Executable Models and Verification from MARTE and SysML: a Comparative Study of Code Generation Capabilities

Marcello Mura Amrit Panda Mauro Prevostini
University of Lugano
ALaRI - Faculty of Informatics
Via Buffi 13, 6904 Lugano, Switzerland
muram@alari.ch, amrit.panda@lu.unisi.ch, mauro.prevostini@unisi.ch

Abstract

In this paper two well known UML profiles, namely SysML and MARTE are closely examined and compared. Both profiles are well suited for the description of embedded systems, although focusing on different aspects and can therefore be considered as complementary. While SysML targets system engineering descriptions in a high level of abstraction and provide diagrams for requirements specification, MARTE is tailored for systems in which Real Time constraints play a major role. Expressiveness of such profiles and their matching with languages that represent the next step in the development of Hardware/Software systems will be the main subject of this work. A Wireless Sensor Network scenario is taken as a reference case study and used to illustrate a practical application of MDA.

1 Introduction

Complexity of electronic systems is constantly increasing and this requires new powerful design strategies. As a result, automatization of the design process, starting from lightweight representations (e.g. UML [11]), is a well estabilished trend nowadays. This process - that is still in progress - produced as major results the definition of profiling in modeling languages [7] and the formalization of Model Driven Architectures [6]. UML2 has been created with the intention of adding new capabilities for large scale systems and achieving great semantic accuracy and concepts consolidation. With UML2 it is possible to develop very sophisticated models and profiles for nearly any type of system (hardware, software, middleware).
In this paper we analyze two of such UML profiles, SysML [9] and MARTE [5]. SysML targets system engineering descriptions in a high level of abstraction and provides diagrams for requirements specification. MARTE on the other hand was developed to model Real Time Systems and is thus mainly targets timing performances (e.g. WCET etc). In particular we show how they can be employed for the automatic generation of executable code. We mainly target generation of SystemC models, but it is possible to think about other possibilities as well. In particular could represent a main topic in the future the mapping into Promela so to be able both to model systems and to extract properties to be verified by *SPIN* model-checker.
Briefly, the paper is organized as follows. Related works are illustrated in Section 2. A summary of SysML and MARTE profiles is provided in Section 3 and Section 4. In Section 5 we describe the process of mapping system description to SystemC and its code generation. A simple case study concerning the modeling of a Wireless Sensor Network is illustrated in Section 6. In Section 7 conclusions are drawn and future work is briefly outlined.

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, [23] started to extend UML by a profile for SystemC that allowed to express a SystemC model in UML. In there a consistent research effort on generation of SystemC code starting from UML diagrams has been done (see e.g. [25], [21] and [22]). Also well known commercial software products target similar issues. I-Logix StateMate [3] generates executable models starting from StateCharts, MATLAB Stateflow [4] does the same starting from a similar concurrent FSM formalism. In [14] the translation of StateCharts into Hierarchical Finite State Machines (HFSMs) is explored in order to build test cases for the corresponding VHDL realization. StateCharts formalism is also very appropriate for the formal validation of models. In particular, automatic translation into *Promela/SPIN*, a language used for automatic Model Checking, was presented in [19], [20] and [16]; recently an interesting approach to this problem was reported in [18]. 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 [13] a specialization of UML is presented to express embedded real-time applications in an abstract way. [12] 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 [15], 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 generations.

3 SysML

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. 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*. 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 [17] in order to show by which kind of parts, including their features, a block is composed.

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 1 shows how to represent the internal structure of a block called $SoC1$ by means of an IBD. The IBD corresponds to a block decomposition 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 $SoC1$.

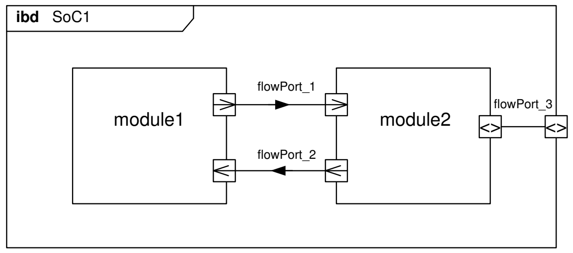

Figure 1. Internal Block Diagram representing the usage of parts within block SoC1

3.3 SysML ports

Two types of ports are available in SysML:

- *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.

