Scenario-based Validation of Embedded Systems*

A. Gargantini
DIIMM, Università di Bergamo, Italy
angelo.gargantini@unibg.it

E. Riccobene, P. Scandurra, A. Carioni
DTI, Università di Milano, Italy
{riccobene, scandurra, carioni}@dti.unimi.it

Abstract

This paper describes a scenario-based methodology for system-level design validation based on the Abstract State Machines formal method. This scenario-based approach complements an existing model-driven design methodology for embedded systems based on the SystemC UML profile. It allows the designer to functionally validate system components from SystemC UML designs early at high levels of abstraction and without requiring strong skills and expertise on formal methods. A validation tool integrated into an existing model-driven co-design environment to support the proposed scenario-based validation flow is also presented.

1 Introduction

SystemC (built upon C++) has emerged as de facto, open [20], industry-standard language for *system-level* models, specifically targeted at architectural, algorithmic, transaction-level modelling of embedded systems. [27]. Recently, a further improvement has been achieved by exploiting lightweight software modelling languages like UML (Unified Modeling Language) [28] to describe system specifications and generate from them executable models in C/C++/SystemC. See the numerous standardization activities controlled by the OMG group [19] like the Schedulability, Performance, and Timing Analysis (SPT) profile, the recent UML extension for SoC (USoC), the SysML proposal, which extends UML towards the Systems Engineering domain, and the MARTE (Modeling and Analysis of Real-Time Embedded systems) initiative. Along the same research line, according to the Model Driven Engineering (MDE) approach, in [25] a model-driven design methodology for embedded systems is introduced. It is based on the UML 2, a SystemC UML profile (for the HW side), and a multi-thread C UML profile (for the SW side). It allows system modelling at higher levels of abstraction (from a functional executable level down to RTL level). The methodology is fostered by a design process called UPES (Unified Process for Embedded Systems) [26] which extends the conventional Unified Process (UP), and by the UpSoC (Unified Process for SoC) subprocess of UPES for refining the HW platform model.

However, UML-like design methods are not yet well supported by effective *formal analysis* (validation & verification) techniques. Formal methods and analysis tools have been most often applied to low level hardware design. But, these techniques are not applicable to system descriptions given in terms of programs of system-level languages like SystemC, since such languages are closer to concurrent software than to traditional hardware description [29], and the focus in the literature so far has been mainly on traditional code-based simulation techniques. Moreover, classical analysis techniques are not directly applicable to UML-based design methods, due to their lack of a strong mathematical foundation necessary for formal model analysis.

We address the problem of formally analyzing high-level UML-like embedded system descriptions by combining model-driven modelling languages with formal notations (like ASMs, Object-Z, Petri Nets, etc.) capable of eliminating ambiguities in the UML semantics. We here show how to complement the design methodology in [25] with a *formal process* for high level system validation involving the Abstract State Machines (ASMs) formal method [6] and the ASMETA (ASM mETAmodelling) toolset [4] as supporting analysis tools around ASMs. The choice of the ASMs as formal method is intentional and due to the fact that this method comes with a rigourous scientific foundation [6], and since it provides executable specifications, it is suitable for high-level model validation. Validation is intended as the process of investigating a model with respect to its user perceptions, in order to ensure that the specification really reflects the user needs and statements about the application, and to detect faults in the specification as early as possible with limited effort. Validation should precede the application of more expensive and accurate methods, like formal verification of properties, that should be applied only when a designer has enough confidence that requirements satis-

*This work is supported in part by the project *Model-driven methodologies and techniques for embedded system design through UML, ASMs and SystemC* at STMicroelectronics.

faction is guaranteed. Moreover, the validation activity should be supported by tools allowing the user to effectively interact with the specification/prototype.

In this paper, we propose a *scenario-based* approach to system-level design validation which allows the designer to build critical scenarios reflecting given requirements to be guaranteed and check for requirements satisfaction. System components from SystemC UML designs can be, therefore, functionally validated early at high levels of abstraction, and even in a transparent way, i.e. no strong skills and expertise on formal methods are required to the user. We also present the validation tool integrated into the model-driven HW-SW co-design environment originally presented in [24] to support the scenario-based validation flow. As a proof-of-concept, the paper reports the results of the scenario-based validation for the well-known Simple Bus case study from the SystemC distribution.

