The ASMETA framework

Paolo Arcaini[1], Angelo Gargantini[2], Elvinia Riccobene[1], and Patrizia Scandurra[2]

[1] Dip. di Tecnologie dell'Informazione, Università degli Studi di Milano, Italy
{paolo.arcaini,elvinia.riccobene}@unimi.it
[2] Dip. di Ing. dell'Informazione e Metodi Matematici, Università di Bergamo, Italy
{angelo.gargantini,patrizia.scandurra}@unibg.it

1 Introduction

The use of formal methods, based on rigorous mathematical foundations, is essential for system development. However, some skepticism exists against formal methods mainly due to the lack of tools supporting formal development, or to the tools' loosely coupling that does not allow reuse of information. The integration and interoperability of tools is hard to accomplish, so preventing formal methods from being used in an efficient and tool supported manner during the system development life cycle.

The ASMETA (ASM mETAmodeling) framework[3] [4,10] is a set of tools around the Abstract State Machines (ASMs). These tools support different activities of the system development process, from specification to analysis, and are strongly integrated in order to permit reusing information about models during several development phases.

ASMETA has been developed [4,11,13] by exploiting concepts and technologies of the Model-Driven Engineering (MDE), like metamodeling and automatic model transformation. The starting point of the ASMETA development has been the *Abstract State Machine Metamodel* (AsmM) [12], an abstract syntax description of a language for ASMs. From the AsmM, by exploiting MDE techniques of automatic model-to-model and model-to-text transformation, a set of software artifacts (concrete syntax, parser, interchange format, API, etc.) has been developed for model editing, storage and manipulation. These software artifacts have been later used as a means for the development of new more complex tools, and the integration within ASMETA of already existing tools, so providing a powerful and useful tool support for system specification and analysis.

After briefly introducing the ASM formal method and its potentiality as system engineering method, we present the ASMETA toolset which provides basic functionalities for ASM models creation and manipulation (as editing, storage, interchange, access, etc.) as well as advanced model analysis techniques (validation, verification, testing, review, requirements analysis, runtime monitoring, etc.).

A suitable set of ASM benchmark examples will be selected for the demo purposes in order to show all the potentialities of the ASMETA framework over different characteristics of the ASM models (parallelism, non determinism, distributivity, submachine invocations, etc.)

2 Abstract State Machines

The *Abstract State Machine* (ASM) method is a systems engineering method that guides the development of software and embedded hardware-software systems seamlessly from

[3] http://asmeta.sourceforge.net/

requirements capture to their implementation. Within a single precise yet simple conceptual framework, the ASM method supports and uniformly integrates the major software life cycle activities of the development of complex software systems. The process of *requirements capture* results into constructing rigorous *ground models* which are precise but concise high-level system blueprints ("system contracts"), formulated in domain-specific terms, using an application-oriented language which can be understood by all stakeholders. From the ground model, by stepwise refined models, the *architectural and component design* is obtained in a way which bridges the gap between specification and code. The resulting *documentation* maps the structure of the blueprint to compilable code, providing explicit descriptions of the software structure and of the major design decisions, besides a road map for system *(re-)use* and *maintenance*.

Even if the ASM method comes with a rigorous scientific foundation [5], the practitioner needs no special training to use the ASM method since Abstract State Machines are a simple extension of Finite State Machines, obtained by replacing unstructured "internal" control states by states comprising arbitrarily complex data, and can be understood as pseudo-code over abstract data structures. 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.

The notion of ASMs formalizes simultaneous parallel actions of a single agent, either in an atomic way, *Basic ASMs*, or in a structured and recursive way, *Structured or Turbo ASMs*. It also supports a generalization where multiple agents interact in parallel in a synchronous/asynchronous way, *Synchronous/Asynchronous Multi-agent ASMs*. Appropriate rule constructors also allow non-determinism (**choose** or existential quantification) and unrestricted synchronous parallelism (universal quantification **forall**).

A complete mathematical definition of the ASMs can be found in [5], together with a presentation of the great variety of its successful application in different fields such as: definition of industrial standards for programming and modeling languages, design and re-engineering of industrial control systems, modeling e-commerce and web services, design and analysis of protocols, architectural design, language design, verification of compilation schemas and compiler back-ends, service-oriented applications, etc.

The ASM method allows a *modeling technique* which integrates static (*declarative*) and dynamic (*operational*) descriptions, and an *analysis technique* that combines *validation* and *verification* methods at any desired level of detail. The ASMETA framework makes the application of this modeling technique practically feasible.

3 The ASMETA tool-set

Concrete syntax and other language artifacts To write ASM models in a textual and human-comprehensible form, a platform-independent concrete syntax, *AsmetaL*, is available, together with a text-to-model compiler, *AsmetaLc*, to parse AsmetaL models and check for their consistency w.r.t. the AsmM metamodel OCL constraints. It is also possible to save ASM models into an XMI interchange format, and Java APIs are available to represent ASMs in terms of Java objects[4].

[4] All these software artifacts have been developed in a generative manner from the AsmM metamodel, by exploiting MDE techniques of automatic model-to-model/text transformations.

Simulator Simple model validation can be performed by *simulating* ASM models with the ASM simulator *AsmetaS* [9] to check a system model with respect to the desired behavior to ensure that the specification really reflects the user needs. AsmetaS supports *invariant checking* to check whether invariants expressed over the currently executed ASM model are satisfied or not, *consistent updates checking* for revealing inconsistent updates, *random simulation* where random values for monitored functions are provided by the environment and *interactive simulation* when required inputs are provided interactively during simulation.