4 MARTE

MARTE profile [5] is an evolution of the Schedulability, Performance and Time (SPT) profile with the aim to upgrade this profile to UML2. It is made of various packages: namely MARTE foundations, MARTE design model, MARTE analysis model and MARTE annexes. The profile is intended to be a fundamental tool in the design of real

time systems. Both modeling and analyzing concerns are tackled leading to a complete instrument to improve design phase.
In this section we focus in particular on the Time modeling which is part of MARTE foundations package. It allows the designer to define time constraints and bind his system to them. We focus on the granularity as well as abstraction provided in the temporal domain by this profile. As far as generation of code is involved, this piece of the profile is particularly useful and is fully complementary to SysML profile as the timing aspect is not dealt with in this latter profile.
Time constraints, to which any system must stick to, represent a fundamental analysis aspects and are necessary in order to determine performance of any kind of system. Specially in the case of real time systems, time is very crucial and the system must abide by the hard as well as soft time constraints. In MARTE there are two kinds of time: the logical time and the chronometric time. The chronometric time concerns with the time cardinality alone, whereas the logical time concerns the ordering and organization of events which can again be synchronous or asynchronous. The specialization of the logical time model for synchronous events is independently known as the synchronous time model. For extreme real time applications the simultaneity of events occurrence is taken care by the synchronous time model.
The MARTE profile allows $<< Clocks >>$ which are instances of $<< ClockType >>$. The $<< modelLibrary >>$ TimeTypesLibrary of MARTE has the enumerations $<< enumeration >>$ TimeNatureKind and $<< enumeration >>$ TimeInterpretationKind. TimeNatureKind specifies whether the time is discrete or dense. In the first case time is seen as in a High Level Hardware Description Language in which clock ticks are discrete and time does not elapse during delta cycles. The second case matches the physical time that is known to be continuous. TimeInterpretationKind specifies whether time should be interpreted as instant or as duration. Typically the instant and duration time interpretation kind refers to the time related events that occur in a particular instant called instant time events or occur over some period of time called duration time events. The $<< stereotype >>$ ClockType gives the attributes that any clock can take. The units of time that are supported by MARTE $<< modelLibrary >>$ TimeLibrary are defined based on a base time and a conversion factor. The specifications of the clock objects can be obtained from $<< ClockConstraint >>$ and the scheduling of the clocked events from the $<< TimedConstraints >>$. The $<< timedDomain >>$ encapsulates a certain number of events with their clocks and triggers.

5 Mapping to SystemC

Automatic code generation is a fundamental objective when dealing with design of complex systems. On one hand it speeds up development time, on the other extensive testing is possible from the early development stages therefore improving quality and reliability of final products. Choice of the starting and target formalisms are the main issues in such process. In fact they have a major impact on expressiveness and performance of the generated code that are the major concerns in such an effort. UML represents a convincing starting point as it is the most widely used platform independent modeling language. It contains possibility of conveying both structural and behavioral information, moreover its expressiveness can be enriched by profiling. In particular subset of the language or of its profiles can contain useful information for the generation phase as will be shown later. SystemC [10] represents a convenient choice as target formalism. It can be used to bridge high level specifications (normally expressed in languages as C or through scripts in an environment like Matlab) with actual implementation of Hardware systems (normally described in languages such as Verilog or VHDL). SystemC is a class of libraries built on top of C++ that add to such language a simulation framework, a conceptual representation of time and Hardware oriented datatypes. For this reason it represents a valid option for prototyping and early design phases but can also be employed for later phases up to syntesis. There exist commercial tools for generating Hardware starting from SystemC representations (e.g. Cynthesizer [2]) and translation of SystemC into synthesizable VHDL or Verilog is not known to be an untractable problem.

5.1 Code Generation

The process of code generation happens in two steps as in a Software Compiler. Starting from information exported in textual format from the UML tools (i.e. xmi or di2) an intermediate representation is built. Then code is built starting from such a representation. In this way it is possible to extend the reuse of both parts. Changing the UML tool (and as a consequence the source XMI dialect) reflects only in applying changes to the frontend reusing the existing backend. In case different target formalisms are used the frontend part can be reused and only the backend needs to be modified.

5.2 Translation Template

In order to define how high level UML specifications are translated into SystemC code it is necessary to define a template code. Ideally there is a one to one mapping between UML (also involving stereotypes coming from profiles) and transformations can be performed in both directions using such mapping. In practice realization of such mapping is not straightforward. While it is possible to profile UML in a way that all the SystemC building blocks can be represented (see e.g. [24]) it is not straightforward to create a consistent UML model starting from representation of a given SystemC code. Limits on the style of coding must be imposed so that behavioral information can be conveyed into appropriate state diagrams.

