# A SoC design flow based on UML 2.0 and SystemC

Sara Bocchio[1], Elvinia Riccobene[2], Alberto Rosti[1], and Patrizia Scandurra[3]

[1] STMicroelectronics, AST Agrate Lab R&I, Italy
{sara.bocchio, alberto.rosti}@st.com
[2] Università di Milano, Dip. di Tecnologie dell'Informazione , Italy
riccobene@dti.unimi.it
[3] Università di Catania, Dip. di Matematica e Informatica, Italy
scandurra@dmi.unict.it

**Abstract.** This paper describes a system design framework for SoC that allows to model together the functional application, the hardware architecture and the embedded software. It relies on a commercial CASE tool that provides a graphical design entry by the UML, we added code generation capabilities to produce an executable model based on SystemC and introduced a reverse engineering flow. We use UML as higher system-level language; it works in synergy with lower level implementation languages: C/C++ and SystemC. As experimental results we present three examples: the Simple Bus (1), the OCCN(2), and the 802.11b(3).

## 1 Introduction

This work is originated from the need to have a design flow for SoC, which starts at high level and integrates embedded processors, memories and hardware. We need a design flow that derives a functional executable model from the specification, maps the functionality to hardware and software, refines the hardware parts through a set of abstraction levels, and compiles the software for a final co-simulation model. The software part can either be simulated on a transactional model of the hardware or it can be compiled and simulated on an Instruction Set Simulator. For such a design flow we need a language that spans all the needed levels of abstraction. We envisage UML [1] as the system specification language. Even if it was born for the specification, design, validation and documentation of software artifacts UML is evolving to model the entire system. We envisage SystemC [2] as the system implementation language, since it provides an answer to the basic needs of system level design.

## 2 The design environment

Our design environment is based on a tool supporting UML 2.0, since only this version provides the needed features to model the system structure. Our implementation is based on Enterprise Architect [3] from Sparxsystem. We add new features to UML to describe more complete system models made of hardware and software. This enhancement is obtained by profiling: first we defined a UML profile for SystemC. Our UML profile for SystemC [4,5] allows describing structural and behavioral features using class, composite class, objects and extended state machine diagrams. It models SystemC components by classes marked with the proper stereotypes (modules, channels, interfaces, port). Behavior is modeled by enhanced state machine diagrams, where we added a set of constructs to model the control flow and we enriched the semantics of states so that they can contain SystemC statements. These behavioral models are conceived for code generation, as far as an isomorphic SystemC implementation can be easily derived out of them. The structural description is completed by the composite structure diagrams that describe the connections of the system components and by the object diagrams that contain also the actual parameters of the objects. We provide a design environment where both the application and the architecture are described together in UML. SystemC models the hardware architecture within UML and provides the overall system simulation environment.

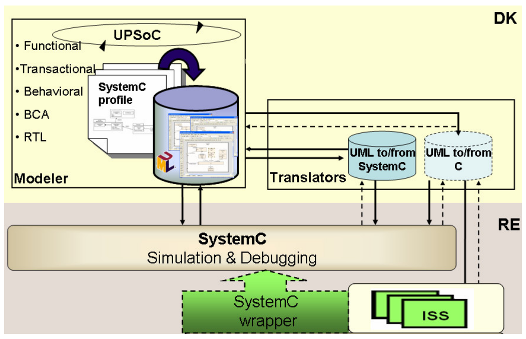


**Fig. 1.** Design environment

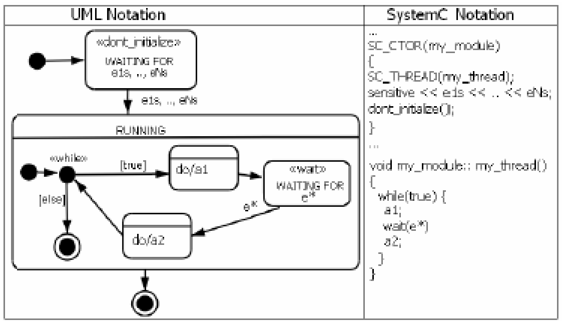
In Figure 1 the design environment that we want to provide is shown. Components visualized inside dashed lines are still under development. The tool consists of two major parts: a development kit (DK) with design and development components, and a runtime environment (RE) represented by the standard SystemC execution engine. The DK consists of a UML 2.0 *modeler* supporting the UML profile for SystemC and *translators* for the forward-reverse engineering to-from C++/SystemC.
To provide a more complete environment for HW/SW co-design, we add the capability to describe also the software application. The application is still modeled by class, state machines and object diagrams by a C profile of UML. In our vision the application model are divided into different groups implemented as threads for an RTOS (like eCos or linux pthread) running in the processor or in a multiprocessor platform. Driving by the idea to facilitate an MDA transformation between the two

profiles, we build the C profile keeping it very close to our SystemC profile (i.e. state diagrams are used to define the behavior) Therefore, the C profile has the same diagrams of our SystemC profile, with the addition of the sequence diagram to model the scheduler that performs synchronization and scheduling of threads. This is still an on-going activity: the profile is almost completed, but no tool support is ready yet (the dashed line in fig.1).