Scenario-based validation A more powerful validation approach is *scenario-based validation* by the ASM validator *AsmetaV* [6]. AsmetaV is based on the AsmetaS simulator and on the *Avalla* modeling 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, 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` (or a sequence of steps) as reaction of the actor actions.

AsmetaRE Use cases are commonly used to structure and document functional requirements, and should be used for validating system requirements. The AsmetaRE [14] automatically maps use case models – written by the tool aToucan[5] according to the approach *Restricted Use Case Modeling* (RUCM) [15] – into ASM models written in AsmetaL. The result of such model-to-text transformation is an executable ASM specification that serves as basis to perform requirements validation by the ASMETA toolset. In particular, an ad-hoc transformation allows also the generation of Avalla scenarios from use cases for scenarios-based validation with the AsmetaV tool.

Model review Model review is a validation technique aimed at determining if a model is of sufficient quality; it allows to identify defects early in the system development, reducing the cost of fixing them. The *AsmetaMA* tool [2] permits to perform *automatic* review of ASMs; it looks for typical vulnerabilities and defects a developer can introduce during the modeling activity using the ASMs.

Model checking Formal verification of ASM models is supported by the *AsmetaSMV* tool [1]; it takes in input ASM models written in AsmetaL and maps these models into specifications for the model checker NuSMV. AsmetaSMV supports both the declaration of *Computation Tree Logic* (CTL) and *Linear Temporal Logic* (LTL) formulas.

Runtime verification Runtime verification is a technique that allows checking whether a run of a system under scrutiny satisfies or violates a given correctness property. *CoMA* (Conformance Monitoring by Abstract State Machines) [3] is a specification-based approach (and a supporting tool) for runtime monitoring of Java software. Based on the information obtained from code execution and model simulation, the conformance of the concrete implementation is checked with respect to its formal specification given in terms of ASMs. At runtime, undesirable behaviors of the implementation, as well as incorrect specifications of the system behavior are recognized.

ATGT Model-based testing aims to use models for software testing. One of its main applications consists in test case generation where test suites are automatically gen-

[5] `http://www.sce.carleton.ca/~tyue/`

erated from abstract models of the system under test. The ATGT tool [8] is available for testing of ASM models. ATGT implements a set of adequacy criteria defined for the ASMs [7] to measure the coverage achieved by a test suite and determine whether sufficient testing has been performed. To build test suites that satisfy some coverage criteria, it implements a technique that exploits the capability of a model checker to produce counterexamples, and it uses the model checker SPIN for the automatic test case generation.

References

1. P. Arcaini, A. Gargantini, and E. Riccobene. AsmetaSMV: a way to link high-level ASM models to low-level NuSMV specifications. In *Abstract State Machines, Alloy, B and Z, 2nd Int. Conference (ABZ 2010)*, volume 5977, pages 61–74. Springer, 2010.
2. P. Arcaini, A. Gargantini, and E. Riccobene. Automatic Review of Abstract State Machines by Meta Property Verification. In C. Muñoz, editor, *Proceedings of the Second NASA Formal Methods Symposium (NFM 2010)*, pages 4–13. NASA, 2010.
3. P. Arcaini, A. Gargantini, and E. Riccobene. CoMA: Conformance monitoring of java programs by abstract state machines. In *2nd International Conference on Runtime Verification, San Francisco, USA, September 27 - 30 2011*, 2011.
4. P. Arcaini, A. Gargantini, E. Riccobene, and P. Scandurra. A model-driven process for engineering a toolset for a formal method. *Software: Practice and Experience*, 41:155–166, Feb. 2011.
5. E. Börger and R. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer Verlag, 2003.
6. A. Carioni, A. Gargantini, E. Riccobene, and P. Scandurra. A Scenario-Based Validation Language for ASMs. In *Proceedings of the 1st international conference on Abstract State Machines, B and Z*, ABZ '08, pages 71–84, Berlin, Heidelberg, 2008. Springer-Verlag.
7. A. Gargantini and E. Riccobene. ASM-Based Testing: Coverage Criteria and Automatic Test Sequence Generation. *J.UCS*, 7:262–265, 2001.
8. A. Gargantini, E. Riccobene, and S. Rinzivillo. Using Spin to Generate Tests from ASM Specifications. In *ASM 2003 - Taormina, Italy, March 2003. Proceedings, LNCS 2589*, 2003.
9. A. Gargantini, E. Riccobene, and P. Scandurra. A Metamodel-based Language and a Simulation Engine for Abstract State Machines. *JUCS*, 14(12):1949–1983, jun 2008.
10. A. Gargantini, E. Riccobene, and P. Scandurra. Model-Driven Language Engineering: The ASMETA Case Study. In *Int. Conf. on Software Engineering Advances, ICSEA*, pages 373–378, 2008.
11. A. Gargantini, E. Riccobene, and P. Scandurra. Integrating Formal Methods with Model-Driven Engineering. In *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on*, pages 86 –92, sept. 2009.
12. A. Gargantini, E. Riccobene, and P. Scandurra. Ten Reasons to Metamodel ASMs. In J.-R. Abrial and U. Glässer, editors, *Rigorous Methods for Software Construction and Analysis*, volume 5115, pages 33–49. Springer Berlin / Heidelberg, 2009.
13. A. Gargantini, E. Riccobene, and P. Scandurra. Combining formal methods and mde techniques for model-driven system design and analysis. *International Journal on Advances in Software*, 3(1,2):1 – 18, 2010.
14. P. Scandurra, T. Yue, A. Arnoldi, and M. Dolci. Functional requirements validation by transforming use case models into abstract state machines. In *Proceedings of the 27th Symposium On Applied Computing (SAC 2012)*, 2012.
15. T. Yue, L. C. Briand, and Y. Labiche. A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation. In *MoDELS*, pages 484–498, 2009.