Modeling with Abstract State Machines: A support for accurate system design and analysis

Egon Börger

Università di Pisa, Dipartimento di Informatica, I-56125 Pisa, Italy
boerger@di.unipi.it

Abstract. We survey applications of the Abstract State Machines (ASM) method for high-level system modeling and for well-documented refinements of abstract models to code.

The outstanding feature of the ASM method is that within a single, precise yet simple, conceptual framework it naturally supports and uniformly links the major activities which occur during the typical software life cycle, namely:

- **requirements capture** by constructing satisfactory *ground models*, i.e. accurate high-level system blueprints, serving as precise contract and formulated in a language which is understood by all stakeholders (see [5]),
- **detailed design** by *stepwise refinement*, bridging the gap between specification and code design by piecemeal, systematically documented detailing of abstract models down to executable code (see [6]),
- **validation** of models by their *simulation*, based upon the notion of ASM *run* and supported by numerous tools to execute ASMs (*ASM Workbench* [18], *AsmGofer* [32], C-based *XASM* [2], .NET-executable *AsmL* engine [23]),
- **verification** of model properties by proof techniques, also tool supported, e.g. by KIV [31] or PVS [20, 24] or model checkers [37, 19, 26],
- **documentation** for *inspection*, *reuse* and *maintenance* by providing, through the intermediate models and their analysis, explicit descriptions of the software structure and of the major design decisions.

The key to the **practicability** of ASMs also under industrial constraints is to be found in the simple and intuitive way in which ASMs support defining most general Virtual Machines (VMs) and their refinements to lower levels of abstraction. ASMs naturally extend Finite State Machines by allowing a) *states* with arbitrarily complex or abstract data structures and b) *runs* with transitions where multiple components act either simultaneously (synchronous parallellism) or asynchronously (like globally asynchronous, locally synchronous Codesign-FSMs). The flexible ASM refinement notion provides a uniform conceptual framework for effectively relating different system views and aspects, in both design and analysis, filling a gap in the UML framework.

The general yet frugal character of the ASM language, namely for a rigorous form of "pseudo-code over abstract data", is the source for the **flexibility** in modeling and analysis which has been experienced in such different areas as:

- industrial standardization projects: models for the ITU-T standard for SDL-2000 [27], the ECMA standard for C# [10], the IEEE-VHDL93 standard [11], the ISO-Prolog standard [8],
- programmming languages: definition and analysis of the semantics and the implementation for the major real-life programmming languages, e.g. SystemC [30], Java and its implementation on the Java Virtual Machine [34], domain-specific languages used at the Union Bank of Switzerland [29], etc.
- architectural design: verification (e.g. of pipelining schemes [12] or of VHDL-based hardware design at Siemens [33, Ch.2]), architecture/compiler co-exploration [35, 36],
- reengineering and design of industrial control systems: software projects at Siemens related to railway [7, 13] and mobile telephony network components [17], debugger specification at Microsoft [3],
- protocols: for authentication, cryptography, cache-coherence, routing-layers for distributed mobile ad hoc networks, group-membership etc., focussed on verification,
- verification of compilation schemes and compiler back-ends [14, 9, 21, 34],
- modeling e-commerce [1] and web services [22],
- simulation and testing: fire detection system in coal mines [16], simulation of railway scenarios at Siemens [13], implementation of behavioral interface specifications on the .NET platform and conformence test of COM components at Microsoft [4], compiler testing [28], test case generation [25].

The AsmBook [15] introduces into the ASM method and illustrates it by textbook examples, which are extracted from real-life case studies and industrial applications. Additional material, including slide decks for lecturers, can be downloaded from the AsmBook website http://www.di.unipi.it/AsmBook/.
To appear in: B. Rumpe and W. Hesse (Eds): Modellierung 2004. Proceedings (Marburg 23.-26.3.2004). GI-Edition Lecture Notes in Informatics, Vol.P-45, pp.235-239.

