UML for ESL Design − Basic Principles, Tools, and Applications

W. Mueller
Paderborn University/C-LAB
Fuerstenallee 11
33102 Paderborn, Germany

A. Rosti
S. Bocchio
STMicroelectronics
Via Olivetti 2
Agrate Brianza, 20041, Italy

E. Riccobene
P. Scandurra
University of Milano, DTI
Via Bramante 65
Crema, 26013, Italy

W. Dehaene
Y. Vanderperren
KU Leuven
Kasteelpark Arenberg 10
3001 Heverlee, Belgium

ABSTRACT
This paper starts with a brief introduction to the UML 2.0 and application-specific UML customizations via profiles. After a discussion of UML design tools with focus on EDA support, we present a HW/SW co-design approach and demonstrate how HW architectures are described together with application SW in a unique UML based environment. Using a dedicated profile providing support for SystemC in UML, and a SystemC wrapper for the SimIt instruction set simulator of a StrongARM, an executable model of the complete architecture is generated which can be simulated by the SystemC kernel. The physical layer of an 802.11a system is used as an application example.

Categories and Subject Descriptors
I.6.5 Model Development

General Terms
Design, Languages, Verification

Keywords
ESL Design, SoC, UML, Profiles, Tools, SystemC, Simulation

1. INTRODUCTION
The OMG (Object Management Group) standard UML (Unified Modeling Language™) has received wide acceptance in software engineering over the last years. Meanwhile, we can find significant investigations how to apply UML for real-time systems [6] and professional development environments with UML support became available in that area. As electronic systems design moved towards software engineering, there is emerging interest for UML within the hardware community [7,28] and different UML diagrams and their variations found their application in:

- requirements specification,
- testbenches,
- architectural descriptions, and
- behavioral modeling.

ICCAD'06, November 5-9, 2006, San Jose, CA

In most cases UML is just applied as a graphical capture, though UML 2.0 meanwhile comes as a computationally complete language based on a generic metamodeling[1] mechanism. However, that metamodel-based approach makes the language definition really complex. Nevertheless, it constitutes one of the key strengths of UML 2.0, providing a flexible foundation for its customization towards different application domains through so-called UML profiles, which currently receives increasing tool support and gives UML great potential to complement current C++-oriented languages for ESL design. In this context, SysML [12] and the UML for SoC extension [17] are already available as OMG profiles for Systems Engineering and SoC application. In this paper, we present a UML profile for SystemC which goes beyond the OMG profile [17] and supports the description of the structural and the behavioral SystemC.

Another great potential lies in UML's structural parts, which apply well to current practices in IP integration and packaging for architectural, component, and interface descriptions. The SPRINT project [27], for instance, investigates how structural parts of UML or related profiles can complement IP-based design flow. Of particular interests are partial cross-compilations between the two XML-based standard exchange formats: the OMG standard XMI[2] [10] and SPIRIT [26], the emerging IEEE standard.

For industrial applications, the availability of appropriate tool support is crucial for deployment of UML for SoC design (e.g., see the UML-SoC workshop survey [28]). As explained in Section 4, UML tools come in different variations based on different UML versions and subsets with the support of specific flows, so that the selection of the appropriate tools becomes a key decision for the successful introduction of UML.

The remainder for this paper is structured as follows. We first give an overview of basic UML 2.0 features and profiles. After an overview of UML related tools and a discussion of tool-related aspects, we introduce a UML profile for SystemC and demonstrate how it can be applied in the context of simulation. Finally, the paper closes by a brief summary and conclusion.

2. UML 2.0
The UML 2.0 standard is defined based on *MOF* (Meta Object Facility) [11], which is a simple language for the definition of

[1] Defining a model by means of a model
[2] XML Metadata Interchange – XML-based format for the exchange of UML models

languages via metamodels. The definition of UML 2.0 is divided into two parts: the *infrastructure* and the *superstructure* [14], where the first one defines the kernel with all core constructs and the latter defines constructs which are visible to the user. The UML standard defines the structure (i.e., syntax) by means of class diagrams with constraints and the semantics through textual outlines. As far as possible, constraints are defined by *OCL* (Object Constraint Language) expressions [13], another OMG standard language in the context of UML.

UML 2.0 itself is composed of a set of structural (classes, components, composite structures, deployments) and behavioral concepts (actions, activities, state machines, interactions, use cases) clearly separating those concepts from their visual representation, i.e., the diagrams composed of graphical symbols. This separation has the advantage to easily replace the graphical representation without changing underlying concepts.

The fundamental structural units of UML are *classes* with attributes, operations, ports, and interfaces with class relationships by means of inheritance, generalizations, and associations. *Components* can be either groups of elements (packaging component) or specialized classes (basic component), which are connected by their required or provided interfaces. *Deployments* and *composite structures* define nested structures. As the latter two compare well to architectural descriptions, they currently find several applications in electronic systems representation.

