Chapter 1

MODEL-DRIVEN SYSTEM VALIDATION BY SCENARIOS *

A.CARIONI[1], A.GARGANTINI[2], E.RICCOBENE[1], P.SCANDURRA[1]

Abstract The chapter presents a method for scenario-based validation of embedded system designs provided in terms of UML models. This approach is based on model transformations from SystemC UML graphical models into Abstract State Machine (ASM) formal models, and exploits the scenario-based model validation of the ASMs. This validation approach complements an existing model-driven design methodology for embedded systems based on the SystemC UML profile. A validation tool integrated into an existing model-driven co-design environment to support the proposed scenario-based validation flow is also presented. It allows the designer to functionally validate system components from SystemC UML designs early at high levels of abstraction.

1. Introduction

In the Embedded System (ES) and System-on-Chip (SoC) design area, conventional system level design flows usually start by developing a system functional executable model from a system specification written in natural language. It is an emerging practice to develop the functional model and refine it with SystemC (built upon C++), which is considered as de facto, open [22], industry-standard language for functional *system-*

*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.
[1]DIPARTIMENTO DI TECNOLOGIE DELL'INFORMAZIONE, UNIVERSITÀ DI MILAN, ITALY. {CARIONI, RICCOBENE, SCANDURRA}@DTI.UNIMI.IT
[2]DIPARTIMENTO DI INGEGNERIA INFORMATICA E METODI MATEMATICI, UNIVERSITÀ DI BERGAMO, ITALY. ANGELO.GARGANTINI@UNIBG.IT

level models [29]. The functional executable model, as program code, introduces design decisions which should be postponed when a commitment between software applications and hardware platform has been established, and suffers of the all the limitations of coding with respect to modeling: less flexibility, limited reusing, unreadable documentation. Furthermore, a system design given in terms of code is hardly traceable with respect to the initial specification and prevents a meaningful analysis of the system.

The improvement of the current system level design would require new design methods, languages and tools capable of raising the level of abstraction to a point where productivity can be improved, errors can be easier to identify and correct, better documentation can be provided, and embedded system designers can collaborate more effectively. Furthermore, early stages of the design process would benefit from the use of graphical interface tools that visualize the system specification and allow multiple team members to share the relevant information [1]. All these reasons have, therefore, caused more and more increasing interest toward visual software modeling languages like the UML (Unified Modeling Language) [30] able to capture and visualize system structure and behavior at multiple levels of abstraction, and to generate executable models in C/C++/SystemC from system specifications.

Along this research line, we defined a model-driven methodology [27] and a development process [28] for embedded system design. The new design flow is based on the principles of high level modeling, models transformation and automatic code generation of the Model Driven Engineering (MDE) approach. As modeling languages, it involves the UML 2, a SystemC UML profile (for the hardware side), and a multi-thread C UML profile (for the software side). It allows system modeling from a functional executable level down to the Register Transfer Level (RTL).

We here address the problem of analyzing high-level UML-based embedded system descriptions, namely to find techniques for system model validation and verification. 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 satisfaction is guaranteed. There exist different techniques for system design validation. The *scenario-based* one allows the designer to build critical scenarios reflecting given system requirements

to be guaranteed and check for requirements satisfaction. Of course, this technique requires tools able to support automatic scenario execution.

UML-based design methods are not yet well supported by effective validation methods, and, in general, formal model validation and verification techniques are not directly applicable to UML-based models, due to their lack of a precise semantics. Formal methods and analysis tools have been most often applied to low level hardware design. However, these techniques are not applicable to system descriptions given in terms of programs of system-level languages like SystemC, since system description are closer to software programs than to traditional hardware description [31]. So far, the focus in the literature has been mainly on traditional code-based simulation than on design model validation.