This paper is organized as follows. Sect. 2 provides some background on the ASMs and their supporting toolset. Sect. 3 briefly reminds the model-driven HW-SW co-design environment architecture of [24] and its components features. Sect. 4 focus on the new validation component of this environment by describing the mapping from the SystemC UML models to ASM models and the scenario-based approach for high-level validation of SystemC UML models. Sect. 5 provides some results of the scenario-based validation of the Simple Bus case study. Sect. 6 quotes some relevant related work. Finally, Sect. 7 concludes the paper.

2 ASMs and ASMETA

Abstract State Machines are an extension of FSMs, where unstructured control states are replaced by states with arbitrary complex data. The *states* of an ASM are multi-sorted first-order structures, i.e. domains of objects with functions and predicates defined on them, while the *transition relation* is specified by "rules" describing the modification of the functions from one state to the next. A complete mathematical definition of the ASM method can be found in [6]. The notion of ASMs moves from a definition which formalizes simultaneous parallel actions of a single agent, either in an atomic way, *Basic ASMs*, and in a structured and recursive way, *Structured or Turbo ASMs*, to a generalization where multiple agents interact *Multi-agent ASMs*. Appropriate rule constructors also allow non-determinism and unrestricted synchronous parallelism.

The *ASMETA* (ASM mETAmodelling) toolset [9, 4] is a set of tools around ASMs developed according to the model-driven development principles. At the core of the toolset, the *AsmM metamodel* [4] is a complete meta-level representation of ASMs concepts based on the OMG's Meta-Object-Facility (MOF) [16]. AsmM is also available in the meta-language EMF/Ecore [3] thanks to the ATL-KM3 plug-in [2], which allows model transformations both in the EMF and MOF.

The ASMETA toolset includes: a notation, *AsmetaL*, to write ASM models conforming to the AsmM in a textual and human-comprehensible form; a text-to-model compiler, *AsmetaLc*, to parse AsmetaL models and check for their consistency w.r.t. the AsmM OCL constraints; a simulator, *AsmetaS*, to execute ASM models; the *Avalla* language, a domain-specific modelling language for scenario-based validation of ASM models, with its supporting tool, the *AsmetaV* validator; and the *ATGT* tool that is an ASM-based test case generator based upon the SPIN model checker [14].

3 Model-driven co-design environment

The overall co-design environment presented in [24] works as front-end for consolidated lower level co-design tools, and is intended to assist the designer across the refinement steps in the modelling activity, from a high-level functional UML model of the system down to the SystemC RTL. Fig. 1 shows the environment architecture. The environment consists of two major parts: a development kit (DK) with design and development components, and a runtime environment (RE) that is the SystemC execution engine.

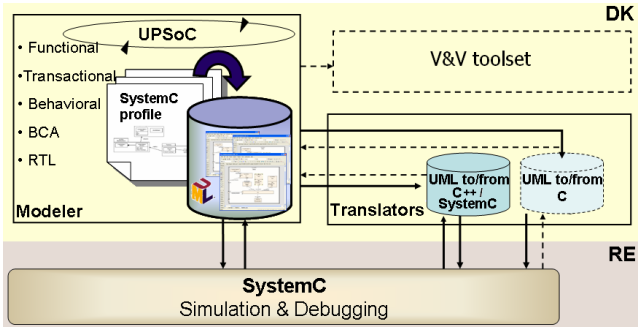

Figure 1. Tool architecture

The DK consists of a UML2 *modeler* supporting the UML profile for SystemC and for multi-thread C, *translators* for forward/reverse engineering to/from C/C++/SystemC. The modeler is based on the Enterprise Architect (EA) UML tool [8] by SparxSystems.