5.2.1 SysML models

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. The header files of $module_1$ and $module_2$ of $SoC1$ described in Figure 1 can be translated into SystemC code as follows:

```
SC_MODULE (module_1) {

 public:
    sc_out  <int> flowPort_1: type;
    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();
};
```

Secondly we can build the implementation files (.cpp) that describes the module behavior and could be implemented starting from StateCharts diagrams as described in our previous [21].

5.2.2 MARTE models

The main limitation we encountered with SysML is the lack of constructs for modelling time. Thus we were pushed to examining different packages and model libraries in the MARTE time profile and try to find a mapping into SystemC. This way increasing the expressiveness of the modelling formalism by the concurrent use of multiple profiles we improve the automatic generation phase.
Discrete time can be mapped in SystemC to any digital clock object with a defined period and duty cycle and also the unit of time but it fails to map the dense time to any kind of clock. In fact being SystemC a high level programming language the representation of dense time in non-discrete domain is not feasible.
The chronometric time model can be considered as directly mapped to SystemC simulation time that expresses the time cardinality. Logical time can be represented by means of the sensitivity list of any process of the system. Since the idea behind the notion of logical time is the ordering of events in a sequence, we take the advantage of the execution of processes on triggering of certain events. For example if we want to order the execution of events A, B and C in the order C follows B follows A. And the only criterion for their execution is following the preceding event.
These events are mapped to processes in SystemC namely *proc_a*, *proc_b*, and *proc_c*. We make these processes sensitive to only a single signal say *trig_a*, *trig_b* and *trig_c* respectively. The state of *trig_b* is changed only at the end of *proc_a*. Process A may include the modifications of one or many signals but we need to make sure that the last change that the process makes is that of *trig_b* which triggers the execution of *proc_b*. Similarly *proc_b* should also change state of *trig_c* at its end to trigger *proc_c*. This assures the ordered execution of the processes. We trigger the first process (A) in the main control unit by changing the state of *trig_a*. There may be a case when the criteria for execution of a process may not be following the preceding event alone. In such case the additional signals to which the process is sensitive are moved from the sensitivity list of each process and appended to the sensitivity list of the first process (in this case *proc_a*).
The SystemC *wait (duration, unit)* function can be used to regulate the time constraint equations. For example if a specific event is expected to occur within a particular duration of time after a preceding event , we can wait for that amount of time and then poll for the completion of the event. If the poll fails then the constraint is no longer valid and system

gives up in performance. The *wait ()* should be invoked at the end of the preceding process (in the control unit of the system) and flow should be directed to polling the next event. Similar visualization in SystemC is achieved by the clock object which is an object of the class *sc_clock*.
Mapping *sc_time_unit* (*SC_FS*; *SC_PS*; *SC_NS*; *SC_US*; *SC_MS*; *SC_SEC*) to the *TimeLibrary*, we can design any specific clock with any time unit. As long as the TimeInterpretationKind is involved the mapping s as follows:

- Duration is mapped to a scheduler,started with *sc_start (double d, sc_time_unit t)* and executing a sequence of operations which cannot be accomplished with a single instruction in an instant. It takes more than one clock cycle to complete the job.
- Instant is mapped to the processes that are involved in signal modifications or initializing a scheduler as this is done only with one instruction in an instant. It takes only one clock cycle to complete job.

6 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 2 shows the IBD of a Sensor Node describing its internal structure. As we can see our system is composed of CPU, $Sensor$, $Actuator$, RF transceiver, and $Memory$. When switched on, the Sensor measures the external temperature and sends the value to the CPU to be elaborated. If a predefined threshold 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 sends the data to another sensor through its RF transceiver and then goes to the Idle state waiting for the next measurement to be performed. Given SysML diagram

