

A Quick Look At Abstract State Machine Tools

For more than 15 years, abstract state machines have been studied, practiced, and applied in modeling and specification of systems to bridge the gap between formal and pragmatic approaches. ASMs have become a well-known technique and assumed a major role in providing a solid and flexible mathematical framework for specification and modeling of virtually all kinds of discrete dynamic systems. Therefore, with a growing demand on reliability of software and hardware systems, ASM tools that facilitate validation and verification of abstract specifications prove to be beneficial in all stages of the development process.

There is a considerable variety of executable ASM languages that have been developed over the years. The first generation of tools for running ASM models on real machines goes back to Jim Huggins' interpreter written in C [4] and, even further back, to the Prolog-based interpreter by Angelica Kappel [5]. Other interpreters and compilers followed: the lean *EA* compiler [1] from Karlsruhe University, the *scheme*-interpreter [3] from Oslo University, and an experimental EA-to-C++ compiler developed at Paderborn University. Besides practical work on ASM tools, conceptual frameworks for more systematic implementations were developed. The work on the *evolving algebra abstract machine (EAM)* [2], an abstract formal definition of a universal ASM for executing ASM models, contributed to a considerably improved understanding of fundamental aspects of making ASMs executable.

Based on such experience, a second generation of more mature ASM tools and tool environments was developed, among which one can refer to *AsmL*, *Xasm*, *ASM Workbench*, and *AsmGofer*. In the following pages, we introduce a number of these second generation projects that are currently active, briefly present their goals and their current stage of development, and provide some contact information for future references.

We would like to thank Angelo Gargantini, Nikolai Tillmann, Giuseppe Del Castillo, Pavel Vasilyev, Martin Ouimet, and Matthias Anlauff (in no particular order) for kindly providing us with information on these projects.

Developing a metamodel for the ASMs and a tool set around it.

main contact:

Elvinia Riccobene <riccobene@dmi.unict.it>

affiliation:

University of Milan and University of Bergamo

other team members:

Angelo Gargantini, Patrizia Scandurra

home page: <http://asmeta.sourceforge.net>

current development stage:

Beta version released.

The Asmeta project aims to define the Abstract State Machine Metamodel (AsmM, in brief), a metamodel for the *Abstract State Machines* (ASMs) formal method following the guidelines of the *Model-Driven Engineering* (MDE).

We exploit the metamodeling approach suggested by MDE to achieve the goals of developing a unique notation for ASM, a notation independent from any specific implementation syntax and allowing a more direct encoding of the ASM mathematical concepts and constructs, and tackling the problem of ASM tool interoperability.

The AsmM can be seen as a *pivot metamodel* towards a definition of a standard family of languages for the ASMs and a systematic integration of ASM tools. Through metamodels and model transformations, the metamodel of each tool (usually not given explicitly, but is built-in into the tool that provide it) is linked to those of other tools by the logical pivot metamodel which abstracts a certain number of general concepts about ASMs.

Starting from the AsmM we have derived a concrete syntax (AsmetaL : the Asmeta Language), namely an EBNF (extended Backus-Naur Form) grammar derived from the AsmM (the abstract syntax) as a notation to be used by modelers to effectively write ASM models in a textual form.

We are also developing a tool set to support the use of Asmeta in practice:

- (available) the AsmetaL compiler: to compile an Asm specification written in AsmetaL to xmi
- (available) the AsmetaS simulator: to simulate an Asm specification
- (available) Asmee: a plugin for eclipse for AsmetaL and AsmetaS
- (available) the ATGT: a test generator for Asms
- (work in progress) a graphical editor
- (work in progress) AsmetaL to CoreASM translator

AsmL

Date of Birth: 2000

AsmL is an advanced ASM-based language taking advantage of the .NET runtime.

main contact:

Nikolai Tillmann <nikolait@microsoft.com>

affiliation:

Foundations of Software Engineering

other team members:

Colin Campbell, Wolfgang Grieskamp, Yuri Gurevich, Wolfram Schulte, Margus Veanes

home page: <http://research.microsoft.com/fse/asml/>

current development stage:

Released. Available for academic use.