References

1. S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. In *Proc. 17th ACM Sympos. Principles of Database Systems (PODS 1998)*, pages 179–187. ACM Press, 1998.
2. M. Anlauff and P. Kutter. Xasm Open Source. Web pages at http://www.xasm.org/, 2001.
3. M. Barnett, E. Börger, Y. Gurevich, W. Schulte, and M. Veanes. Using Abstract State Machines at Microsoft: A case study. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines: Theory and Applications*, volume 1912 of *Lecture Notes in Computer Science*, pages 367–380. Springer-Verlag, 2000.

4. M. Barnett and W. Schulte. Contracts, components and their runtime verification on the .NET platform. *J. Systems and Software*, Special Issue on Component-Based Software Engineering, 2002.
5. E. Börger. The ASM ground model method as a foundation of requirements engineering. In N.Dershowitz, editor, *Manna-Symposium*, volume 2772 of *LNCS*. Springer-Verlag, 2003.
6. E. Börger. The ASM refinement method. *Formal Aspects of Computing*, 15:237–257, 2003.
7. E. Börger, H. Busch, J. Cuellar, P. Päppinghaus, E. Tiden, and I. Wildgruber. Konzept einer hierarchischen Erweiterung von EURIS. Siemens ZFE T SE 1 Internal Report BBCPTW91-1 (pp. 1–43), Summer 1996.
8. E. Börger and K. Dässler. Prolog: DIN papers for discussion. ISO/IEC JTCI SC22 WG17 Prolog Standardization Document 58, National Physical Laboratory, Middlesex, England, 1990.
9. E. Börger and I. Durdanović. Correctness of compiling Occam to Transputer code. *Computer Journal*, 39(1):52–92, 1996.
10. E. Börger, G. Fruja, V. Gervasi, and R. Stärk. A complete formal definition of the semantics of C#. *Theoretical Computer Science*, to appear, 2004.
11. E. Börger, U. Glässer, and W. Müller. The semantics of behavioral VHDL'93 descriptions. In *EURO-DAC'94. European Design Automation Conference with EURO-VHDL'94*, pages 500–505, Los Alamitos, California, 1994. IEEE Computer Society Press.
12. E. Börger and S. Mazzanti. A practical method for rigorously controllable hardware design. In J. P. Bowen, M. B. Hinchey, and D. Till, editors, *ZUM'97: The Z Formal Specification Notation*, volume 1212 of *LNCS*, pages 151–187. Springer-Verlag, 1997.
13. E. Börger, P. Päppinghaus, and J. Schmid. Report on a practical application of ASMs in software design. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 361–366. Springer-Verlag, 2000.
14. E. Börger and D. Rosenzweig. The WAM – definition and compiler correctness. In C. Beierle and L. Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, volume 11 of *Studies in Computer Science and Artificial Intelligence*, chapter 2, pages 20–90. North-Holland, 1995.
15. E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis.* Springer, 2003.
16. W. Burgard, A. B. Cremers, D. Fox, M. Heidelbach, A. M. Kappel, and S. Lüttringhaus-Kappel. Knowledge-enhanced CO-monitoring in coal mines. In *Proc. Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA-AIE)*, pages 511–521, Fukuoka, Japan, 4–7 June 1996. .
17. G. D. Castillo and P. Päppinghaus. Designing software for internet telephony: experiences in an industrial development process. In A. Blass, E. Börger, and Y. Gurevich, editors, *Theory and Applications of Abstract State Machines*, Schloss Dagstuhl, Int. Conf. and Research Center for Computer Science, 2002.
18. G. Del Castillo. *The ASM Workbench. A Tool Environment for Computer-Aided Analysis and Validation of Abstract State Machine Models.* PhD thesis, Universität Paderborn, Germany, 2001.
19. G. Del Castillo and K. Winter. Model checking support for the ASM high-level language. In S. Graf and M. Schwartzbach, editors, *Proc. 6th Int. Conf. TACAS*