For behavioral modeling, UML 2.0 has actions, activities, state machines, interactions, and use cases with various graphical representations. *Actions* represent the fundamental behavioral units of UML with functional input/output behavior. Actions can be called by activities and state machines, where *activities* basically compare to Petri-Nets with queuing semantics and control and object token flows. *State machines* represent a variant of Harel's StateCharts [5]. *Interactions* give definitions of partially ordered events and can be represented as sequence, timing, interaction overview, and communication diagrams, where *sequence diagrams* owe very much to ITU's Message Sequence Charts. Though state machines are similar to StateCharts it has to be noted here that UML defines a so-called Run-To-Completion semantics based the sequential selection of events from an event pool.

Current electronic system designs more often apply variants of state machines for modeling, mainly due to more intuitive synthesizable HDL code generation and available tools. Sequence diagrams currently come more into application when interactions of objects, like protocol descriptions, are sketched as well as for testbench definitions.

3. UML PROFILES

3.1 Defining a UML Profile

The possibility to define *profiles* is a standard extension mechanism provided by UML [14]. Profiles allow customizing the UML so that any system could in theory be modeled at any level of detail. A profile is made of a set of stereotypes, tagged values, and constraints to define how the syntax and the semantics of the UML metamodel are extended for a specific domain terminology. *Stereotypes* are specific metaclasses (classes in the metamodel), *tagged values* are standard attributes of metaclasses.

A profile provides mechanisms for specializing a reference metamodel, like the UML 2.0 standard metamodel [14], for its customization to a specific application domain. The profile adds information to adapt the metamodel to a specific problem domain. The profile cannot remove information, e.g., constraints, from the reference metamodel. Because profiles extend a metamodel, they are derived from the Meta Object Facility (MOF) definition and provide just enough extensibility to create line of sight while avoiding the complexity of defining a new metamodel.

Profiles can customize the UML reference metamodel with constraints, tagged values, and extensions. An extension indicates that the properties of a metaclass are extended through a stereotype. An extension is a kind of association. One end ties the association to the metaclass. The other end ties the stereotype to the metaclass it extends. Extensions contain an isRequired attribute indicating whether an instance of the stereotype must be created when an instance of the extended class is created.

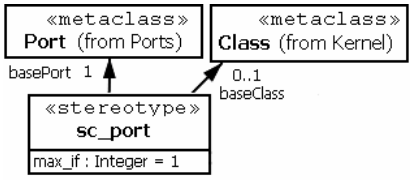


Figure 1. Stereotype definition for sc_port

Within the profile package, a class of the UML metamodel that is extended by a stereotype is shown as a conventional class with the optional keyword `<<metaclass>>`. A stereotype is depicted as a class with the keyword `<<stereotype>>` (see Figure 1). The extension relationship between a stereotype and a metaclass is depicted by a solid black arrow pointing towards the metaclass.

When applied to an element in a model, a stereotype is shown as a keyword consisting in the name of the stereotype within a pair of guillemets, near the symbol of the element or with the special icon defined for it (if one was defined for it) in place of the conventional symbol for the element.

Constraints are semantic conditions or restrictions and can be applied to stereotypes. It is possible to define additional semantic constraints expressed as OCL (Object Constraint Language) [13] formulas over its base metaclass. A constraint enforces a semantic restriction of the extended modeling element. Table 1 shows an example of semantic constraint in OCL.

Table 1. OCL constraint example for an sc_port

English: An sc_port can have exactly one *required* interface and no *provided* interfaces.
OCL: baseClass.required->size()=1 and baseClass.provided->size()=0

Tagged values define properties as name/value pairs and they can be attached to any modeling element. They can describe properties such as the initial value of an attribute, or the fanout of a port, like the tagged value of the sc_port max_if in Figure 1.

A UML profile is therefore based on the following elements:

1) the UML reference metamodel or a subset of it
2) a standard graphical notation for the *set of extensions* (i.e. stereotypes and their tagged values)

3) OCL and natural language to write further constraints

Another fundamental requirement about a profile is that it must be formally defined and machine-readable, from the definition of the profile it is possible to derive a specification in XMI [10] format that can be used in practice to import the profile in the UML tools.

In order to provide a co-design flow based on UML, we need to extend the UML with capabilities for modeling the hardware. We achieve this by defining a *UML profile for SystemC* as introduced in Section 5.

3.2 Relevant UML Profiles

There already exist several profiles available as OMG standards or close to final OMG adoption. Hereafter, we have summarized those which are of particular interest for SoC and embedded systems design; see [7] for more details. It has to be noted here, that the OMG does not prescribe compatibilities between different profiles, i.e., definitions between may overlap and contradict. Some cannot easily applied to all cases since they still refer to UML 1.x.

The *Schedulability, Performance, and Timing Analysis* (SPT) profile [16] provides constructs to represent more easily the kinds of timing and performance artifacts useful in embedded real-time systems, such as Rate Monotonic Analysis (RMA) and Deadline Monotonic Analysis (DMA).

A related profile is the one for *QoS and Fault Tolerance* [15], which defines the notion of concurrently executing resource-consuming components (RCC). This profile covers real-time issues with a focus on communication policies and their latency with hard and soft deadlines. However, though the QoS profile has some overlap with the SPT profile there has been no effort yet to combine both on a joint basis.