To tackle the problem of validating UML-based system models, we combine our SystemC UML modeling language with the Abstract State Machine (ASM) [6] formal notation in order to automatically map a visual UML model into a formal ASM model, and then to exploit well established techniques for ASM model analysis. This approach allows us to functionally validate SystemC UML designs early at high levels of abstraction. In particular, we here present the *scenario-based* validation of embedded system designs provided as SystemC UML models. As a proof-of-concept, the paper reports the results of the scenario-based validation for the Simple Bus case study from the SystemC distribution.

We also present a validation tool, integrated into our model-driven HW-SW co-design environment originally presented in [26], to support the scenario-based validation flow. It makes use of the ASMETA(ASM mETAmodeling) toolset [4] as supporting tools around ASMs.

The choice of the ASMs among other formal methods is intentional and due to the fact that this method (a) comes with a rigorous scientific foundation [6], (b) provides executable specifications and, therefore, it is suitable for high-level model validation, (c) is endowed with a metamodel [11] defining the ASM abstract syntax in terms of an object-oriented representation, and the metamodel availability allows automatic mapping of SystemC UML models into ASM models by exploiting MDE techniques of automatic model transformations [32].

A preliminary version of this work was presented in [12]. We here provide more details on the language for scenarios modelling and the tool components that allow transformations from visual to formal models, and models validation.

This paper is organized as follows. Sect. 1.2 provides some background on the ASMs and their supporting toolset. Sect. 1.3 presents our basic idea on how targeting validation in the ASM context, and presents the language for scenarios construction. Sect. 1.4 focus on the

model validation flow 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. 1.5 provides some results of the scenario-based validation of the Simple Bus case study. Sect. 1.6 quotes some relevant related work. Finally, Sect. 1.7 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 [11, 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], available in both meta-languages OMG/MOF [18] and EMF/Ecore [3], provides a complete object-oriented representation of ASM concepts.

The ASMETAtoolset 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 modeling language for scenario-based validation of ASM models, with its supporting tool, the ASMETAV validator; and the *ATGT* tool that is a test case generator based on the SPIN model checker [16].

3. Scenario-based validation of ASM models

Scenario-based validation of ASM models [10] requires the formalization (complete or incomplete) of the system behavior in terms of an ASM specification, and a *scenario* representing a description of external actor actions and system reactions.

We support two kinds of external actors: the *user*, who has only a black box (i.e. outside) view of the system, and the *observer* having, instead, a gray box (i.e. also internal) view. By allowing two types of actors, we are able to build scenarios useful for classical validation (those including user actions and machine reactions), and scenarios useful for testing activity (those including also observer actions) requiring the inspection of the internal configurations of the machine. Therefore, our scenario-based validation approach goes behind the UML use-cases it was inspired from, and has the twofold goal of model validation and model testing.

A user actor is able to interact, in a black box manner, with the system by *setting* the values of the external environment, so asking for a particular service, waits for a *step* of the machine as reaction to his/her request, and can *check* the values given in *outputs*from the system. An observer actor has the further capabilities of inspecting the internal state of the system (i.e. values of machine functions and locations), to require the *execution* of particular system (sub-)services of the machine, and to check the validity of possible *invariants* of a certain scenario. We describe scenarios in an algorithmic way as interaction sequences consisting of *actions*, where each action in turn is an activity of a user or observer actor, and an activity of the machine as reaction of the actor actions.

3.1 The AValLa Language

The AValLa language has been defined in [10] as a domain-specific modeling language in the context of scenario-based validation of ASM models written in AsmetaL.

Fig. 1.1 shows the AValLa metamodel, which defines the language abstract syntax in terms of an (object-oriented) model. For a formal definition of the AValLa semantics, see [10].

An instance of the class `Scenario` represents a scenario of a provided ASM specification. A scenario has an attribute `name`, an attribute `spec` denoting the ASM specification to validate, and a list of target commands of type ***`Command`***. Additionally, a scenario may contain the specification of some critical properties, here referred to as *scenario invariants*, that should always hold (and therefore checked) for the particular scenario. The composite associations between the `Scenario` class (the *whole*) and its component classes (the *parts*) `Invariant` and ***`Command`*** assures that each part is included in at most one `Scenario` instance.