4 The Validation component

Fig. 1 exposes (inside a dashed rectangular) a V&V tool as new component of the co-design environment. It is built upon the ASMETA toolset and should guarantee traceability and correctness along the UPSoC design process from a UML high-level abstract description of the system to the final SystemC implementation. The validation component architecture is depicted in Fig. 2 together with the phases (denoted in the figure with

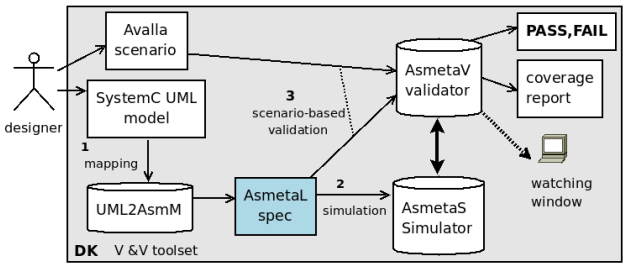

Figure 2. V&V toolset

a number and a label) the designer (or analyst) undertakes in the scenario-based validation process. The process starts by applying the mapping (**phase 1**) of the SystemC-UML model of the system (exported from the EA-based modeler in Fig. 1) into a corresponding ASM model (written in AsmetaL). This transformation is defined (once for all) by establishing a set of semantic mapping rules between the SystemC UML profile and the AsmM metamodel (see below). The `UML2AsmM` transformation is completely automatized by means of the ATL transformation engine [2] developed as a possible implementation of the OMG QVT [22] standard.

Once the ASM model of the system is generated, the designer can do basic simulation (**phase 2**) and scenario-based validation (**phase 3**). A brief description of each activity follows. Note that as required skills and expertise the designer has to familiarise with the SystemC UML profile (embedded in the EA-based modeler), and with very few commands of the Avalla textual notation to write pertinent validation scenarios.

Mapping from SystemC UML profile to AsmM In this step the SystemC UML model provided in input from the EA tool is transformed into a corresponding ASM model (an instance of the AsmM metamodel). In order to provide a one-to-one mapping (for both the structural and behavioural aspects), first we had to express in terms of ASMs the SystemC discrete (absolute and integer-valued) and event-based simulation semantics. To this goal, we took inspiration from the ASM formalization of the SystemC 2.0 simulation semantics in [18] to define a precise and executable semantics of the SystemC UML profile and, in particular, of the SystemC scheduler and the *SystemC process state machines* (an extension of the UML statecharts for modelling the behaviour of the reactive SystemC processes). We then proceeded to model in ASMs the predefined set of interfaces, ports and primitive channels (the SystemC layer 1), and SystemC-specific data types. The resulting SystemC-ASM component library is available as target of the `UML2AsmM` transformation process.

Exploiting the SystemC-ASM component library, a SystemC module M is mapped into an ASM containing in its signature a dynamic abstract domain M. This domain is the set of instances that can be created by the corresponding module. Module attributes and ports of type T are mapped into controlled ASM functions declared in the signature of the ASM corresponding to the module. Basically, these functions have M as domain, and T as codomain. Multiplicity and properties (like *ordered*, *unique*, etc..) of attributes and ports are captured by the codomain types of the corresponding functions. A multi-port of type T, for example, is mapped into a controlled ASM function with codomain $\mathcal{P}(\mathcal{T})$, i.e. the mathematical powerset of T. A hierarchical channel is treated as a module. A primitive channel is mapped, instead, into a concrete sub-domain of the predefined abstract domain *PrimChannel*, which is part of the SystemC-ASM component library provided as foundational semantic basis. An event is mapped into an element of a predefined abstract domain *Event*.

For the behavioural part, a process (a sc_thread or a sc_method) is mapped into an element of a predefined abstract domain *Process*. A process behaviour within a module is defined by a named, possibly parameterized, transition rule declared within the ASM corresponding to the container module. Moreover, since in the SystemC process state machines, control structures (like `if-then-else`, `while` loop, etc.) and process synchronization points (statements like `wait`, `static_wait`, `dont_initialize`, etc.) are modelled in terms of stereotyped pseudo-states (junction or choice) and states, respectively, a one-to-one mapping is defined between the state-like diagram of the process behaviour and the basic ASM rule constructs (if-then-else rule, seq rule, etc.). Some special ASM rule constructs, however, have been introduced in the SystemC-ASM component library in order to capture in ASMs the semantics underlying all possible forms of synchronization calls (which require dealing with the ASM agent representing the SystemC scheduler). In particular, the infinite loop mechanism of a thread has been modelled with a specific design pattern of ASM rule constructors.

