SysML Profile for SoC Design and SystemC Transformation

Mauro Prevostini, Elena Zamsa
ALaRI, Faculty of Informatics
University of Lugano
via G. Buffi 13, CH-6904 Lugano

May 11, 2007

Abstract

As hardware and software co-design of embedded applications are becoming more and more complex, it is necessary to approach some methodologies that will help to decrease the time and increase the quality of design. A modeling language that try to fill the gap, from a system engineering point of view, is the Systems Modeling Language (SysML): a UML2 profile with extensions. More and more UML and SysML based design is attracting attention on its possibilities of modeling Systems on a Chip (SoCs), because they are platform independent languages, and thus it is possible to design a system without knowing which part of the system will be implemented in hardware or in software. The goal of this work is to propose a SysML profile for modeling SoCs with the aim to transform the model into SystemC code.

1 Introduction

Embedded systems are more and more complex and multiple subsystems are easier to design and programme if we are using a variety of components, hardware and software, both joined into flexible platform architectures. In order to describe these kinds of systems we need a Platform Independent Modeling (PIM) language like the Systems Modeling Language (SysML) [Sysa]. In the meanwhile, because of validating the PIM model, we also need a Platform Specific Modeling (PSM) language like e.g. SystemC [Sysb]. SysML has been proposed by the Object Management Group (OMG) [Omg], together with the International Council on Systems Engineering (INCOSE) [Inc] and AP233 consortium [Ap2] with the aim to define a general purpose modeling language for systems engineering. It is based on the actual standard for software engineering, the Unified Modeling Language (UML) [Uml] version 2.0, with some extensions, and it was developed as a response to the request for proposal (RFP) issued by the OMG in March 2003 [OMG03a] and adopted as a standard in May 2006 [OMG06]. SysML is a modeling language for representing systems and product architectures, as well as their behavior and structure. It tries to adopt modeling techniques known from software development to systems engineering, and supports the specification, analysis, design, verification and validation of a broad range of complex systems. SystemC provides hardware-oriented constructs within the context of C++ as a class library implemented in standard C++. Its use spans design and verification from concept to implementation in hardware and software. SystemC for design implementation is supposed to be a prospective way to survive system's complexity [VSvO02]. In this paper, we would like to present our ideas regarding SysML modeling of Systems on a Chip oriented to SystemC transformation. In particular we will present our approach on how to transform SysML diagrams into SystemC code. Our transformation procedure is taking into consideration SysML structural diagrams only plus allocations of activities. In fact SysML behavioral diagrams are basically the same as in UML2.0, and the transformation of UML behavior modeling into SystemC code has been already investigated since a long time as explained in Section 2. This document is organized as follows: Section 2 describes the related works, while Section 3 describes the proposed SysML profile for modeling SoCs. Section 4 describes our approach on how to map SysML diagrams to SystemC. Section 5 validates the approach by means of a case study. Section 6 concludes the paper.

2 Related works

In this section we present the usage of UML combined to SystemC and related works. To allow using UML for HW/SW co-design, [RSRB05] started to extend UML by a profile for SystemC that allowed to express a SystemC model in UML. A profile is a standard mechanism for extending UML by adding a collection of domain specific notation made of stereotypes, tagged values and constraints, all with the proper semantic. They added the capability to generate SystemC code from the UML model by using the feature to customize the scripting generation capabilities of the UML tool or by exporting the model in XMI (XML Metadata Interchange) [OMG05] format and generating the SystemC code from it. A lot of work has been done in the field of embedded systems, but unfortunately a lot of designers were focused on software design, that excluded hardware part of embedded systems. In [CSL+03] a specialization of UML is presented to express embedded real-time applications in an abstract way. [BS02] defines an UML Profile for SystemC, but no code generation capabilities for behavioral information are considered. Another point of view and proposal was presented in [GT03], the authors provide an analysis about the structural modeling concepts in UML 2.0, which has added structural information such as class and structured class. A lot