## 2.1 The modeler

The modeler is built on top of Enterprise Architect (EA), version 4.5, a commercial UML visual modeling tool by Sparx Systems [5]. Among all the tools on the market this one seems to be most suitable for our purposes. EA supports UML 2.0 and the UML extension mechanism. It has state-of-the-art development including XMI import/export to allow model interchange between tools, and forward/reverse engineering in the C++ programming language. There is however no reason to use other tools supporting UML 2.0 and the standard extension mechanism of UML profiles.

We introduced the SystemC UML profile definition within the EA tool exploiting the Profile section of the UML toolbox, and saved it in a XML file, with a specific format, for use in UML modeling. The editing task included creating the profile package, defining the stereotypes and the metaclasses they apply to, as well as tagged values, constraints and alternative stereotype images.



```
...
SC_CTOR(my_module)
{
SC_THREAD(my_thread);
sensitive << e1s << .. << eNs;
dont_initialize();
}
...
void my_module:: my_thread()
{
  while(true) {
   a1;
   wait(e*)
   a2;
  }
}
```

**Fig. 2.** A thread process pattern

To start a new design project, the XML file of a UML profile has to be imported into EA. Once imported, the user can drag and drop profile elements onto the current diagram. EA will attach automatically all the extensions (tagged values, default values, etc.) provided by the profile. Stereotypes for the SystemC building constructs (modules, interfaces, ports and channels) are available to be used in various UML structural diagrams such as UML class diagrams and composite structure diagrams to represent hierarchical structures and communication blocks Behavior is modeled by the use of special state and action stereotypes which lead to a variation of the UML state machine diagram, the SC Process State Machines. This formalism has been appositely included in the profile definition to model the control flow and the reactive behavior of SystemC processes (methods and threads) within modules. A finite number of abstract behavior patterns of state machines [6] have been identified. Figure 2 depicts one of these behavior patterns together with the corresponding SystemC pseudo-code for a thread process that: (i) is not initialized, (ii) has both a static (the event list e1s, …, eNs) and a dynamic sensitivity (the state with the stereotype keyword wait), and (iii) runs continuously (by the infinite while loop).

## 2.2 The code generator

Code Generation Facility Driving is one of the key ideas of the OMG's Model Driven Architecture [7]. The goal is to model once, and generate everywhere: models are used to build programs by model transformations. A platform-independent model (PIM) is created in UML without technology dependent details. The PIM can be mapped to a platform specific model (PSM), which contains design and implementation details; the PSM is then implemented in a particular coding language.

Modeling tools support code generation in different ways. In [8] three categories of code generators are identified according to the degree of completeness of the model and of the resulting code: skeleton generation, partial generation, and full generation.

For our SystemC code generator we followed a full generation approach. The EA already supports partial code generation to C++. We added the capability to generate complete SystemC code from UML models for both structural and behavioral views. For this purpose, EA provides automation and scripting interface to customize its user interface and the templates used for code generation. It is based on the Windows OLE Automation (ActiveX) technology: all environments capable of generating ActiveX COM clients are able to connect to the EA automation interface, and this involves also Microsoft Visual Basic 6.0. Alternatively, the XMI import/export [9] facility can be exploited to extract the model information required to generate code for a target language. Although this second solution is more general (it should be independent from the particular UML tool used), the first one is the most straightforward for a rapid development of a code generator. So we decided to rely on the EA automation/scripting interface.

We developed an EA add-in in Visual Basic 6.0 which exploits the added semantics in the profile definition to generate SystemC code from input models written in the SystemC UML profile. This application can be invoked from the main menu selecting **`Tools | SystemC`**. It is possible to generate code from the model, at package level or even for single diagrams. Starting from the selected element in the EA project browser (project, package or class diagram) the code generation analyzes the underlying hierarchy of views generating the corresponding mixed C++/SystemC code. The code generator traverses all class diagrams and for every encountered class it produces a header file (.h). For each method contained in a class, it is possible to describe its behavior either as an inline code description or as a state machine diagram. The state machine diagrams contribute to generate and enrich a single body file (.cpp) that contains the implementation code of all methods of a class or module or channel. The composite structure diagrams and object diagrams are used to derive the module constructors in the header files.

For the SW part, we are working on the C code engineering and on how to support the specification of multiple tasks and their mapping on abstract RTOSs models by including precise modeling primitives directly at UML level.