As example of application of such mapping, Fig. 3 shows the UML notation, the SystemC code, and the resulting ASM (in AsmetaL) for a module.

Basic simulation The AsmetaS simulator interprets ASM models (as instances of the AsmM metamodel). It can be used in a standalone way to provide basic simulation of the overall system behaviour. As key features for model validation, AsmetaS supports *axiom checking* (to check whether axioms expressed over the currently executed ASM model are satisfied or not), *consistent updates checking* for revealing inconsistent updates, *random simulation*, and configurable *logging* facilities to inspect the machine state. Axiom checking and random simulation allow the user to perform a

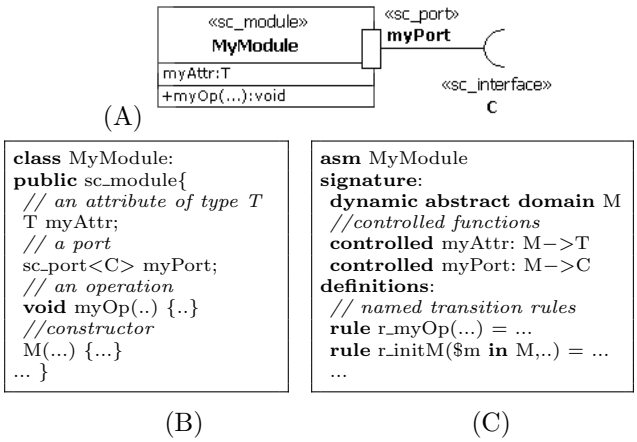

Figure 3. A UML module (A), its SystemC code (B) and its corresponding ASM (C)

draft system validation with minimal effort.

Scenario-based functional validation The AsmetaV validator is based on the AsmetaS simulator and on the Avalla modelling language. This last provides constructs to express execution scenarios in an algorithmic way as interaction sequences consisting of *actions* committed by the *user actor* to `set` the environment (i.e. the values of monitored/shared functions), to `check` the machine state, to ask for the `execution` of certain transition rules, and to enforce the machine itself to make one `step` (or a sequence of steps by `step until`) as reaction of the actor actions. AsmetaV reads a user scenario written in Avalla (see Fig. 2), it builds the scenario as instance of the Avalla metamodel by means of a parser, it transforms the scenario and the AsmetaL specification which the scenario refers to, to an executable AsmM model. Then, AsmetaV invokes the AsmetaS interpreter to simulate the scenario. During simulation the user can pause the simulation and watch the current state and value of the update set at every step, through a watching window. During simulation, AsmetaV captures any check violation and if none occurs it finishes with a "PASS" verdict. Besides a "PASS"/"FAIL" verdict, during the scenario running AsmetaV collects in a final report some information about the coverage of the original model; this is useful to check which transition rules have been exercised.

5 The Simple Bus case study

The `Simple Bus` case study is a well-known transactional level example, designed to perform also cycle-accurate simulation. It is made of about 1200 lines of code that implement a high performance, abstract bus model. The complete code is available at the official SystemC web site [20].

The Simple Bus system was modelled [23] in a forward engineering flow using the SystemC UML profile. The UML object diagram in Fig. 4 shows the internal collaboration structure of the objects involved in a specific configuration of the Simple Bus design: three master blocks (a blocking master `master_b`, a non-blocking master `master_nb`, and a monitor `master_d`; two slave memories (one fast, `mem_fast` and one slow, `mem_slow`); a `bus` connecting masters and slaves; an `arbiter` with a priority-based arbitration to select a request to serve and with bus-locking support; a clock generator `C1`[1]. Every master submits read/write requests to the bus at regular time instants. The designer assigns a unique priority to each master: `master_nb` has priority 3, while `master_b` has priority 4. Masters can issue a request at the same time, so the arbiter must choose one request according to some deterministic rules. In the simplest case, precedence is accorded to the device with higher priority[2], in our case the non-blocking master has priority 3 which is higher (following a decreasing order) than the priority 4 of the blocking master. When a master occupies the bus, an incoming request is therefore queued and served later in a different time instant, or served from the next clock cycle if it has a higher priority (and the current request will be terminated later).