of tools are presented that encapsulate the possibility of UML to SystemC code generation. For example the YAML tool presented in [SDS00] which can translate UML class diagrams and object diagrams into SystemC and display block interconnections graphically. Another tool is UMLSC, which can capture not only the static structure, but also the dynamic behaviors of the system under design [XJZY05]. In [MRB+06] authors discuss UML design tools with focus on EDA support, and present a HW/SW co-design approach and demonstrate how HW architectures are described together with SW applications in a unique UML based environment using a profile to support SystemC transformation. As described above we can see that UML to SystemC transformation has been investigated since the last few years, and all this research was considered as the basis in order to provide a UML Profile for System on a Chip officially adopted as a standard by OMG in August 2006 [Gro06]. Although all these research works, we can definitely say that the SysML to SystemC transformation is something new that we would like to approach in this paper.

3 Modeling SoCs with SysML

In this section we propose the way of modeling SoCs by means of a SysML profile. SysML provides a structural element called *block*. Blocks can be used to represent any type of components of the system, functional, physical, and human, etc. and are used within *Block Definition Diagrams*, which aim to describe the structure of the system. The SysML Block Definition Diagram (BDD) is based on UML Class Diagrams and UML Composite Structure Diagrams. It is also used to represent the system decomposition using for example associations and composition relationships.

3.1 Block Definition Diagram

The role of a BDD is to describe the relationships among blocks, which are basic structural elements aiming at specifying hierarchies and interconnections of the system to be modeled. A block is specified by its *parts*, and *flow ports*. Figure 1 shows an example of a small BDD with two blocks, $SoC1$ and $SoC2$, that might represent two SoCs communicating among each other through their flow ports. We can also see that $SoC1$ is composed of two parts: $module_1$ and $module_2$, and one flow port: $flowPort_3$. Parts represent the physical components of the block while flow ports represent the interfaces of the block, through which the block communicates with other blocks. BDDs can also represent the decomposition of a block [HM06] in order to show by which kind of parts, including their features, a block is composed. For example $SoC1$ could be decomposed in its two parts as shown in Figure 2. In order to show the internal structure of a block and the interconnection of its parts, we make use of the SysML *Internal Block Diagram* (IBD).

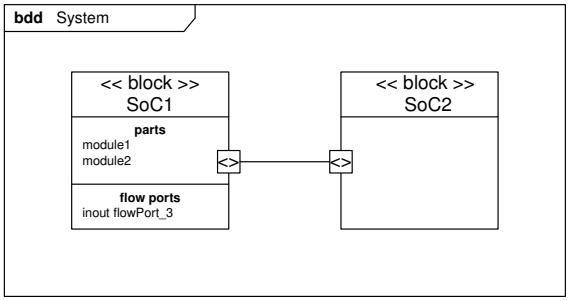

Figure 1: Block Definition Diagram representing the relationships among blocks SoC1 and SoC2

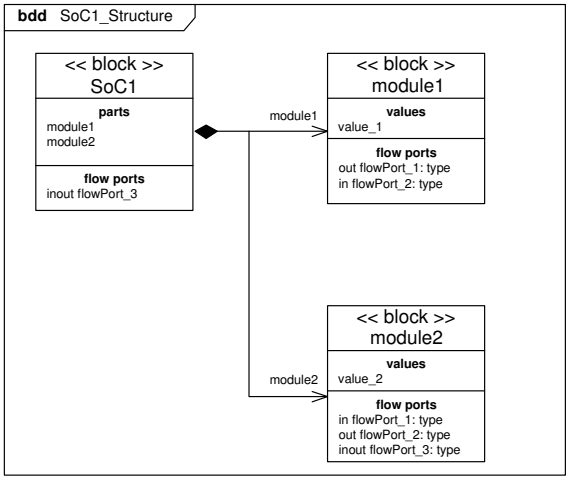

Figure 2: Block Definition Diagram representing the decomposition of the block SoC1

3.2 Internal Block Diagram

The SysML IBD allows the designer to refine the structural aspect of the model. The IBD is the equivalent of the composite structure in UML. In the IBD, parts are the basic elements of the diagram and they are assembled to define how they collaborate to realize the behavior of the block. A part represents the usage of the corresponding block. The most important aspect of the IBD is allowing the designer to refine the definition of the interaction between the usage of blocks by defining flow ports as follows:

- ports are parts available for connection from the outside of the owning block;
- ports are typed by interfaces or blocks that define what can be exchanged through them;
- ports are connected using connectors that represent the use of an association in the IBD.