### 2.2 The reverse engineer

We decide to rely on the XMI [10] import facility to extract from the code all the model information required to have a coherent representation according the profile defined, since the EA automation/scripting interface doesn't allow a fast development (we need to modify the parser from a generic C++ code to a C++/SystemC code). We develop a stand alone application in Java. The software is made of three parts: a parser, a data structure and a XMI writer. The parser for C++/SystemC is build using the JavaCC tool [10] and taking as a starting point the ParSyC [11] grammar file. The component accepts both SystemC code which is translated into constructs of the UML profile for SystemC and C++ code which is translated to UML classes (including the behavioral description). The XMI writer finally produces a UML model that can be imported in the EA tool: for every file .h the tool produces a relative class diagram an for every .cpp file a relative state diagram. The XMI file can be imported from the main menu selecting `Project | Import/Export | Import Package from XMI….`

The reverse engineering facility allows us to import existing models into our environment and to achieve rapidly a high number of design cases. It is also indispensable to allow round trip engineering, which is the synchronized cooperation with code generation (forward engineering) to complete the description of a model working on both the UML model and the code. It is also useful in practice as a tool to inspect the structure of a source code graphically. This tool is a powerful help for IP-reuse and component based design, where the design phase mainly consists building complex architecture from basic existing IPs (that could be automatically imported.)

## 4 Case Study

As experimental results we present three examples:

(1) The Simple Bus is a transactional level example designed to perform cycle-accurate simulation; it implements a high performance, abstract bus model. It contains the three master blocks (blocking, non-blocking, monitor), two slave memories (fast and slow), the bus connecting masters and slaves, an arbiter and a clock generator. Masters issue transactions for the slave memories through the bus. We described this model using the UML profile for SystemC testing our code generator on it. The UML model is isomorphic to SystemC; the generated code is identical to the original SystemC model.

(2)The On-Chip Communication Network (OCCN) provides an efficient modeling of network on-chip (NoC) based on an object-oriented C++ library in SystemC. It spans multiple abstraction levels, from functional, to transactional, to clock accurate, to register-transfer models. OCCN is the main test for our reverse engineering flow. The design was in fact imported in the environment by the reverse engineering flow and then the model was adjusted within the Enterprise Architect tool. OCCN contains a mixture of SystemC constructs and C++ classes; the situation is also complicated by complex inheritances and template classes.

(3) In [12] we already described this application example related to a system composed by a VLIW processor developed in ST, called LX, with some dedicated hardware for an 802.11b physical layer transmitter and receiver. We provide the UML model of the application description and its translation to C/C++ language, it is will be 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. The software part will be executed by the LX ISS: a ISS encapsulation in UML is also provided, in order to represent all the elements of the system.

## 4 Conclusions

This work demonstrated that you can build a system environment taking advantage of general CASE tools based on UML that can easily be integrated in a design flow thanks to standard formats to exchange data such as XMI. Relying on other standard notation as SystemC allows integrating the design flow in a refinement design chain to implementation. The flow is completed by point tools for code generator and a reverse engineering. About the future enhancements we plan to work about a refinement methodology that allows going through different levels, both for the hardware and the software part.

## References

[1] OMG, UML Specification, Document ad/03-03-09, version 1.5.

[2] The Open SystemC Initiative. http://www.systemc.org.

[3] The Enterprise Architect Tool. http://www.sparxsystems.com.au/

[4] E. Riccobene, P. Scandurra, A. Rosti and S. Bocchio. *A UML 2.0 Profile for SystemC*. ST Microelectronics Technical Report, 2004.

[5] E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio. *A SoC Design Methodology Based on a UML 2.0 Profile for SystemC*. In *DATE 05*.

[6] E. Riccobene, P. Scandurra. *Modelling SystemC Process Behavior by the UML Method State Machines*. In *RISE 04*, Springer-Verlag 2004.

[7] OMG, Model Driven Architecture. http://www.omg.org/mda/.

[8] P.-A. Muller, P. Studer, F. Fondement, and J. Bézivin. *Platform independentWeb Application Modeling and Development with Netsilon.* Journal SoSym, 2005

[9] 2005OMG, XML Metadata Interchange (XMI) Specification, v1.2.

[10] Java Compiler Compiler. https://javacc.dev.java.net/.

[11] G G. Fey, D. Groe, T. Cassens, C. Genz, T. Warode, and R. Drechsler. ParSyC*: An efficient SystemC parser*. In Synthesis And System Integration of Mixed Information technologies, 2004.

[12] S. Bocchio E. Riccobene, A. Rosti, P. Scandurra, *A SoC Design Flow Based on UML 2.0 and SystemC*, UML-SoC 05