The abstract class ***`Command`*** and its concrete sub-classes provide a classification of scenario commands. The `Set` command allows the user actor to set the external environment, i.e. to supply values of monitored or shared functions as input signals to the system. The `Check` class

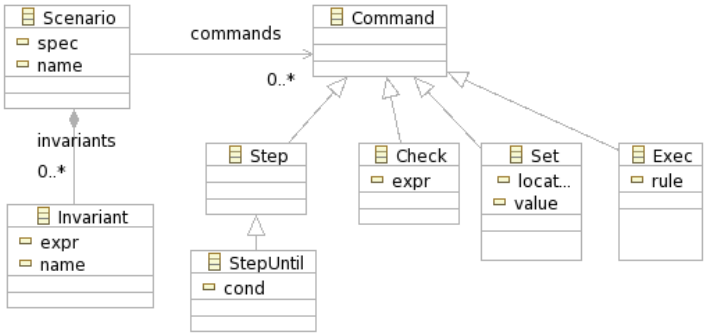

Figure 1.1. The AVALLA metamodel

represents commands supplied either by the user actor to inspect external property values, or by the observer actor to further inspect internal property values in the current state of the underlying ASM. By an `Exec` command, an observer actor may require the execution of particular ASM transition rules performing given system (sub-)services. Finally, commands `Step` and `StepUntil` represent the reaction of the system, which can execute one single ASM step and one ASM step iteratively until a specified condition becomes true.

Examples of scenario scripts are provided in Sect. 1.5 for the `Simple bus` case study.

4. The model-driven validation environment

The scenario-based validation environment has been developed as a component of a more complex co-design environment [26], which allows embedded system modeling, at different levels of abstraction, by using the SystemC UML profile [25] and forward/reverse engineering to/from C/C++/SystemC programming languages.

Fig. 1.2 shows the architecture of the validation component.

The scenario-based validation process starts by applying (**phase 1**) the `UML2AsmM` transformation to the SystemC-UML model of the system (exported from the UML modeler component of the co-design environment [26]). This automatic mapping transform the input visual model into a corresponding ASM model written in AsmetaL.

Once the ASM model is generated, system validation (**phase 2**) is possible by supplying suitable scenarios written in AVALLA .

A brief description of each activity follows. Note that as required skills and expertise the designer has to familiarise with the SystemC UML

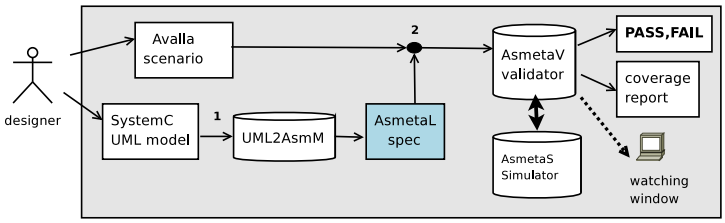

Figure 1.2. Architecture of the validation environment

profile (embedded in the UML modeler), and with very few commands of the AValLa textual notation to write pertinent validation scenarios.

4.1 From SystemC UML models to ASM models

SystemC UML models, provided in input from the co-design tool [26], are transformed into corresponding ASM models (an instance of the AsmM metamodel). This transformation is defined (once for all) by establishing a set of semantic mapping rules between the SystemC UML profile and the AsmM metamodel. This `UML2AsmM` transformation is completely automatized by means of the ATL transformation engine [2] developed as a possible implementation of the OMG QVT [24] standard.

In order to provide a one-to-one mapping (for both the structural and behavioral 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 [20] 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 modeling the behavior 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.

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 subdomain of the predefined abstract domain *PrimChannel*, which is part of the SystemC-ASM component library. An event is mapped into an element of a predefined abstract domain *Event*.