Figure 3 shows how to represent the internal structure of the block $SoC1$ by means of an IBD. The IBD corresponds to the block decomposition shown in Figure 2 where $module_1$ and $module_2$ are connected each other through *in* and *out* ports, while $module_2$ is connected through an *inout* port to the outside of the corresponding block $SoC1$.

3.3 SysML ports

Two types of ports are available in SysML:

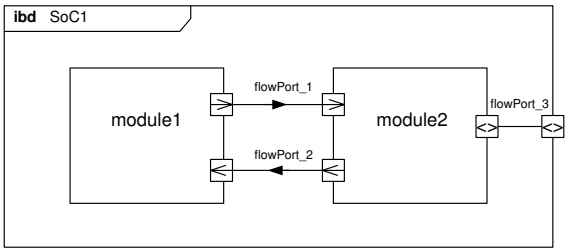

Figure 3: Internal Block Diagram representing the usage of parts within block SoC1

- *standard ports* handling requests and invocations of services with other blocks (basically the same concept as in UML2);
- *flow ports* which let blocks exchange flows of information.

Flow ports specify the interaction points among blocks and parts supporting the integration of behavior and structure. For standard ports, an interface class is used to list the services offered by the block. For flow ports, a flow specification is created to list the type of data that can flow through the port.

3.4 Parametric Diagram

Blocks can be used in multiple context like for example within a SysML *Parametric Diagram* (PD). The role of a PD is to express constraints (i.e. equations) with the aim to provide support for engineering analysis like performance or reliability analysis. Each constraint block captures equations which can be expressed using formal (e.g. using MathML [mat] or OCL [OMG03b]) or informal languages. The mathematical expression can for example represent the physical constraints of the system, and gives us the possibility to easily use them within a tool with computational engine. Such constraints can also be used to identify critical performance parameters and their relationships to other parameters, which can be tracked throughout the system life cycle. Figure 4 shows an example of a constraint block which includes the constraint, such as $value_1 > value_2$, and the parameters of the constraint such as $value_1$ and $value_2$. Constraint blocks define generic forms of constraints that can be used in multiple contexts. Reusable constraint definitions may be specified on block definition diagrams and packaged into general-purpose or domain-specific model libraries. Such constraints can be arbitrarily complex mathematical or logical expressions [Bal06].

3.5 Allocations

Another interesting feature of SysML is the possibility to allocate behaviors to parts. Allocations can be described within Activity Diagrams, as shown in Figure 5, where each activity is allocated to a part of the block $SoC1$. The Activity Diagram in Figure 5 shows the behavior of $SoC1$, which get $value_2$ from $SoC2$