AsmL is the Abstract State Machine Language. It is an executable specification language based on the theory of Abstract State Machines, developed originally by Yuri Gurevich. The current version is embedded into Microsoft Word and uses XML and Word for literate specifications. It is fully interoperable with other languages taking advantage of .NET runtime.

Providing a framework for quickly and easily developing, in a functional language, analysis and transformation tools for ASM.

main contact:

Giuseppe Del Castillo <giuseppe.delcastillo@onlinehome.de>

affiliation:

Design of Parallel Systems, Paderborn University
Prof. Rammig (1995-2000)

other team members:

Kristen Winter (co-authored the interface to the SMV model checker)

current development stage:

The type checker and interpreter are fully functional. The interface to the SMV model checker is working, but prototypical (several missing features). The development was halted in 2001. The tool is currently not available on the Web. A revised version is to be re-released soon under open source conditions.

The ASM Workbench provides a framework for the systematic development of ASM tools. It is based on a specification language called ASM-SL (ASM-based Specification Language), which includes a typed version of the language of Abstract State Machines and additional constructs to define the underlying data model with the help of generic data types, such as lists, finite sets and finite maps.

Various tools can be developed as modules building upon the Abstract Syntax Tree (AST) representation, the type system and the standard function library provided. The implementation language is Standard ML.

The main tools included in the basic ASM Workbench distribution are a type checker and an interpreter. The interface to the SMV model checker (ASM2SMV) has been implemented as an additional module.

The CoreASM project focuses on the design of a lean executable ASM (Abstract State Machines) language, in combination with a supporting tool environment for high-level design, experimental validation and formal verification of abstract system models.

main contact:

Roozbeh Farahbod <info@coreasm.org>

affiliation:

Simon Fraser University, University of Pisa

other team members:

Vincenzo Gervasi, Uwe Glässer, George Ma, Mashaal Memon,
Mike-Ming Su

home page: <http://www.coreasm.org>

current development stage:

Beta version 0.9.1 released. Available for academic use.

Model-based systems engineering can benefit from *abstract executable specifications* as a tool for design exploration and experimental validation through simulation and testing. The CoreASM language emphasizes freedom of experimentation and supports the evolutionary nature of design as a product of creativity. It is particularly suited to Exploring the problem space for the purpose of writing an *initial specification*. The CoreASM language allows writing of highly abstract and concise specifications by minimizing the need for encoding in mapping the problem space to a formal model, and by allowing explicit declaration of the parts of the specification that are purposely left abstract. The principle of minimality, in combination with robustness of the underlying mathematical framework, improves modifiability of specifications, while effectively supporting the highly iterative nature of specification and design.

The CoreASM project is not the first attempt to make ASMs executable. However the main commonality between our approach and others is simply that they provide platforms which execute specifications with ASM-like semantics. Beyond this, CoreASM departs in its approach and spirit of principle from its predecessors. The project's core ideology, consisting of two simple tenets, both serves as a motivation and as a guide development of the engine, the language, and the supporting tool environment:

- the preservation of pure ASM semantics, and
- ensuring freedom through extensibility.

Verification by simulation for real-time abstract state machines with constraints

main contact:

Pavel Vasilyev, <pvasilyev@ya.ru>

affiliation:

University of Paris 12, LACL and University of St Petersburg

other team members:

Prof. Daniele Beauquier, Prof. Anatol Slissenko, Prof. Igor Soloviev

current development stage:

Working prototype

The goal of the project is to develop a simulator of Real-Time Abstract State Machines. The simulator is used for on-the-fly verification of formulas in an expressible timed predicate logic (First Order Timed Logic). Time can be continuous or discrete and it can be used explicitly in specifications. Guards can be defined by linear inequalities with time. Two semantics are considered: with and without non-deterministic bounded delays between actions. The simulator is easily configurable. Simulation tasks can be generated according to descriptions in a special language. The following parameters must be defined before the simulation: external functions definition, delays settings, constraints specification, and others.

