A Model-driven Validation & Verification Environment for Embedded Systems

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

E. Riccobene
DTI, Università di Milano, Italy
Email: riccobene@dti.unimi.it

P. Scandurra
DTI, Università di Milano, Italy
Email: scandurra@dti.unimi.it

Abstract— **This paper presents a model-driven environment for HW/SW co–design and analysis of embedded systems based on the Unified Modeling Language, UML profiles for SystemC/multi-thread C, and the Abstract State Machine formal method. The environment supports a model-driven design methodology which provides a graphical high-level representation of hardware and software components, and allows C/C++/SystemC code generation from models, a reverse engineering process from code to graphical UML models, and a transparent and tool-assisted system validation and verification based on ASMs.**

I. Introduction

SystemC (built upon C++) has emerged as de facto, open [18], industry-standard language for *system-level* models, specifically targeted at architectural, algorithmic, transaction-level modelling [24]. Recently, a further improvement has been achieved by exploiting lightweight software modelling languages like UML (Unified Modeling Language) [25] to describe system specifications and generate from them executable models in C/C++/SystemC. See the numerous standardization activities controlled by the OMG group [17] like the Schedulability, Performance, and Timing Analysis (SPT) profile, the recent UML extension for SoC (USoC), the SysML proposal, and the MARTE initiative.

In accordance with the design principles of the OMG's *Model-driven architecture* (MDA) [14], we defined a model-driven design methodology for embedded systems [22] based on the UML 2, a SystemC UML profile (for the HW side), and a multi-thread C UML profile (for the SW side), which allows modelling of the system 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) [23] which extends the conventional Unified Process (UP) (by the authors of UML), and by the UpSoC (Unified Process for SoC) sub-process of UPES used for refining the HW platform model.

In this paper, we present our work in progress on complementing our methodology with a *formal analysis process* for high level system validation and verification (V&V) which involves the Abstract State Machine (ASM) formal method [5]. This analysis flow is supported by a V&V toolset integrated into a model-driven HW-SW co-design environment [21] to assist the designer in both the modelling and analysis activity.

Our overall goal is to provide a design and analysis environment where both the software application and the hardware architecture are described together by a multi-views UML model representing the mapping of the functionality (of the software application) onto an architecture, and where the system components can be functionally validated and verified early at high levels of abstraction and in a transparent way (i.e. no strong skills and expertise on formal methods are required).

This paper is organized as follows. Sect. II provides our motivations. Sect. III provides some background on the ASMs and their supporting toolset. Sect. IV describes the architecture of the design/analysis environment with its components features. Sect.V focus on the new V&V component of the environment. Finally, Sect. VI quotes some relevant related work.

II. Motivations

Formal methods and analysis tools have been most often applied to low level hardware design. The higher abstraction offered by most system-level design languages (mostly based on ANSI-C, like SystemC, SpecC, etc.), instead, are not yet amenable to rigourous, formal analysis [26]. The ambiguity in the specifications of the underlying (vastly more expressive) programming languages such as C and C++ makes the creation of formal models for verification even more difficult. As such languages are closer to concurrent software than to traditional hardware description, we propose to address this problem by using formal techniques from *software analysis*, and, going up of a further abstraction step (to further reduce the system design complexity), formal techniques from *model analysis* for UML-like designs capable of eliminating ambiguities in the UML semantics by compiling UML designs in an intermediate formal representation (like ASMs, Object-Z, Petri Nets, etc.). In particular, we adopt the ASM formalism to provide a formal intermediate representation of the intended system. Although the ASM method [5] comes with a rigourous scientific foundation, it can be understood correctly as pseudo-code or Virtual Machines working over abstract data structures. The method boasts a great variety of successful applications in different fields such as: definition of industrial standards for programming and modelling languages, design and re-engineering of industrial control systems, modelling e-commerce and web services, design and analysis of protocols, architectural design, language design, verification of compilation schemas and compiler back-ends, etc. Moreover, a number of ASM tools have been developed for model simulation, model-based testing, verification of model properties by proof techniques or model checkers – see [5] for a detailed description.