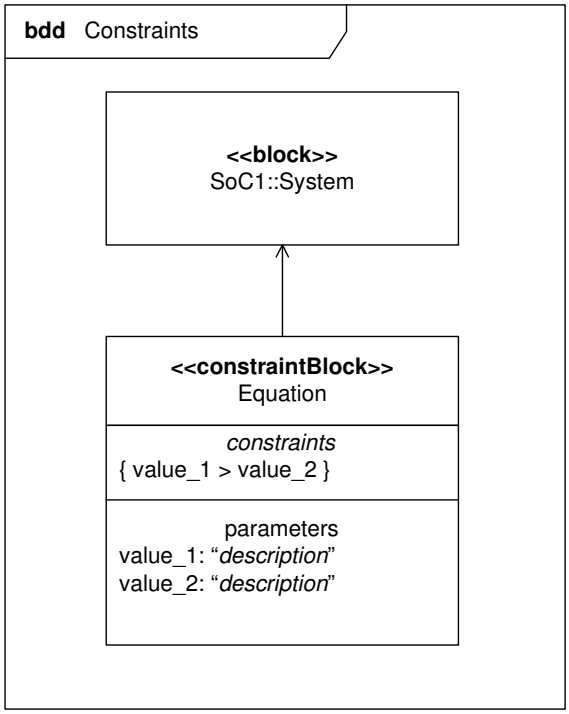

Figure 4: Constraint Block

and increase it for example by one. Then $SoC1$ verifies whether $value_1 > value_2$, which is the constraint that must be satisfied, and, if it is the case it goes ahead reading values from $SoC2$, otherwise it stops working.

4 Mapping SysML Diagrams to SystemC

In this section we would like to propose our approach to map SysML diagrams to SystemC code. As said in Section 1, we will base our mapping procedure on SysML structural diagrams only. The behavioral diagram in Section 3.5 is needed just for process declaration as explained below. We will start by defining basic SystemC entities like *module*, *port* and *process*. The mapping between SysML and SystemC items is defined as follows:

- SysML *parts* mapped to SystemC *modules*;
- SysML *flow port* mapped to SystemC *ports*;
- SysML *allocations* mapped to SystemC *processes*.

Now we are able to firstly create a header file (.h) for each module that must include: *module* definition; *port* declaration; *process* declaration. Secondly we can build the implementation files (.cpp) that describes the module behavior and could be implemented as described in [MRB+06]. The header files of $module_1$ and $module_2$ of $SoC1$ described in Figure 3 can be translated into SystemC code as follows:

```
SC_MODULE (module_1) {

 public:
    sc_out  <int> flowPort_1: type;
```

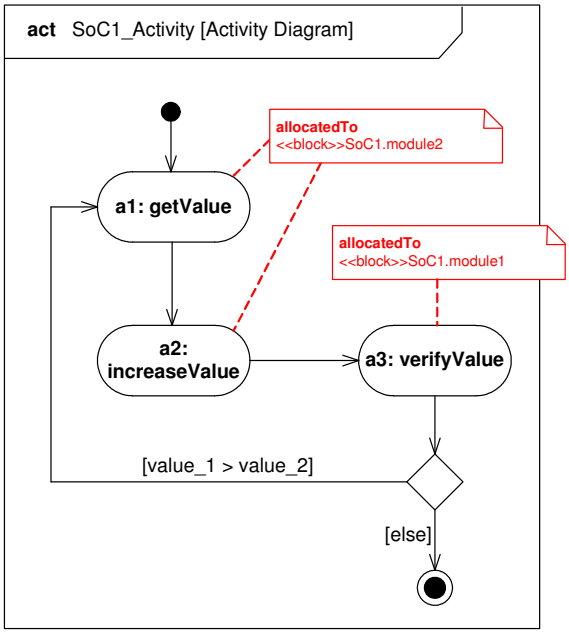

Figure 5: Allocations of activities to parts

```
    sc_in   <int> flowPort_2: type;

 SC_CTOR (module_1)
     {
      SC_THREAD(verifyValue);
     }

  void verifyValue();
};

SC_MODULE (module_2) {

 public:
    sc_in    <int> flowPort_1: type;
    sc_out   <int> flowPort_2: type;
    sc_inout <int> flowPort_3: type;

 SC_CTOR (module_2)
     {
      SC_THREAD(getValue);
      SC_THREAD(increaseValue);
     }

  void getValue();
  void increaseValue();
};
```

When only a single type of object can flow through a port, such a port is named Atomic Port. As can be observed in Figure 2, ports are defined as objects inside a module, and, in order to make them available also to other modules, we define them as *public*. The transformation procedure needed to translate SysML diagrams to SystemC code is shown in Figure 6. After the designer has described the SoC by means of SysML diagrams, he must generate an XMI file through the SysML tool. The XMI file is a textual repository of the complete SoC description and must be transformed into SystemC code. To perform this transformation we make use of XSLT (eXtensible Stylesheet Language Transformations) [xsl] rules which give the guidelines to be able to generate SystemC code template which build the structure of our SoC.