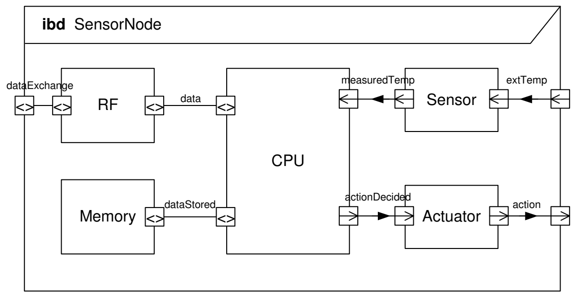

Figure 2. Internal Block Diagram of the Sensor Node

in Figure 2 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);
        }
    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()
};
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()
};
```

As it is easy to notice no information on timings (i.e. clocks of the system) is available. For this reason we used the previously described mapping of the MARTE time profile. As a simplification we use a single clock for the whole system that defines the timescale of the simulation. In practice it is possible to have multiple clocks in the various blocks.

The classes sketched in Figure 3 allow defining a clock for

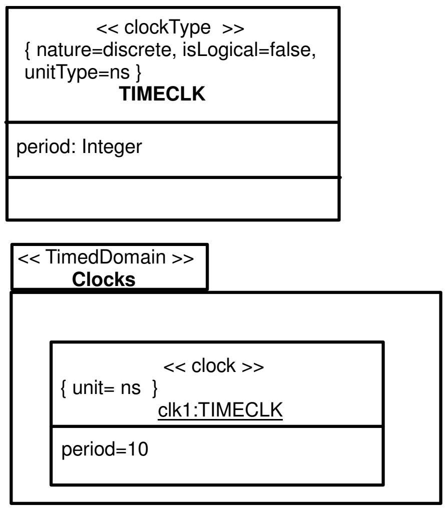

Figure 3. MARTE representation of a 100MHz Clock

the whole system. In particular the information contained in such figure can be translated in SystemC in the statement *sc_clock(clk1,10,SC_NS,0.5)*. It is straightforward to insert in the classes more information (e.g. duty cycle) and reuse them for generation of code.

7 Conclusions

In this paper we investigated how enrichment coming from UML profiles can be used in order to enhance code generation techniques. We closely analyzed both SysML and MARTE profiles and found that even though the former looks better suited for generating SystemC code, interesting possibilities can be enabled also by the latter. In particular we noticed that contemporary use of stereotypes from both profiles may represent an interesting option. We found that tool support for such operation is still to be improved. Some tools (i.e. Rhapsody [1]) do not support MARTE profile. On the other hand Papyrus [8], supporting both profiles through plugins, has some major limitations in the available features (i.e. missing diagrams, limited xmi translation). Future work will involve refinement of obtained results and application of similar techniques for validation of systems. Analysis of existing UML tools and possibly extensions are foreseen.

Acknowledgements

The authors would like to thank Luis Gabriel Murillo for his precious help.

References

[1] *http://modeling.telelogic.com/products/rhapsody/index.cfm*.
[2] *http://www.ForteDS.com*.
[3] http://www.ilogix.com/sublevel.aspx?id=74.
[4] http://www.mathworks.com/products/stateflow/.
[5] *http://www.omgmarte.org/*.
[6] http://www.omg.org/mda/.
[7] http://www.omg.org/technology/documents/profile_catalog.htm.
[8] *http://www.papyrusuml.org*.
[9] *http://www.sysml.org*.
[10] *http://www.systemc.org*.
[11] UML: The unified modeling language - standard, version 2.0, Aug. 2005.
[12] F. Bruschi and D. Sciuto. *A SystemC based Design Flow starting from UML Model*. In Proc. of ESCUG '02, 2002.
[13] R. Chen, M. Sgroi, L. Lavagno, A. Vincentelli, and J. Rabaey. *UML And Platform Based Design*. In UML for Real Design of Real-Time System, Kluwer Academic Publisher, 2003.
[14] F. Fummi, M. Sami, and F. Tartarini. Use of Statecharts-Related description to achieve testable design of control subsystems. In *Proc. GLSVLSI*, 1997.
[15] S. Gerard and F. Terrier. *UML for Real-Time In UML for Real design of Real-Time System*. Kluwer Academic Publisher, 2003.
[16] S. Gnesi, D. Latella, and M. Massink. Modular semantics for a UML statechart diagrams kernel and its extension to multicharts and branching time model-checking. *Journal of Formal Aspects of Computing*, 51, 2002.
[17] M. Hause and A. Moore. *The Systems Modeling Language - A paper presentation from ARTiSAN Software Tools*. ARTiSAN Software Tools, 2006.
[18] D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the spin model-checker. *Journal of Logic and Algebraic Programming*, 11, 1999.
[19] J. Lilius and I. P. Paltor. vUML: A tool for verifying UML models. *ase*.
[20] E. Mikk, Y. Lakhnech, M. Siegel, and G. J. Holzmann. Implementing statecharts in promela/spin. In *Proc. WIFT*, 1998.
[21] M. Mura and M. Paolieri. Sc2: State charts to system c: Automatic executable models generation. In *proceedings FDL07*, Barcelona, Spain.
[22] K. D. Nguyen, Z. Sun, P. Thiagarajan, and W. Wong. Model-driven SoC design via executable UML to SystemC. In *Proc. of RTSS*.
[23] 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.
[24] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A soc design methodology involving a uml 2.0 profile for systemc. In *Proc. DATE*, 2005.
[25] C. Xi, L. JianHua, Z. ZuCheng, and S. YaoHui. Modeling SystemC design in UML and automatic code generation. In *Proceedings of ASP-DAC*, 2005.