III. ASMs and the ASMETA toolset

Abstract State Machines are an extension of FSMs, where unstructured control states are replaced by states comprising arbitrary complex data. The *states* of an ASM are multi-sorted first-order structures, i.e. domains of objects with functions and predicates (boolean functions) 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 method is in [5]. 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, *Turbo ASMs*, to a generalization where multiple parallel agents interact in a synchronous/asynchronous way, *synchr/asynch multi-agent ASMs*. Appropriate rule constructors also allow non-determinism (or existential quantification) and unrestricted synchronous parallelism (universal quantification).

The *ASMETA* (ASM mETAmodelling) toolset [10] [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) [15]. AsmM is also publicly available as expressed in the meta-languages AMMA/KM3 [3] and in EMF/Ecore [8].

The ASMETA toolset includes a textual 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 (as instances of AsmM), 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 a test case generator based upon the SPIN model checker [12].

IV. Environment architecture

Figure 1 shows the HW/SW co-design environment architecture [21]. Components inside dashed lines are under development. 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. 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, and a *V&V toolset* based on the ASM formal method. In our current implementation, the modeler is based on the Enterprise Architect (EA) UML tool [7] by SparxSystems. Details on the new V&V toolset are given in the next section.

V. The V&V toolset

The V&V toolset is built upon the ASMETA toolset. Its essential components are depicted in Fig. 2 together with the phases (denoted with a number and a label) the designer (or analyst) undertakes in the analysis process. The process starts by applying the mapping (**phase 1**) of the SC-UML model of the system (exported from the EA-based modeler) into a corresponding ASM model (written in AsmetaL) providing the basis for analysis.

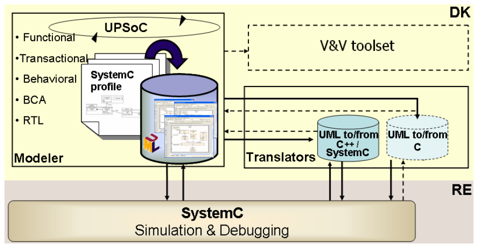

Fig. 1. Tool architecture

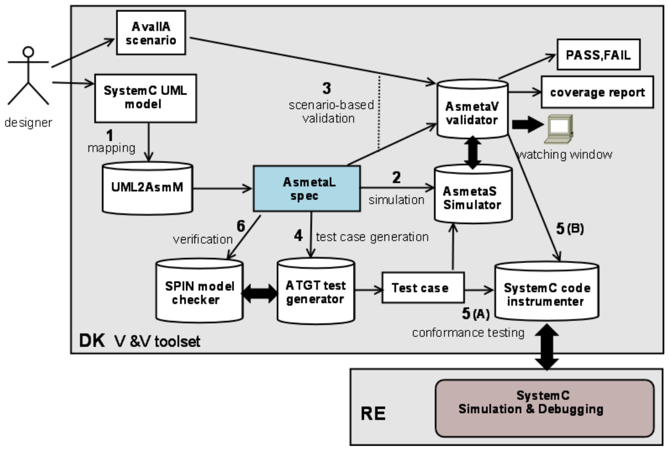

Fig. 2. V&V toolset

Once the ASM formal model of the system is generated, several phases can be executed in parallel: basic simulation (**phase 2**), scenario-based validation (**phase 3**), test-case generation (**phase 4**) and conformance testing (**phases 5 (A)** and **5 (B)**), formal verification by model checking (**phase 6**). Components and interfaces for phases 1, 2, and 3 have been already implemented. Components/interfaces for phases 4 and 6 are currently under development, while those for phase 5 will be tackled in a long-term period.

A brief description of each phase and supporting tool components follows. It should be noted that as required skills and expertise the designer (or analyst) has to familiarize with the SystemC UML profile (embedded in the EA-based modeler), with very few commands of the Avalla textual notation to write pertinent validation scenarios, and with property specification notations like temporal logics.