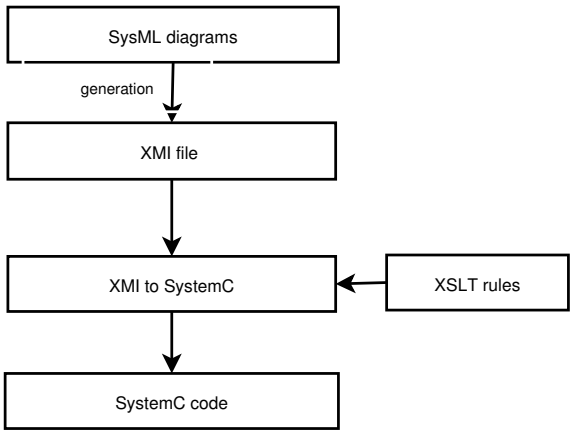

Figure 6: SysML to SystemC transformation procedure

5 Case Study

This section discusses a case study with the aim to present the general rules described in this paper and apply them to a *SensorNode* within a Wireless Sensor Network (WSN) system. Figure 7 describes a BDD representing the structure of a Sensor Node under design. As we can see our system is composed of *CPU*, *Sensor*, *Actuator*, *RF* transceiver, and *Memory*.

Figure 8 shows the IBD of the Sensor Node describing its internal structure. The behavior of the Sensor Node is described in Figure 9. When switched on, the Sensor measures the external temperature and send the value to the CPU to be elaborated. If the threshold described in Figure 10 is overflown, the CPU tells the Actuator to perform the appropriated action, otherwise it stores the data in the Memory. If the Memory is full, the CPU send the data to another sensor through its RF transceiver and then goes to the Idle state waiting for the next measurement to be performed. Figure 9 describes the Activity Diagram of the Sensor Node with the corresponding allocations between activities and the parts responsible for their executions.

Given the above SysML diagrams the following SystemC code will be generated:

```
SC_MODULE(CPU) {
 public
    sc_in    <float> measuredTemp: Temperature;
    sc_inout <float> data: Info;
    sc_inout <float> dataStored: Info;
    sc_out   <bool>  actionDecided: Impulse;

    SC_CTOR(CPU)
        {
         SC_THREAD(elaborate);
         SC_THREAD(store);
```