An additional profile was defined for applications in software, hardware, and protocol testing. The UML *testing profile* [18] gives several definitions for test benches, test architectures, stimuli, and procedures. The standard gives a mapping of the concepts to the ITU standard test language TTCN-3 (Testing and Test Control Notation), which plays an important role in telecom and automotive systems design.

The *UML for SoC profile* [17] is an outcome of the Japanese UML for SoC Forum (USoCF) and mainly defines structure diagrams through specific SoC stereotypes, e.g., SoCModule, SoCPort, and SoCClock, including their SystemC-like graphical symbols as well as modeling guidelines for Systems-on-Chip.

The *SysML* (Systems Modeling Language) [12] was jointly developed as a UML profile by INCOSE (International Council of Systems Engineering), ISO AP233 Working Group, and an OMG special interest group. It basically takes classes, components, composite structures, activities, state machines, use cases and parts of interactions from UML and additionally defines parametrics (i.e., constraints), requirements (mainly defining dependencies and properties in design flows), and allocations, which define cross-association of elements within the various structures or hierarchies of a model, such as the assignment of behavior to structures, for instance. In addition to some new diagrams like the internal block and block definition diagram, SysML also adds different behavioral concepts to activities like rates and probabilities of token flows. Though the SysML specification was recently finalized, several tools with SysML support are already available on the market.

4. UML TOOLS FOR ESL DESIGN

UML tools applicable for ESL design can be classified in four different categories: UML CASE[3] tools, UML modeling tools with real-time support, UML tools with co-simulation support, and UML in-house tools or tool extensions, respectively.

4.1 CASE Tools

UML CASE tools are used to model primarily software systems, support object oriented analysis and design methods, support model exchange by XMI, and provide code generation for languages, such as C++ or Java. IBM Rational Rose, Sparx Systems Enterprise Architect, and Gentleware Poseidon are typical examples of tools which fit in this category. A more detailed classification of this type of tools can be found in [2].

4.2 Tools with Real-Time Support

UML tools which provide (sometimes limited) support for *real-time systems* constitute a second category. Products such as I-Logix Rhapsody (now acquired by Telelogic), ArtisanSW Real-Time Studio or IBM Rational Rose RealTime, provide proprietary mechanisms to represent timeliness properties and execute UML models with timing annotations. Several UML tools, such as I-Logix Rhapsody, Telelogic Tau, and the Mentor Graphics EDGE UML suite, provide support for common RTOS programming interfaces, like Green Hills Integrity or Mentor Graphics Nucleus, respectively.

4.3 Tools with Co-Simulation Support

Third, some tools associate different application areas and are located *at the crossroad of domains*. For instance, Extessy AG Exite is a coupling tool which allows co-simulating Simulink models with UML state diagrams in Rhapsody or Real-Time Studio. As a second example, Real-Time Innovation Constellation is a UML-based integrated platform for real-time systems which can be used for system-level semiconductor design, although it is initially meant for controls and robotics. It provides a graphical environment with execution tracing and data-visualization tools, generation of C++ code with support for concurrent behaviors similar to SystemC, and means to integrate C++ code generated from Matlab/Simulink into the C++ produced by Constellation. These two examples illustrate how Matlab and Simulink can be integrated with UML: by co-simulation in the first case (Figure 2), and by translation to a common language (C++) in the second case (Figure 3). The circular arrow indicates in each situation which model is executed. To the very best knowledge of the authors, co-simulation is currently not supported for Matlab programs written in M. However, several solutions exist which rely on a dedicated coupling tool interfacing the Simulink model via a coupling block which handles the transfer of data. In contrast, the co-translation approach supports both Matlab code and Simulink models, as C/C++ code can be generated using the Matlab Compiler or Real-Time Workshop, and then linked to a C++ implementation of the UML model. Further details about the

[3] Computer Aided Software Engineering

integration between UML tools and Matlab/Simulink can be found in [29].

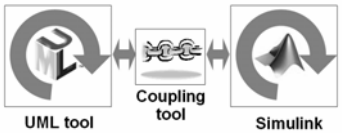


Figure 2. UML and Simulink integration: co-simulation

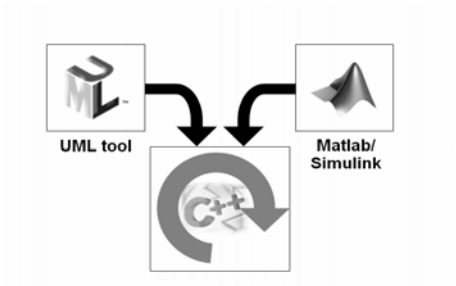


Figure 3. UML and Matlab/Simulink integration: co-translation to a common language (C++)

4.4 In-house Tools or Tool Extensions

In-house tools are developed by users to customize and/or extend the tools in the previous categories with extra features, such as behavioral synthesis on top of Rhapsody [1], SystemC code generation from UML 2.0 in the context of Enterprise Architect [23], or Real-Time Studio [21]. These efforts towards tools providing SystemC code generation realize the idea of associating UML and SystemC and confirm its benefits, as investigated by previous studies and experiences based upon UML 1.x [3,22,31].

Another example of tool extension is the Mentor Graphics BridgePoint model compiler, which transforms UML design models into C or C++ code usually, but allows its rule-based translation templates to be modified so that the same UML model can be used to generate code for other targets, such as VHDL [8].