1. *Mapping:* The semantic mapping is defined (once for all) in terms of a set of transformation rules between the SystemC UML profile and the AsmM metamodel. First, we had to express in terms of ASMs the SystemC discrete (absolute and integer-valued) and event-based simulation semantics taking inspiration from the ASM formalization of the SystemC 2.0

simulation semantics in [16]. We then proceeded to map UML-SystemC concepts (data types, modules, ports, *process* state machines[1], etc.) into ASMs concepts. The `UML2AsmM` transformation is completely automatized by a *model transformation engine* such as the ATL engine [2] (the one we adopted) developed within the Eclipse Modeling Project as implementation of the OMG QVT [20] standard.

2. *Basic simulation:* The AsmetaS simulator [10] 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.

3. *Scenario-based functional validation:* The AsmetaV validator is based on the AsmetaS tool and on the Avalla language. This last provides constructs to express execution scenarios in an algorithmic way as interaction sequences consisting of *actions* committed by the *design actor* to `set` the environment (i.e. the values of monitored/shared functions), to `check` the machine state, to ask for the `exec`ution of certain transition rules, and to enforce the machine itself to make one `step` 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.

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.

4. *Test-case generation:* In accordance with *model-based testing*, in which test cases are derived in whole or in part from a model that describes some aspects of the system under test, the ATGT tool [9] takes as input an ASM model, produces a set of test predicates, translates the original ASM model to Promela – the language of the model checker SPIN [12] used to generate tests –, and generates a set of test sequences by exploiting the counter example generation of the model checker (*test case generation by model checking*). ASMs are therefore used as *test oracles* to predict the expected outputs of units under test. The test cases derived from the ASM model are functional tests on the same level of abstraction as the model. These test cases are known as the *abstract test suite*.

5. *Conformance testing:* Moreover, in order to execute the abstract test suite directly against the system under test (i.e. the SystemC code), an executable test suite must be derived from the abstract test suite (because this last is on the wrong level of abstraction) that can communicate with the system under test. This is done by the SystemC code instrumenter (see Fig. 2) by mapping the abstract test cases generated by ATGT, or even generated by Avalla from the provided scenarios, to concrete test cases suitable for execution. Selected behavioural aspects of the system can be studied, therefore, by *conformance testing*, i.e. by instrumenting the SystemC implementation code from the model.

6. *Verification by model checking:* Every model-checking tool comes with its own modelling language. By model transformations, appropriate links can be provided from ASMETA to model checkers like the well-known SPIN/PROMELA, and therefore encoding ASM models into PROMELA communicating automata. The user has therefore to specify only the properties (linear temporal logic (LTL) formulas, in the case of SPIN) that the final system is expected to satisfy. The model checker then outputs yes if the given ASM model satisfies the given properties, or generates a counterexample otherwise. By studying the counterexample, you can pinpoint the source of the error in the model, correct the model, and try again. The idea is that by ensuring that the model satisfies enough system properties, we increase our confidence in the correctness of the model.

It should be noted that two different levels of model execution for analysis are supported by our environment: (i) the V&V toolset based on AsmetaS and SPIN technologies; (ii) the SystemC simulation and debugging. The first one is more abstract and aimed at high-level functional validation to investigate a model with respect to the user perceptions 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. The second one is low-level, based on SystemC code, and necessary to deliver correct system-level designs. The joint use of these two simulation modes bring the advantage of having specification-based test oracles to be used to drive the SystemC implementation code for conformance testing.

We have been testing our analysis methodology on case studies taken from the standard SystemC distribution. Thanks to the ease in raising the abstraction level using ASMs, we believe our approach scales effectively to industrial systems.

VI. Related work

In [19], 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

[1]SystemC *process state machines* are an extension of the UML statecharts formalism for modelling the behaviour of the reactive SystemC processes.

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 does not want to deal with them.