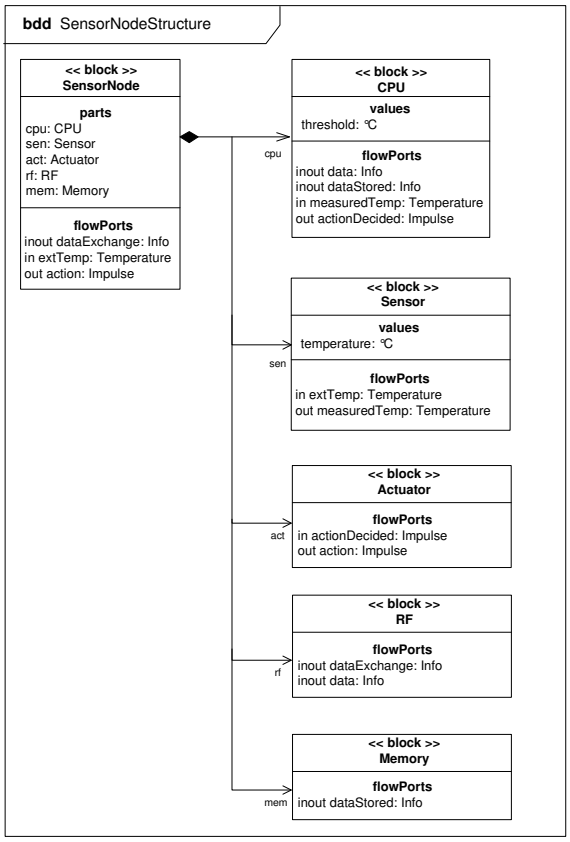

Figure 7: Block Definition Diagram representing the Sensor Node structure

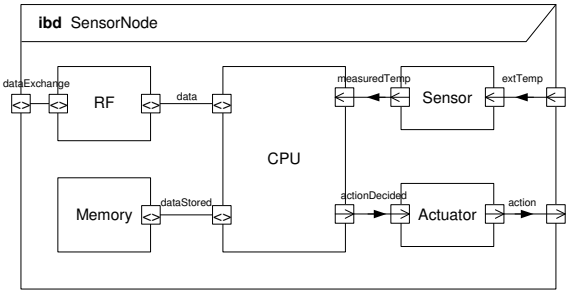

Figure 8: Internal Block Diagram of the Sensor Node

```
        }

    void elaborate()
    void store()
};

SC_MODULE(Sensor) {
 public
    sc_in  <float> extTemp: Temperature;
    sc_out <float> measuredTemp: Temperature;

    SC_CTOR(Sensor)
        {
         SC_THREAD(measure);
        }

    void measure()
```

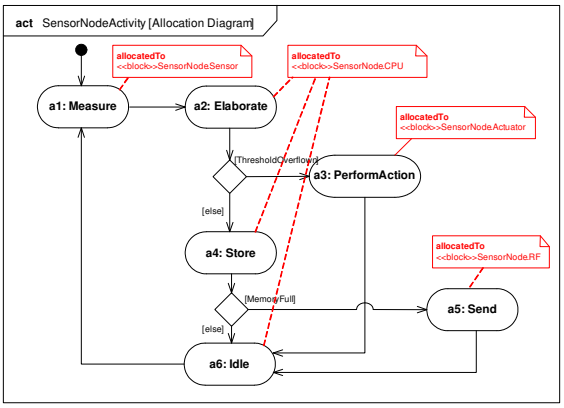

Figure 9: Activity Diagram with allocations

```
};

SC_MODULE(Actuator) {
 public
    sc_in  <bool> actionDecided: Impulse;
    sc_out <bool> action: Impulse;

    SC_CTOR(Actuator)
        {
         SC_THREAD(performAction);
        }

    void performAction()
};

SC_MODULE(RF) {
 public
    sc_inout <float> dataExchange: Info;
    sc_inout <float> data: Info;

    SC_CTOR(RF)
        {
         SC_THREAD(send);
        }

    void send()
};
```

6 Conclusions

In this paper we have presented a SysML profile for modeling Systems on a Chip oriented to SystemC transformation. We have shown that by means of SysML diagrams like BDD, IBD, and Activity allocations it is possible to describe a SoC and then map the SoC descriptions to a SystemC code template which contains module definitions, port- and process declarations. We also described a possible SysML to SystemC transformation procedure based on XMI and XSLT rules that we would like to automate within our future work. The SysML-SystemC mapping has been also evaluated by means of a case study in the field of WSN and a possible SystemC code has been derived from SysML diagrams describing a Sensor Node.