4.5 Discussion

The previous paragraphs might suggest that heavy UML tools are required to take full advantage of UML. In practice, the selection of a subset of UML appropriate for the application domain, and efficient code generation are sometimes more important aspects to consider. [24] demonstrates, for instance, that state machines can be efficiently implemented in C to fit a low-end 8-bit micro-controller architecture with modest memory fingerprint (4 KB ROM for a non-trivial hierarchical state machine and minimalist RTOS).

In general, it can be observed today that the vast majority of tools do not provide complete support for UML 2.0 and that the interpretation of the UML specification varies among the vendors. Several reasons explain this situation.

First, the complexity of UML makes a selection of a subset appropriate for each particular application domain mandatory. Such subset typically includes class diagrams, composite structure diagrams, object diagrams, state diagrams, activity diagrams, sequence diagrams, and timing diagrams.

Second, UML 2.0 constitutes a major change with respect to UML 1.5 and most vendors have not yet completely adapted their tools for UML 2.0. This lack of support for UML 2.0 is reflected by the discrepancy between the XMI descriptions of UML models. While it should be theoretically possible to export a UML model made with a particular UML tool as an XMI file and import it into the UML tool of another vendor, this operation fails in practice in several cases.

Finally, tool vendors may include some form of support for a particular development process of their own (such as the Rational Unified Process in the case of IBM Rational), which biases the use of UML and the tool support towards that particular process.

The situation becomes even more complex when considering extensions to UML (i.e. profiles). While SysML has already gained the support of several tool vendors who have participated to the standardization effort of SysML, other profiles applicable to SoC design, such as the UML-SoC profile [17], are far less supported. At the time of writing, only ArtisanSW has publicly reported support for this profile in the context of the joint work with Philips Semiconductors [21]. The situation may change in the future if more designers use the profile, now that a first version of the profile has been standardized, and if the profile solves the lack of behavior semantics in UML.

In front of such a complex situation, it is understandable that the lack of appropriate tool support was the weakness of UML which was most often chosen in the survey at last UML-SoC workshop at DAC [28] was the lack of appropriate tool support. Another interesting result was that the design time can decrease by using UML but at the condition that users have suitable tools supporting the design flow, not by virtue of the inherent features provided by UML.

As a conclusion, it is crucial to choose a UML tool according to a well-defined list of user-defined criterion and desired features, in order to assess each tool objectively and understand the strengths and weaknesses of each solution. Such list of criterion includes for example the degree of compliance with UML 2.0, the ease to customize interfaces and code generation, the support for forward and reverse engineering, the degree of compliance with XMI standard, the tool cost, the capability to provide a collaborative modeling environment etc.

5. AN SOC CO-DESIGN FLOW BASED ON UML AND SYSTEMC

To demonstrate UML in industrial application, the remainder of this paper presents an approach to SoC design that allows modeling the hardware and software architecture together. We show how a hardware architecture can be described together with the application software in a unique environment based on UML 2.0, and how we also generate an executable model which can be simulated by the SystemC kernel.

The system architecture we are demonstrating contains an encapsulation in a SystemC wrapper of the SimIt instruction set simulator for the StrongArm processor. It is modeled as an element of a UML composite structure diagram. The architecture contains also other major hardware components modeled using our UML profile for SystemC (that allows embedding SystemC parts within UML 2.0): the memory architecture, the OCCN (On Chip Configurable Network) for the interconnections, and other dedicated components to increase the overall system performance. We provide an application example implementing an 802.11a physical layer transceiver.