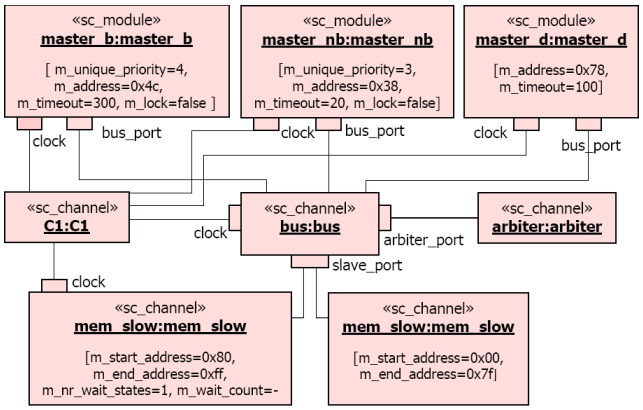

Figure 4. Simple Bus – UML object diagram

To illustrate the typical use of the Avalla language in writing validation scenarios, below we report two scenario examples and their related validation results for the Simple Bus design. The first scenario shows how high level modelling tools like AsmetaV/Avalla are helpful to abstract and stand out monitoring and debugging functionality, typically embedded within the SystemC design (in our case within the `master_d` monitor, the arbiter, and the bus) by inserting C++ code lines, thus further alleviating the designers'burden of

[1]Note that all connectors are intented as stereotyped with «sc_connector».

[2]Two devices can not have the same priority, so the determinism is assured.

writing code. The second scenario shows instead how to validate the fairness of the arbitration rules adopted for scheduling the masters requests.

Scenario s1: At given time instants, the memory locations between the address 120 and the address 132 are read (`directReadBus`). The actual values must match the expected values.

```
scenario s1 load Top.asm
step until time = 0 and phase = TIMED_NOTIFICATION;
check directReadBus(bus, 120) = 0
    and directReadBus(bus, 124) = 0
    and directReadBus(bus, 128) = 0
    and directReadBus(bus, 132) = 0;
step until time = 1600 and phase = TIMED_NOTIFICATION;
check directReadBus(bus, 120) = 16
    and directReadBus(bus, 124) = 0
    and directReadBus(bus, 128) = 0
    and directReadBus(bus, 132) = 0;
```

Scenario s2: At time 0, the `master_nb` (with priority 3) issues a read request(`status = SIMPLE_BUS_REQUEST` and `do_write = false`) at address 56 (`address = 56`), and the `master_b` (with priority 4) issues a burst read request from address 76 to 136. At time 15, the bus must serve the master with higher priority, i.e. the `master_nb`, and complete it (`status = SIMPLE_BUS_OK`). At time 30, the `master_nb` issues a write request at address 56. At time 45, the bus serves again the `master_nb` ignoring for the second time the still pending read request of the `master_b`.

```
scenario s2 load Top.asm
step until time = 0 and phase = TIMED_NOTIFICATION;
check (exist $r00 in Request with priority($r00) = 3
    and do_write($r00) = false
    and address($r00) = 56
    and status($r00) = SIMPLE_BUS_REQUEST);
check (exist $r01 in Request with priority($r01) = 4
    and do_write($r01) = false
    and address($r01) = 76
    and end_address($r01) = 136
    and status($r01) = SIMPLE_BUS_REQUEST);
step until time = 15 and phase = TIMED_NOTIFICATION;
check (exist $r02 in Request with priority($r02) = 3
    and status($r02) = SIMPLE_BUS_OK);
step until time = 30 and phase = TIMED_NOTIFICATION;
check (exist $r03 in Request with priority($r03) = 3
    and do_write($r03) = true
    and address($r03) = 56
    and status($r03) = SIMPLE_BUS_REQUEST);
step until time = 45 and phase = TIMED_NOTIFICATION;
check (exist $r04 in Request with priority($r04) = 3
    and status($r04) = SIMPLE_BUS_OK);
```

Both scenarios ended with verdict PASS and allowed a coverage of all ASM rules of the Simple Bus model.