For the behavioral part, a process (a `sc_thread` or a `sc_method`) is mapped into an element of a predefined abstract domain *Process*. A process behavior 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 modeled in terms of stereotyped pseudostates (junction or choice) and states, respectively, a one-to-one mapping is defined between the state-like diagram of the process behavior 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 modeled with a specific design pattern of ASM rule constructors.

As example of application of such mapping, Fig. 1.3 shows the UML notation, the SystemC code, and the resulting ASM (in AsmetaL) for a module.

4.2 Model Validator

Scenarios written in AValLa are executed by means of the AsmetaV validator. It is a Java application which makes use of the AsmetaS simulator to run scenarios. AsmetaVreads a user scenario written in AValLa (see Fig. 1.2), 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, AsmetaVinvokes 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, AsmetaVcaptures any check violation and if none occurs it finishes with a "PASS" verdict. Besides a "PASS"/"FAIL" verdict, during the scenario running AsmetaVcollects

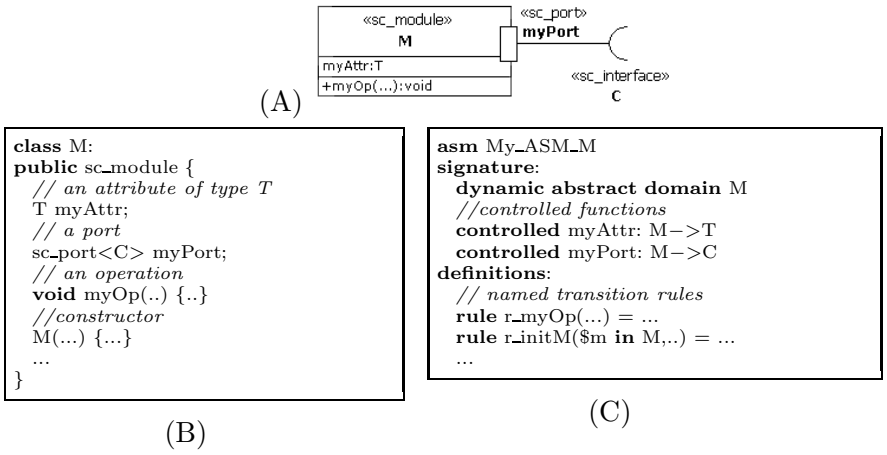

Figure 1.3. A UML module (A), its SystemC code (B) and its corresponding ASM (C)

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 [22].

The Simple Bus system was modelled [25] in a forward engineering flow using the SystemC UML profile. The UML object diagram in Fig. 1.4 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`[3]. 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.

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

In the simplest case, precedence is accorded to the device with higher priority[4], 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).

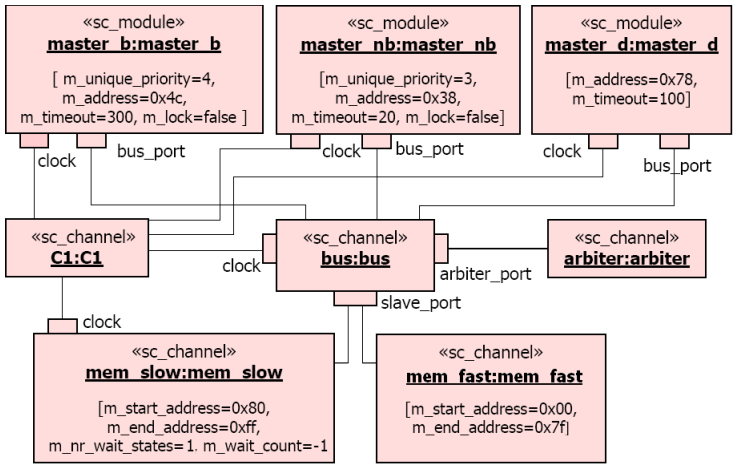

Figure 1.4. Simple Bus – UML object diagram

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 modeling 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 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 address 120 and address 132 are read (`directReadBus`). The actual values must match the expected values.