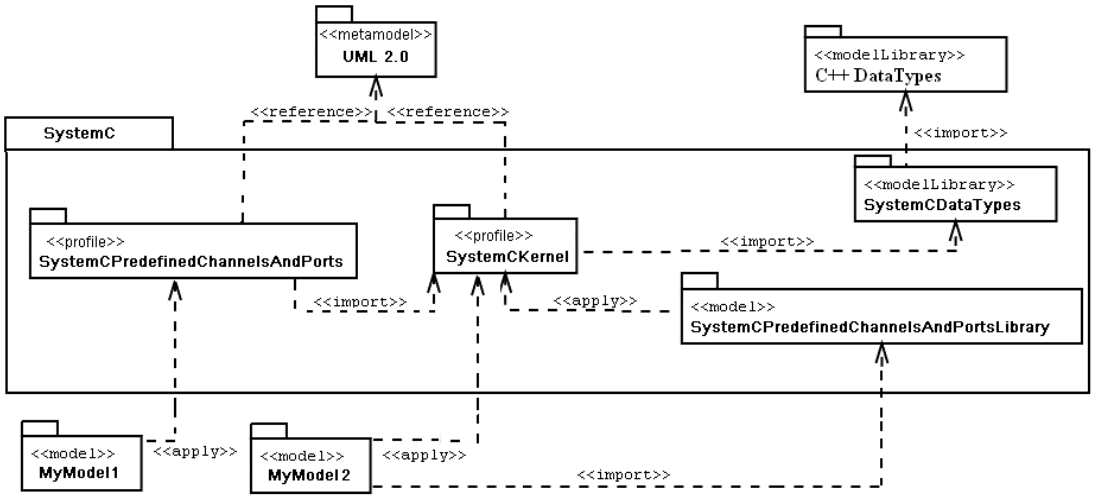


Figure 5. UML for SystemC structure of the profile

5.1 SYSTEMC

Our profile is developed for SystemC application. SystemC [20] is an open standard controlled by a steering group composed of several companies in the EDA (Electronics Design Automation) area. It is one of the most promising system-level design languages intended to support the description and validation of complex systems in an environment completely based on the C++ programming language. SystemC permits to design at system level supporting different levels of abstraction in the same design − functional level, timed or untimed, transactional level, down to bus cycle accurate and RTL − and to perform design refinement in the same language. In our vision, SystemC is a suitable target implementation language, since it is able to represent the system at different levels of abstraction and to mix them in the same design model, allowing describing together the hardware and the software since the early design phases.

Figure 4 shows that the language architecture of SystemC is built on top of the standard C++. The **Core Language** and **Data Types** provide the basic layer of the language, which defines the basic primitives to describe a system (modules, ports, processes, interfaces, channels, events), and new data types more oriented to describe hardware entities than the original C++ data types. The **Elementary Channels** are implemented in a separate layer using the primitives provided by the basic layer.

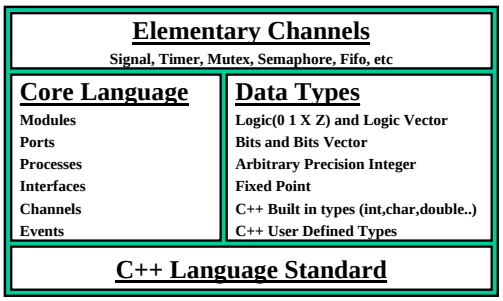


Figure 4. SystemC structure of the language

5.2 UML Profile for SystemC

The main purpose of the *UML 2.0 profile for SystemC* [23] is to provide a graphical entry to SystemC designer, with a special attention to the code generation and back annotation capabilities. Other purposes are the maintainability of the design, the compactness, and expressiveness.

UML 2.0 uses a standard graphical notation for defining profiles: we describe it here briefly together with the definition of our *UML 2.0 profile for SystemC*.

A profile is denoted as a package with the keyword `<<profile>>`. Figure 5 shows the structure of our *UML 2.0 profile for SystemC*. Resembling the structure of the SystemC class library it is organized in two distinct layers: layer-0 corresponds to the part of SystemC that includes the **Core Language** and the **Data Types,** it is modeled by the *SystemC-Kernel* importing the *SystemC-DataTypes* package; layer-1 corresponds to the SystemC **Elementary Channels**, it is modeled by the *SystemCPredefinedChannelsAndPorts* that imports the *SystemCKernel*.

It is important to notice that package *SystemCPredefined-ChannelsAndPorts* is not strictly required for modeling; it just extends the SystemC kernel profile, adding specialized icons and short notations for predefined channels and ports. It is in fact possible to use the package named *SystemCPredefined-ChannelsAndPortsLibrary* that provides the same predefined SystemC channels, interfaces, and ports implemented from the *SystemCKernel*. While modeling, one can choose to apply either the basic layer-0 profile using stereotypes from the kernel and importing the library for predefined channels (long hand notation), or the SystemC layer-1 profile extended with the stereotypes for the predefined channels and ports (shorthand notation).

The profile can then be further analyzed in its four parts:

The SystemC CORE layer: structure and communication in layer-0 defines stereotypes of the core layer of SystemC which

can be used in various UML structural diagrams (i.e., UML class diagrams and composite structure diagrams) to represent SystemC structural and communication elements like modules, interfaces, ports, and channels.

The SystemC CORE layer: behavior and synchronization in layer-0 defines stereotypes of the core layer of SystemC which can be used in various UML behavioral diagrams (such as UML state machines) for behavioral modeling of systems.

The SystemC CORE layer: data types in layer-0 define UML classes for representing SystemC data types.

The SystemC layer of predefined channels, interfaces and ports in layer-1 provides concepts for the layer 1 of the predefined channels, interfaces, and ports of SystemC.

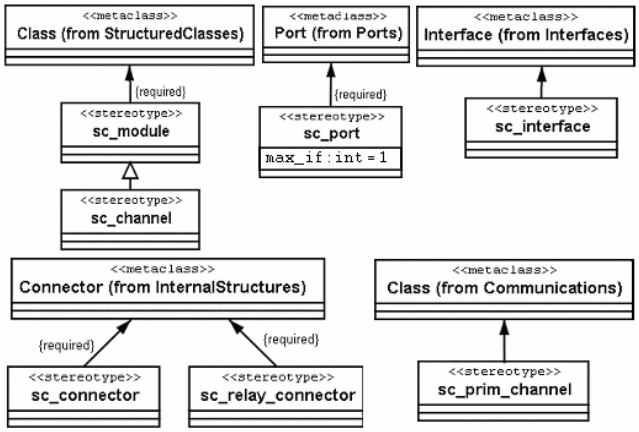


Figure 6. Stereotypes for structural modeling

Figure 6 contains several examples of stereotypes definitions related to the part of the profile that allows describing the structure of a design. At model level, when a stereotype is applied to a model element, an instance of a stereotype is linked to the instance of the corresponding UML metaclass, as shown in Figure 7 where an instance of a class with `<<sc_module>>` stereotype is placed.