6 Related work

In [21], the authors present a model-driven development and validation process which begins by creating (from a natural language specification of the system requirements) a functional abstract model and (still manually) a SystemC implementation model. The abstract model is described using the Abstract State Machine Language (AsmL) – another implementation language for ASMs. Our methodology, instead, benefits from the use of the UML as design entry-level and of model translators which provide automation and ensure consistency among descriptions in different notations (such those in SystemC and ASMs). Moreover, these last can remain hidden to the designer, making the process completely transparent to the user who do not want to deal with them. In [21], a designer can visually explore the actions of interest in the ASM model using the Spec Explorer tool and generate tests. These tests are used to drive the SystemC implementation from the ASM model to check whether the implementation model conforms to the abstract model (*conformance testing*). The test generation capability is limited and not scalable. In order to generate tests, the internal algorithm of Spec Explorer extracts a finite state machine from ASM models and then use test generation techniques for FSMs. The effectiveness of their methodology is therefore severely constrained by the limits inherited from the use of Spec Explorer. The authors themselves say that the main difficulty is in using Spec Explorer and its methods for state space pruning/exploration. The ASMETA ATGT tool that we want to use for the same goal exploits, instead, the method of model checking to generate test sequences, and it is based on a direct encoding of ASMs in PROMELA, the language of the model checker SPIN [14].

The work in [12] also uses AsmL and Spec Explorer to settle a development and verification methodology for SystemC. They focus on assertion based verification of SystemC designs using the Property Specification Language (PSL), and although they mention test case generation as a possibility, the validation aspect is largely ignored. We were not able to investigate carefully their work as their tools are unavailable. Moreover, it should be noted that approaches in [21, 12], although using the Spec Explorer tool, do not exploit the scenario-based validation feature of Spec Explorer. Indeed, in [11, 5] was shown how Spec Explorer allows scenario-oriented modelling.

In [15], a model-driven methodology for development and validation of system-level SystemC designs is presented. The development and validation flow is entirely based on the specification of a functional model (reference model) in the ESTEREL language, a state machine formalism, and on the use of the ESTEREL Studio development environment [1] for the purpose of test generation. The proposed approach concentrates on providing coverage-directed test suite generation for system level design validation.

Authors in [7] provide test case generation by performing *static analysis* on SystemC designs. This approach is limited by the strength of the static analy-

sis tools, and the lack of flexibility in describing the reachable states of interest for directed test generation. Moreover, static analysis requires sophisticated syntactic analysis and the construction of a semantic model, which for a language like SystemC (built on C++) is difficult due to the lack of formal semantics.

The SystemC Verification Library [20] provides API for transaction-based verification, constrained and weighted randomization, exception handling, and HDL-connection. We aim, however, at developing formal techniques to augment SystemC verification.

The Message Sequence Chart (MSC) notation [17], originally developed for telecommunication systems, can be adapted to embedded systems to allow validation. For instance, in [10] MSC is adopted to visualize the simulation of SystemC models. The traces are only displayed and not validated, and the author report the difficulties of adopting a graphical notation like MSC. Our approach is similar to that presented in [13], where the MSCs are validated against the SDL model, from which a SystemC implementation is derived. MSCs are also generated by the SDL model and replayed to cross validation and regression testing.

7 Conclusions and future work

We proposed a *scenario-based validation* approach to system-level design by the use of the SystemC UML profile (for the modelling part) and the ASM formal method and its related ASMETA toolset (for the validation part). We have been testing our validation technique on case studies taken from the standard SystemC distribution, like the Simple Bus presented here, and on some of industrial interest. Thanks to the ease in raising the abstraction level using ASMs, we believe our approach scales effectively to industrial systems.