In [19], 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 specifications 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 and exploration. The ASMETA ATGT tool that we use (see Sect. III) for the same scope 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 [12], i.e. it directly uses ASMs as *test oracles* to predict the expected outputs.

Another limit of the approach in [19] concerns the integration between the abstract model (the ASM specification) and the SystemC implementation model. As Spec Explorer only allows links to C# and Visual Basic, for testing whether a SystemC implementation model satisfies the specification, the designer has to provide explicit bindings to SystemC, i.e. he or she has to manually write a C# interface wrapper that export functions of the SystemC library and of the implementation model to libraries in Spec Explorer. This is heavily manual, time-consuming, and requires expertise in AsmL/C# that go beyond those of the applicant. We intend, instead, to develop the SystemC code instrumenter for conformance testing once for all in a generative manner by model transformation making it reusable for every source SystemC UML model.

The work in [11] 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.

In [13], 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 merely concentrates on providing coverage-directed test suite generation for system level design validation only.

Authors in [6] 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 [18] provides API for transaction-based verification, constrained and weighted randomization, exception handling, and HDL-connection APIs. We aim, however, at the development of formal techniques to augment standard SystemC verification.

Acknowledgment

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.

References

[1] Esterel Studio. www.estereltechnologies.com.

[2] The ATL transformation language. www.eclipse.org/m2m/atl/.

[3] The AMMA Platform. http://www.sciences.univ-nantes.fr/lina/atl/, 2005.

[4] The ASMETA toolset. http://asmeta.sf.net/, 2006.

[5] E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.

[6] Francesco Bruschi, Fabrizio Ferrandi, and Donatella Sciuto. A framework for the functional verification of systemc models. *Int. J. Parallel Program.*, 33(6):667–695, 2005.

[7] The Enterprise Architect tool: www.sparxsystems.com.au/.

[8] Eclipse Modeling Framework. www.eclipse.org/emf/.

[9] A. Gargantini, E. Riccobene, and S. Rinzivillo. Using Spin to Generate Tests from ASM Specifications. In *10th Int. Workshop on Abstract State Machines*, LNCS 2589, p. 263-277. Springer, 2003.

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

[11] A. Habibi and S. Tahar. Design and verification of systemc transaction-level models. *IEEE Transactions on VLSI Systems*, 14:5768, 2006.

[12] Gerard J. Holzmann. The model checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[13] D.A. Mathaikutty, S. Ahuja, A. Dingankar, and S. Shukla. Model-driven test generation for system level validation. *High Level Design Validation and Test Workshop, 2007. HLVDT 2007. IEEE*, pages 83–90, 2007.

[14] OMG. The Model Driven Architecture (MDA Guide V1.0.1). http://www.omg.org/mda/, 2003.

[15] OMG. The Meta Object Facility, v1.4, formal/2002-04-03, 2002.

[16] W. Müller, J. Ruf, and W. Rosenstiel. *SystemC: methodologies and applications*, chapter An ASM based systemC simulation semantics, pages 97–126. Kluwer Academic Publishers, 2003.

[17] The Object Managment Group (OMG). http://www.omg.org.

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

[19] Hiren D. Patel and Sandeep K. Shukla. Model-driven validation of systemc designs. In *DAC '07: Proc. of the 44th annual conference on Design automation*, pages 29–34, New York, NY, USA, 2007. ACM.

[20] OMG, MOF Query/Views/Transformations, ptc/07-07-07, 2007.

[21] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. A model-driven design environment for embedded systems. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, pages 915–918, New York, NY, USA, 2006. ACM Press.

[22] 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.

[23] E. Riccobene, P. Scandurra, A. Rosti, and S. Bocchio. Designing a unified process for embedded systems. In *Int. workshop on Model-based Methodologies for Pervasive and Embedded Software*. IEEE, 2007.

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

[25] OMG. The Unified Modeling Language, v2.1.2. www.uml.org.

[26] Moshe Y. Vardi. Formal techniques for systemc verification; position paper. In *DAC*, pages 188–192. IEEE, 2007.