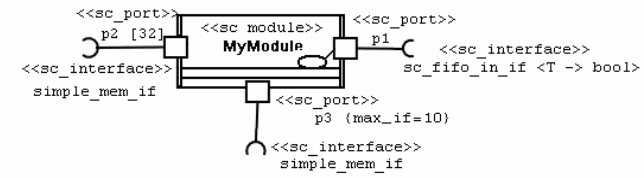


Figure 7. Class with its interfaces

It is then possible to compose hierarchical models of interconnected components as in Figure 8, which shows two instances of SystemC modules which connect through ports and the corresponding interfaces a fifo channel (this element is modeled in the *SystemCPredefinedChannelsAndPortsLibrary*). To complete the description about the features that allow modeling the system structure, we have also to mention the object diagrams. They are similar to the composite structure diagrams, but deal with objects rather that with classes and allow to define actual values of parameters on the object instances.

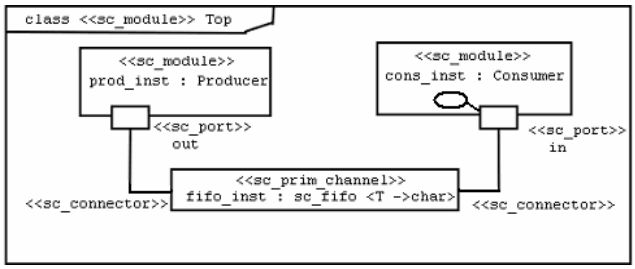


Figure 8. A structural composite class diagram

The most innovative aspect of this *UML 2.0 profile for SystemC* is an enhanced capability for modeling the behavior. In fact behavioral elements such as methods and threads have a double derivation (see Figure 9) both from *Operation* and *StateMachine* allowing to associate behavioral descriptions as extended state machines.

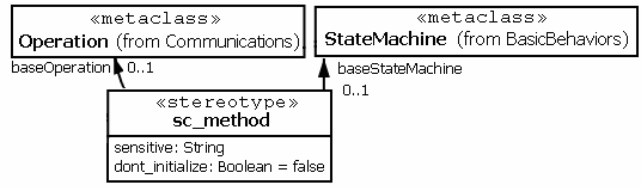


Figure 9. Behavior is operation and state machine

The semantic of state machines diagrams has been enhanced so that they can model the body of any SystemC method or thread, in a one-to-one relationship between the statements of SystemC and the constructs of the state machine. We have introduced all the C control statements (if, switch, for, while, break, return, continue) plus the synchronization primitives from SystemC (wait, next_trigger, event notifications). Inside the states (see Figure 10) it is possible to define directly any SystemC statement by an extended semantics of UML actions. Those state machines have been designed particularly for easy code generation.

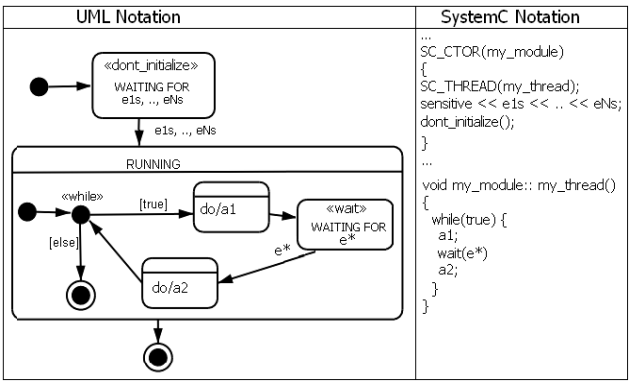


Figure 10. Behavioral state machine

5.2.1 The fifo example

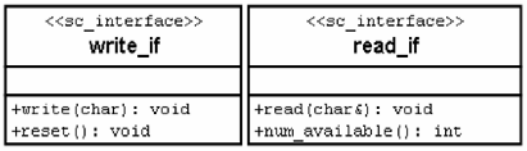
As described before, we can summarize the profile by two main views that contribute to the description of the model:

1) the Structure and Communication view provided by stereotyped UML class diagrams, composite structure dia-

grams, and object diagrams that represent the hierarchical structures and communication blocks of the system;

2) the Behavior and Synchronization view provided by a variation of the UML *method* state machine, called *SystemC Process State Machine*, that specifies the reactive behavior of the SystemC processes, which run concurrently within modules and channels.

We show hereafter the descriptive capability of the SystemC profile for a system that sends and receives data through a FIFO. Instead of using the predefined channel sc_fifo, we provide a custom implementation of a fifo channel. Therefore, the first step is to provide a definition with the SystemC profile constructs of the interface that the channel will implement, as in the upper part of Figure 11; the equivalent SystemC code is shown at the bottom of the figure.




```
class write_if: virtual public sc_interface {
 public:
   virtual void write(char) = 0;
   virtual void reset() = 0;
};
```

Figure 11. Interfaces definition in UML and SystemC code

We use class diagrams to define the basic blocks that will form the system: a producer and consumer module that respectively will send and receive data and the fifo channel. The producer module in UML is shown in Figure 13, while the behavior of the process main is represented in Figure 12.