In the future, we want to integrate AsmetaV with the ATGT tool of the ASMETA toolset to be able to automatically generate some scenarios by using ATGT and ask for a certain type of coverage (rule coverage, fault detection, etc.). Test cases generated by ATGT and the validation scenarios can be transformed in concrete SystemC test cases to test the conformance of the implementations with respect to their specification. Moreover, we plan to support system properties formal verification by model checking techniques. This requires transforming ASM models into inputs for model checkers, for example SPIN.

References

[1] Esterel Studio. `www.estereltechnologies.com`.
[2] The ATL language. `www.eclipse.org/m2m/atl/`.
[3] Eclipse Modeling Framework. `www.eclipse.org/emf/`.
[4] The ASMETA toolset. `http://asmeta.sf.net/`, 2006.
[5] M. Barnett et al. Validating use-cases with the AsmL test tool. In *QSIC Int. Conference on Quality Software*, p. 238–246. IEEE, 2003.
[6] E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.
[7] F. Bruschi, F. Ferrandi, and D. Sciuto. A framework for the functional verification of SystemC models. *Int. J. Parallel Program.*, 33(6):667–695, 2005.
[8] The Enterprise Architect: `www.sparxsystems.com.au/`
[9] A. Gargantini, E. Riccobene, and P. Scandurra. A metamodel-based simulator for ASMs. In *Proc. of the 14th Int. ASM Workshop*, 2007.
[10] T. Kogel. et al. Virtual Architecture Mapping: A SystemC based Methodology for Architectural Exploration of System-on-Chip Designs. In A. D. Pimentel and S. Vassiliadis (Eds.), *Computer Systems: Architectures, Modeling, and Simulation. SAMOS*, *LNCS* 3133, p. 138–148, Springer-Verlag, 2004.
[11] W. Grieskamp, N. Tillmann, and M. Veanes. Instrumenting scenarios in a model-driven development environment. *Information & Software Technology*, 46(15):1027–1036, 2004.
[12] A. Habibi and S. Tahar. Design and verification of SystemC transaction-level models. *IEEE Transactions on VLSI Systems*, 14:57–68, 2006.
[13] M. Haroud et al. HW accelerated ultra wide band MAC protocol using SDL and SystemC. In *IEEE Radio and Wireless Conference*, p. 525–528, 2004.
[14] G. J. Holzmann. The Model Checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.
[15] D. Mathaikutty, S. Ahuja, A. Dingankar, and S. Shukla. Model-driven test generation for system level validation. In *HLVDT'07: High Level Design Validation and Test Workshop*, p. 83–90, 2007. IEEE.
[16] OMG. The Meta Object Facility, formal/2002-04-03.
[17] Message Sequence Charts (MSC) ITU-T. Z.120, 1999.
[18] W. Müller, J. Ruf, and W. Rosenstiel. *SystemC Methodologies and Applications*. Kluwer Academic Publishers, 2003.
[19] The Object Managment Group (OMG). `www.omg.org`.
[20] Open SystemC Initiative. `http://www.systemc.org`.
[21] H. D. Patel and S. K. Shukla. Model-driven validation of SystemC designs. In *DAC'07: Proc. of the 44th Design Automation Conference*, p. 29–34, New York, 2007. ACM.
[22] OMG, Query/Views/Transformations, ptc/07-07-07.
[23] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A UML2 Profile for SystemC 2.1. STMicroelectronics Technical Report, April 2007.
[24] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A model-driven design environment for embedded systems. In *DAC'06: Proc. of the 43rd Design Automation Conference*, p. 915–918, New York, 2006. ACM.
[25] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A Model-driven co-design flow for Embedded Systems. *Advances in Design and Specification Languages for Embedded Systems (Best of FDL'06)*, 2007.
[26] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. Designing a unified process for embedded systems. In *Fourth Int. workshop on Model-based Methodologies for Pervasive and Embedded Software*. IEEE Press, 2007.
[27] T. Gröetker and S. Liao and G. Martin and S. Swan. *System Design with SystemC*. Kluwer, 2002.
[28] OMG. The Unified Modeling Language. `www.uml.org`.
[29] M. Y. Vardi. Formal Techniques for SystemC Verification; Position Paper. In *DAC'07: Proc. of the 44rd Design Automation Conference*, p. 188–192. IEEE, 2007.