device drivers. On the hardware side transactions protocols are implemented by the device controller that interacts with the device drivers. The device controller manages the requests and responses to the appropriate service instance. It is responsible for: (1) converting bus data into the data accepted by the hardware, (2) implementing the communication infra-structure to break or group the data in order to send it to the hardware and (3) to reorganize the data into a format accepted by the

rest of the system. The device controller, as presented in figure 4, is directly connected to the hardware element and to the bus through a wrapper, the bus-controller interface, which converts the bus protocol to the controller supported protocol.

The communication elements are implemented as parameterized templates in a communication library. Specific instances for each SoC platform are generated from these templates based on the UML-ESL specification of the system.

Asynchronous transactions, as an example, are implemented by the controller using internal buffers to store received requests as well as extra information added to the request packet to identify each one. Synchronous transactions are implemented to re-pass requests to the device and adjust the type of the data, when it is necessary. For parallel transactions, the service provided by the hardware element can also be duplicated to allow parallelism. Sequential transactions allow only one access at a time to the device respecting the implementation of the communication type (synchronous or asynchronous).

5. RESULTS

The image segmentation system, composed by two main modules, *Skin* and *Mahalanobis*, was modeled at system level using UML-ESL. The system model was composed by a class diagram (Figure 2), where two modules were identified, and a sequence diagram (Figure 3). There were three synchronous service calls. The *mahalanobisDistance*() transactions is between a software and hardware module. This model lead to the generation of one device driver, one device controller and one bus interface. These components were instantiated from templates in a library following the characteristics presented in section 4.3.1.

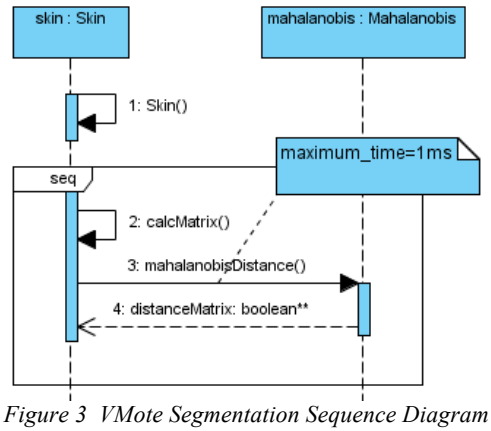

Figure 3 VMote Segmentation Sequence Diagram

The modeled system was synthesized into a virtual platform as presented in the Figure 4, using some previously platform implemented elements (mips processor, simple bus and memory).

Results show the feasibility of modeling SoCs with the proposed UML profile. UML-ESL abstracts communication details, thus reducing designers' efforts, and is still capable of capturing essential systems' characteristics leading to their synthesis.

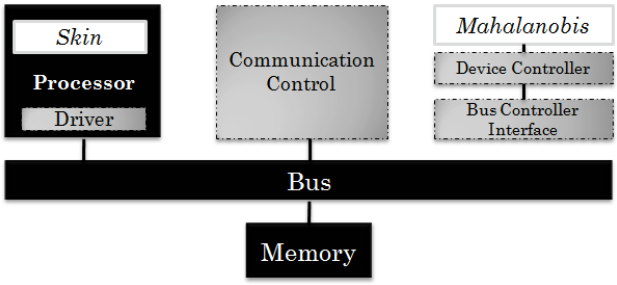

Figure 4 Image Segmentation Application final platform

6. CONCLUSIONS

In this paper, it was proposed an ESL modeling strategy at a high abstraction level. This paper has presented the definition of a UML profile for ESL System Level Modeling. The proposed profile decreases the system design effort by abstracting communication details. It has been shown, from a example, that a path from this high level model to a SoC virtual platform is feasible. As future work is the refinement of a synthesis flow that automates the generation of the communication infrastructure from UML-ESL specifications.

7. REFERENCES

[1] Martin, G., Bailey, B., Piziali, A. 2007. ESL Design and Verification: A Prescription for Electronic System Level Methodology. San Francisco,USA: Morgan Kaufmann, 2007

[2] UML Profile for Soc, OMG, http://www.omg.org/technology/documents/formal/profile_soc.htm, last access 03/11/2008.

[3] Lavagno, L. 2006. "UML: A Next-Generation Language For SoC Design". Electronic Design, May 2006. Available at:http://electronicdesign.com/Articles/Index.cfm?AD=1&ArticleID=12552.

[4] Mueller, W. et all. UML for ESL design: basic principles, tools, and applications. Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design. San Jose, California, 2006.

[5] Riccobene, E., Scandurra, P., Rosti, A., Bocchio, S. "A Modeldriven Design Environment for Embedded Systems".

[6] Gogniat, G., Auguin, M., Bianco, L., Pegatoquet, A. 1998. "Communication synthesis and HW/SW integration for Embedded System Design", in Proc. Of International Conference on Hardware Software Co-design. Seattle, Washington.