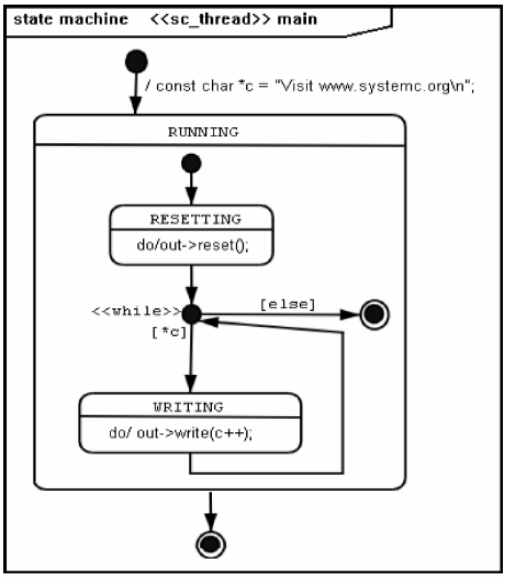


Figure 12. sc_thread main behavior

Figure 14 represents the structure diagram of the system, where the entire basic blocks are connected and instantiated.

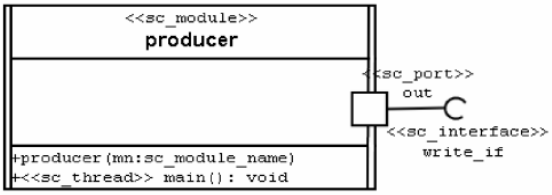


Figure 13. Producer module

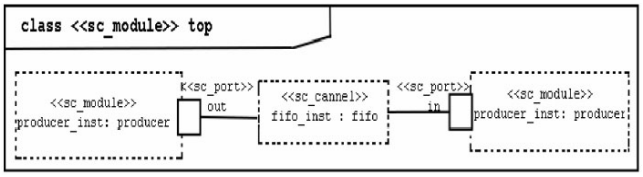


Figure 14. System structure diagram

5.3 Case Study

Following the previously introduced approach, we have implemented an 802.11.a physical layer transmitter and receiver described at instruction level (see Figure 15). Therefore, on one hand the C/C++ application code is encapsulated as a library functions in a UML class. This class provides through ports the I/O interface of the software layer to the hardware system. On the other hand, an ISS encapsulation in UML is also provided, in order to represent all the elements of the system. The UML encapsulation of the ISS is built by the UML profile for SystemC, in order to generate a SystemC wrapper for the ISS and to allow a HW/SW co-simulation at transactional or cycle-accurate level. The application code generated from UML diagrams – Enterprise Architect already provides this feature – is executed by the SimIt ISS [25].

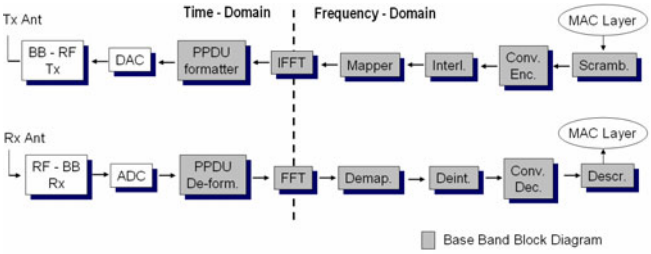


Figure 15. 802.11a TRX block diagram

The system is composed of an ARM processor and a dedicated hardware coprocessor that implements an FFT operation. The processor acts as a master to the hardware module and the memory components, where code and data are stored. The communication is realized by a system bus: we use here On Chip Configurable Network [4], a transactional level cycle accurate SystemC model available at [19]. All the blocks of this system are modeled with UML class diagrams using our profile, in order to automatically generate the SystemC code.

5.4 An ESL Design Environment

Relying on standards, such as UML with its profiling mechanism and SystemC, allows us to easily develop a design environment based on tools that support UML 2.0. We can use such tools as front-end to our design flow with a very low cost of integration. By importing the profile through XMI in the

chosen UML tool (Enterprise Architect, see Figure 16), we obtain a UML schematic entry that can handle SystemC models such as those described in this paper.

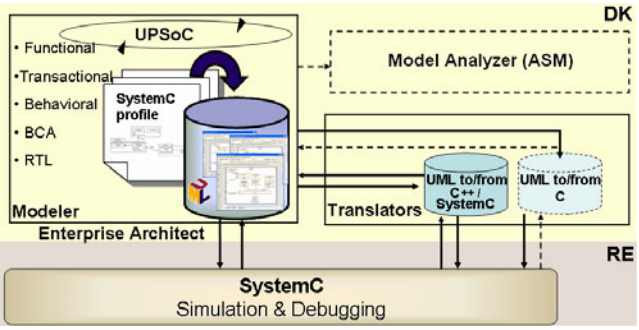


Figure 16. Building a design environment

The rest of the design flow encompasses a code generator that is now mature enough for being used in complex designs and a reverse engineering tool that can import SystemC models in the environment and is currently at a prototypal stage.