Spec Explorer

Date of Birth: 2003

Spec Explorer is a software development tool for advanced model-based specification and conformance testing.

main contact:

Wolfgang Grieskamp <wrrg@microsoft.com>

affiliation:

Foundations of Software Engineering

other team members:

Colin Campbell, Yuri Gurevich, Wolfram Schulte, Nikolai Tillmann,
Margus Veanes

home page: <http://research.microsoft.com/specexplorer/>

current development stage:

Released (version 1.0.9315). Available for academic use.

Spec Explorer is a software development tool for advanced model-based specification and testing, in particular conformance testing. Spec Explorer can help software development teams detect errors in the design, specification and implementation of their systems. The tool is intended to be used by software designers, implementers and testers. Spec Explorer supports the software modeling languages Spec# and AsmL.

To specify, analyze, and verify embedded real-time systems.

affiliation:

Embedded Systems Laboratory, Aeronautics and Astronautics, MIT

main contact:

TASM group (tasm@mit.edu)

other team members:

Kristina Lundqvist, Martin Ouimet, and David Wang

home page: <http://esl.mit.edu/tasm>

current development stage:

Released (Alpha 1, version 1.0.0.7)

The Timed Abstract State Machine (TASM) toolset is a graphical integrated development environment for real-time system engineering. The toolset implements the features of the TASM language, which is an extension of Abstract State Machines (ASM) to incorporate the specification of functional and non-functional properties. The non-functional properties that can be specified in TASM are time and resource consumption. The toolset provides facilities to write, edit, simulate, and analyze TASM specification. The simulation of specifications can be achieved in a step-by-step fashion, with graphical depictions of time and resource consumption.

For its analysis, the toolset incorporates the SAT4J SAT solver and the GNU GLPK mixed integer programming solver to automatically verify completeness and consistency of TASM specifications. Future plans for the toolset include integrating the UPPAAL model checker to verify execution time properties of specifications. Planned functionality for the toolset include the automated generation of test cases, based on TASM specification. Furthermore, the toolset will support a theory of refinement to trace functionality through TASM specifications at different levels of abstraction.

The TASM toolset is entirely written in Java, using a number of open source libraries. The TASM toolset will eventually become open source.

Xasm: Extensible Abstract State Machines Date of Birth: 1999

To develop a programming language whose semantics is based on a minimal, imperative mathematical machine.

main contact:

Matthias Anlauff <matthias.anlauff@sap.com>

Philipp W. Kutter <kutter@montages.com>

affiliation:

German National Research Center for Computer Science, ETH Zurich,
continued at Montages.com

home page: <http://www.xasm.org>

current development stage:

Beta, currently limited support available.

Xasm is an implementation of sequential ASMs focusing on the generation of efficient executable programs simulating the run of the specified ASM. In general, the main design goals of Xasm can be given as follows:

- full support of the ASM language as defined in the Lipari-Guide;
- efficient execution of generated executables;
- comfortable animation and debugging of ASM specifications;
- component-based library concept for managing large-scale specifications;
- external language interface for integrating ASM specifications in other systems.

References

1. B. Beckert and J. Posegga. leanEA: A Lean Evolving Algebra Compiler. In H. Kleine Büning, editor, Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL'95), volume 1092 of *LNCS*, pages 64–85. Springer, 1996.
2. G. Del Castillo, I. Durdanović, and U. Glässer. An Evolving Algebra Abstract Machine. In H. Kleine Büning, editor, Proceedings of the Annual Conference of the European Association for Computer Science Logic (CSL'95), volume 1092 of *LNCS*, pages 191–214. Springer, 1996.
3. D. Diesen. *Specifying Algorithms Using Evolving Algebra. Implementation of Functional Programming Languages*. Dr. scient. degree thesis, Dept. of Informatics, University of Oslo, Norway, March 1995.
4. J. Huggins. An offline partial evaluator for evolving algebras. Technical Report CSE-TR-229-95, University of Michigan, 1995.
5. A. M. Kappel. Executable Specifications Based on Dynamic Algebras. In A. Voronkov, editor, *Logic Programming and Automated Reasoning*, volume 698 of *Lecture Notes in Artificial Intelligence*, pages 229–240. Springer, 1993.