2000, volume 1785 of *Lecture Notes in Computer Science*, pages 331–346. Springer-Verlag, 2000.
20. A. Dold. A formal representation of Abstract State Machines using PVS. Verifix Technical Report Ulm/6.2, Universität Ulm, Germany, July 1998.
21. A. Dold, T. Gaul, V. Vialard, and W. Zimmermann. ASM-based mechanized verification of compiler back-ends. In U. Glässer and P. Schmitt, editors, *Proc. 5th Int. Workshop on Abstract State Machines*, pages 50–67. Magdeburg University, 1998.
22. R. Farahbod, U. Glässer, and M. Vajihollahi. Specification and validation of the business process execution language for web services. In B. Thalheim and W. Zimmermann, editors, *Abstract Sate Machines 2004*, Lecture Notes in Computer Science. Springer-Verlag, 2004.
23. Foundations of Software Engineering Group, Microsoft Research. AsmL. Web pages at http://research.microsoft.com/foundations/AsmL/, 2001.
24. A. Gargantini and E. Riccobene. Encoding Abstract State Machines in PVS. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Abstract State Machines: Theory and Applications*, volume 1912 of *LNCS*, pages 303–322. Springer-Verlag, 2000.
25. A. Gargantini and E. Riccobene. Using Spin to generate tests from ASM specifications. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Abstract State Machines 2003–Advances in Theory and Applications*, volume 2589 of *Lecture Notes in Computer Science*, pages 263–277. Springer-Verlag, 2003.
26. A. Gawanmeh, S. Tahar, and K. Winter. Interfacing ASMs with the MDG tool. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Abstract State Machines 2003–Advances in Theory and Applications*, volume 2589 of *Lecture Notes in Computer Science*, pages 278–292. Springer-Verlag, 2003.
27. U. Glässer, R. Gotzhein, and A. Prinz. Formal semantics of sdl-2000: Status and perspectives. *Computer Networks*, 42(3):343–358, June 2003.
28. A. Kalinov, A. Kossatchev, A. Petrenko, M. Posypkin, and V. Shishkov. Using ASM specifications for compiler testing. In E. Börger, A. Gargantini, and E. Riccobene, editors, *Abstract State Machines 2003–Advances in Theory and Applications*, volume 2589 of *Lecture Notes in Computer Science*, page 415. Springer-Verlag, 2003.
29. P. Kutter, D. Schweizer, and L. Thiele. Integrating domain specific language design in the software life cycle. In *Proc. Int. Workshop on Current Trends in Applied Formal Methods*, volume 1641 of *Lecture Notes in Computer Science*, pages 196–212. Springer-Verlag, 1998.
30. W. Mueller, J. Ruf, and W. Rosenstiel. An ASM-based semantics of systemC simulation. In W. Mueller, J. Ruf, and W. Rosenstiel, editors, *SystemC - Methodologies and Applications*, pages 97–126. Kluwer Academic Publishers, 2003.
31. G. Schellhorn and W. Ahrendt. Reasoning about Abstract State Machines: The WAM case study. *J. Universal Computer Science*, 3(4):377–413, 1997.
32. J. Schmid. Executing ASM specifications with AsmGofer. Web pages at http://www.tydo.de/AsmGofer.
33. J. Schmid. *Refinement and Implementation Techniques for Abstract State Machines.* PhD thesis, University of Ulm, Germany, 2002.
34. R. F. Stärk, J. Schmid, and E. Börger. *Java and the Java Virtual Machine: Definition, Verification, Validation.* Springer-Verlag, 2001. .
35. J. Teich. Project Buildabong at University of Paderborn. http://www-date.upb.de/RESEARCH/BUILDABONG/buildabong.html, 2001.

36. J. Teich, R. Weper, D. Fischer, and S. Trinkert. A joint architecture/compiler design environment for ASIPs. In *Proc. Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES2000)*, pages 26–33, San Jose, CA, USA, November 2000. ACM Press.
37. K. Winter. Model checking for Abstract State Machines. *J. Universal Computer Science*, 3(5):689–701, 1997.