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

```
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. We assume that the clock period is 15 time units. The bus checks the requests at each negative clock edge. 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 new 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 [23], the authors present a model-driven development and validation process which begins by creating (from a natural language specifi-

cation 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 [23], 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 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 ASMETAATGT 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 [16].

The work in [14] 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 [23, 14], although using the Spec Explorer tool, do not exploit the scenario-based validation feature of Spec Explorer. Indeed, in [13, 5] was shown how Spec Explorer allows scenario-oriented modeling.

In [17], 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 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 analysis 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 [22] 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 [19], originally developed for telecommunication systems, can be adapted to embedded systems to allow validation. For instance, in [9] 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 [15], 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 modeling part) and the ASM formal method and its related ASMETAtoolset (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.

This work is part of our ongoing effort to enact *design flows* that start with system descriptions using UML-notations and produce C/C++/SystemC implementations of the SW and HW components as well as their communication interfaces, and that are complemented by *formal analysis flows* for system validation and verification.

As future step, we plan to integrate ASMETAVwith the ATGT tool of the ASMETAtoolset 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 models in the language of the model checkers, such as the Promela language of the SPIN model checker.

References

[1] Chen, R., Sgroi, M., Martin, G., Lavagno, L., Sangiovanni-Vincentelli, A.L., Rabaey, J. Embedded System Design Using UML and Platforms. In *System Specification and Design Languages* (Eugenio Villar and Jean Mermet, eds.). CHDL Series, Kluwer, 2003.

[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] A. Gargantini, E. Riccobene, and P. Scandurra. A metamodel-based simulator for ASMs. In *Proc. of the 14th Int. ASM Workshop*, 2007.

[9] 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.

[10] A. Gargantini, E. Riccobene, and P. Scandurra. A scenario-based validation language for ASMs. In *ABZ' 08: Proc. of the 1st International Conference on Abstract State Machine, B and Z*. LNCS 5238, p. 71–84, Springer, 2008.

[11] A. Gargantini, E. Riccobene, and P. Scandurra. A Language and a Simulation Engine for Abstract State Machines based on Metamodelling. In *Journal of Universal Computer Science* , Vol. 14, No. 12, p.1949-1983, 2008.

[12] A. Gargantini, E. Riccobene, P. Scandurra, A. Carioni. Scenario-based validation of Embedded Systems. In *FDL' 08: Proc. of Forum on Specification and Design Languages*, pp. 191-196. IEEE press, 2008.

[13] W. Grieskamp, N. Tillmann, and M. Veanes. Instrumenting scenarios in a model-driven development environment. *Information & Software Technology*, 46(15):1027–1036, 2004.

[14] A. Habibi and S. Tahar. Design and verification of SystemC transaction-level models. *IEEE Transactions on VLSI Systems*, 14:57–68, 2006.

[15] 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.

[16] G. J. Holzmann. The Model Checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[17] 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.

[18] OMG. The Meta Object Facility, formal/2002-04-03.

[19] Message Sequence Charts (MSC) ITU-T. Z.120, 1999.

[20] W. Müller, J. Ruf, and W. Rosenstiel. *SystemC Methodologies and Applications*. Kluwer Academic Publishers, 2003.

[21] The Object Managment Group (OMG). `www.omg.org`.

[22] Open SystemC Initiative. `http://www.systemc.org`.

[23] 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.

[24] OMG, Query/Views/Transformations, ptc/07-07-07.

[25] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A UML2 Profile for SystemC 2.1. STMicroelectronics Technical Report, April 2007.

[26] 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.

[27] 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.

[28] 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.

[29] T. Gröetker and S. Liao and G. Martin and S. Swan. *System Design with SystemC*. Kluwer, 2002.

[30] OMG. The Unified Modeling Language. `www.uml.org`.

[31] 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.

[32] Zhang, T., Jouault, F., Bézivin, J., Zhao, J. A MDE Based Approach for Bridging Formal Models. In *Proc. 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*. IEEE Computer Society, 2008.