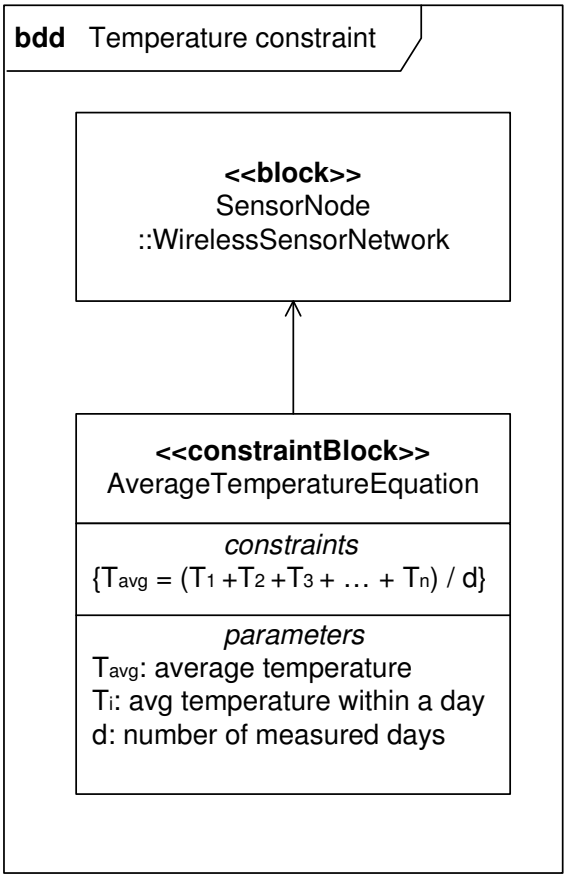

Figure 10: BDD describing the system constraints

This work would like to give a first contribution towards a research topic that has not been investigated so far, namely SysML to SystemC transformation. In fact there is a lot of research works available in the field of UML to SystemC, but nothing within SysML to SystemC. Since we strongly believe that SysML is a very suitable modeling language for SoC design, we think that the transformation from SysML to SystemC is a very important step within an early system design phase.

References

[Ap2] *http://ap233.eurostep.com.*

[Bal06] L. Balmelli. *An Overview of the Systems Modeling Language for Products and Systems Development.* in Journal of Object Technology (JOT) at www.jot.fm., 2006.

[BS02] F. Bruschi and D. Sciuto. *A SystemC based Design Flow starting from UML Model.* In Proc. of ESCUG '02, 2002.

[CSL+03] R. Chen, M. Sgroi, L. Lavagno, A.S. Vincentelli, and J. Rabaey. *UML And Platform Based Design.* In UML for Real Design of Real-Time System, Kluwer Academic Publisher, 2003.

[Gro06] Object Management Group. *UML Profile for System on a Chip (SoC).* OMG 2006, 2006.

[GT03] S. Gerard and F. Terrier. *UML for Real-Time In UML for Real design of Real-Time System.* Kluwer Academic Publisher, 2003.

[HM06] M. Hause and A. Moore. *The Systems Modeling Language - A paper presentation from ARTiSAN Software Tools.* ARTiSAN Software Tools, 2006.

[Inc] *http://www.incose.org.*

[mat] *http://www.w3.org/Math/.*

[MRB+06] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, and Y. Vanderperren. *UML for ESL Design - Basic Principles, Tools, and Applications.* In Proc. of ICCAD'06, 2006.

[Omg] *http://www.omg.org.*

[OMG03a] OMG. *UML for Systems Engineering.* Request For Proposal, 2003.

[OMG03b] www.omg.org OMG. *UML 2.0 OCL Specification.* OCL 2.0, OMG Final Adopted Specification, 2003.

[OMG05] www.omg.org OMG. *MOF 2.0/XMI Mapping Specification, v2.1.* Final Adopted Specification, 2005.

[OMG06] www.omgsysml.org OMG. *OMG Systems Modeling Language (OMG SysML) Specification.* Final Adopted Specification, 2006.

[RSRB05] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. *An HW/SW Co-design Environment based on UML and SystemC.* In Proc. of Forum on specification and Design Languages, FDL'05, 2005.

[SDS00] V. Sinha, F. Doucet, and C. Siska. *YAML: A Tool for Hardware Design Visualization and Capture.* International Symposium on System Synthesis, pp.123-126, 2000, 2000.

[Sysa] *http://www.sysml.org.*

[Sysb] *http://www.systemc.org.*

[Uml] *http://www.uml.org.*

[VSvO02] Y. Vanderperren, G. Sonck, and P. van Oostende. *A design methodology for the development of a complex SoC using UML and executable system models.* In Proc. of Forum on specification and Design Languages, FDL'02, pp 64-69, 2002.

[XJZY05] C. Xi, L. JianHua, Z. ZuCheng, and S. Yao-Hui. *Modeling SystemC Design in UML and Automatic Code Generation.* In Proc. of the conference on Asia South Pacific design automation, 2005, 2005.

[xsl] *http://www.w3.org/TR/xslt.*