6. SUMMARY AND CONCLUSIONS

This paper discussed advanced UML applications for electronic systems design with focus on UML tools and its customizations for SystemC. We have shown how UML profiles can be used within a wider scope of application domains such as the SoC design area. UML profiles provide a standardized visual representation easy to learn and supported by a number of tools to design, implement, and document systems. The possibility to develop a complete design environment, with full round-trip capabilities for code engineering, constitutes one of the benefits provided by a UML extension for SystemC.

Finally, the availability of appropriate tool support is crucial for a successful adoption of UML for SoC design. Several ongoing efforts resulting from collaborations between industrial users, researchers, and tool vendors, constitute steps in the right direction. Similarly to the approach presented in this paper, other positive outcomes have already been reported.

7. ACKNOWLEDGMENTS

The work of Paderborn University is partly funded by the IST project SPRINT (IST-2004-027580).

8. REFERENCES

[1] Basu, A., Lajolo, M., Prevostini, M. *A Methodology for Bridging the Gap between UML and Codesign*. In [7].

[2] Blechar, M.J. *Magic Quadrant for OOA&D* Tools (2H06 to 1H07). Gartner Research Report G00140111. May 2006.

[3] Bruschi, F. et al. *A SystemC based Design Flow starting from UML Models*. In [32].

[4] Coppola, M., Curaba, S., Grammatikakis, M.D., Maruccia G., Papariello, F. *OCCN: A Network-On-Chip Modeling and Simulation Framework*. In Proc. of DATE'04, Munich, 2004.

[5] Harel, D. *Statecharts: A Visual Formalism for Complex Systems*. Science of Computer Programming 8. 1987.

[6] Lavagno, L., Martin, G., Selic, B. (eds) *UML for Real: Design of Embedded Real-Time Systems*. Kluwer, Dordrecht, 2003.

[7] Martin, G. and Mueller, W. (eds) *UML for SoC Design*. Springer, Dordrecht, 2005.

[8] Mellor, S. and Balcer, M. *Executable UML - A Foundation for Model-Driven Architecture*. Addison-Wesley, 2003.

[9] Mellor, S., Wolfe, J., McCausland, C. *Why Systems-on-Chip needs More UML like a Hole in the Head*. In [7].

[10] Object Management Group (OMG). *XML Metadata Interchange (XMI) Specification, Version 2.0*. formal/03-05-02, May 2003.

[11] Object Management Group (OMG). *Meta Object Facility (MOF) Core Specification.Version 2.0.* formal/06-01-01, January 2006.

[12] Object Management Group (OMG). *OMG Systems Modeling Language (OMG SysML) Specification*. ptc/06-05-04, June 2006.

[13] Object Management Group (OMG). *UML 2.0 OCL Specification*. ptc/03-10-14, October 2003.

[14] Object Management Group (OMG). *UML 2.0 Superstructure*. ptc/05-07-04, August 2005.

[15] Object Management Group (OMG). *UML Profile for Modeling QoS and Fault Tolerance Characteristics and Mechanisms, V 1.0.* formal/06-05-02, May 2006.

[16] Object Management Group (OMG). *UML Profile for Schedulability, Performance, and Time Specification, V1.0.* formal/03-09-01, September 2003.

[17] Object Management Group (OMG). *UML Profile for System on a Chip (SoC) Specification V1.0.* formal/06-06-01, June 2006.

[18] Object Management Group (OMG). *UML Testing Profile, V1.0.* formal/05-07-07, July 2005.

[19] OCCN Project. *occn.sourceforge.net*

[20] The Open SystemC Initiative. *www.systemc.org*

[21] Ramanan, M. *SoC, UML & MDA – An Investigation.* In Proc. of the 3rd DAC UML for SoC Design Workshop, San Francisco, July 2006.

[22] Rational Software. *SoC Design with UML and SystemC*. 6th European SystemC Users Group Meeting (ESCUG), 2002.

[23] Riccobene, E., Scandurra, P., Rosti, A., Bocchio, S. *A SoC Design Methodology Based on a UML 2.0 Profile for SystemC.* In Proc. of DATE'05, Munich, 2005.

[24] Samek, M. *UML Statecharts at $10.99*. In Dr. Dobb's Portal, May 2006.

[25] SimIt-ARM Project. *simit-arm.sourceforge.net*

[26] Spirit Consortium. *www.spiritconsortium.org*

[27] SPRINT Project. *www.sprint-project.net*

[28] UML-SoC Workshop at DAC. *www.c-lab.de/uml-soc*

[29] Vanderperren, Y. and Dehaene, W. *From UML/SysML to Matlab/Simulink: Current State and Future Perspectives*. In Proc. of DATE'06, Munich, 2006.

[30] Vanderperren, Y. and Wolfe, J. *Survey of 3rd DAC UML for SoC Design Workshop*. www.c-lab.de/uml-soc.

[31] Vanderperren, Y. et al. *A Design Methodology For The Development Of A Complex System-On-Chip Using UML And Executable System Models.* In [32].

[32] Villar, E. and Mermet, J.P. (eds.) *System Specification & Design Languages*. Springer, Dordrecht, 2003.