European Journal of Computer Science and Information Technology

Vol.2,No.4,pp.1-7, December 2014

Published by European Centre for Research Training and Development UK (www.eajournals.org)

A SURVEY PAPER ON ABSTRACT STATE MACHINES AND HOW IT COMPARES TO Z AND VDM

[1]Dominic Damoah, [2]Edward Ansong, [3]Henry Quarshie, [4]Dominic Otieku-Asare Damoah [5] Steve Thorman, [6]Roy Villafame

[1234]Department of Computer Science
Valley View University
Box AF 595 Adenta
Accra, Ghana

[56] Department of Computer Science and Engineering
Andrews University
Berrien Springs, Michigan, United States

ABSTRACT: *This codeless method of programming using ASMs is not only capable of producing transparent software but reliable, proven and checked, all the way through its specification as ground model to its implementation. In other words evolving algebra method supports all the software life-cycle phases from initial specifications or requirement gathering to executable code. Above all, this method is platform independent and the ease with which computer scientist can build simple and clean system models out of evolving algebra pushes further the potential it needs to influence future industrial formal methods techniques. This formal specifications and designs demonstrate the consistency between the specifications and the code for all possible data inputs at various levels of abstractions.*

KEYWORDS*:* Formal Methods, Evolving Algebra, Specification, Verification

INTRODUCTION

Abstract State Machines (ASM) is a typical formal methods technique which leans heavily on Evolving Algebras with a wide variety of applications in both Hardware and Software Engineering practices. It is a reliable, less expensive and very fast method of system design process for industrial applications. This is a mature engineering discipline for developing, designing and analyzing complex computer systems. The mathematical principles underlying this approach is the abstraction of universes, dynamic functions or abstract structures which are very instrumental at proving the consistency and correctness of the semantics of algorithms and implementation of programming languages. It will be observed later in this paper that formal requirements and design specifications with ASMs make it possible to come out with real—life programming languages, compilers, protocols and architectures and the proofs to verify them. This is what evolving algebra seeks to integrate at various levels of abstraction. The general levels of abstraction determine how the complexity of large systems could be solved. Evolving Algebra makes it possible to have at any level of abstraction and also independently both state data and operations on that state. In [13],[15] N. Wirth: states that "... Data in the first instance represent abstractions of real phenomena and are preferably formulated as abstract structures not necessarily realized in common programming languages." There is interdependency between algorithms and data structures which make it important for the designer using ASM to harness. In Wirth's words: [16] states that, "It is clear that decisions about structuring data cannot be made without knowledge of the algorithms applied to the data and that, vice versa, the structure and choice of algorithms often depend strongly on the structure of the underlying data. In short, the subjects of program composition and data structures are inseparably intertwined." Put differently this method just as is the common practice in systems engineering defines the domain and functions of a system which leads to structures in Mathematics. Hence this method allows the abstraction of evolving algebra to be made without introducing complexities taking advantage of systematic stepwise refinements. This ensures that rigorously transforming an abstract high level model into a correct implementation and effective controllable construction of reliable computer-based systems are inherent advantages of using the formal methods approach [12][14].

ASM has useful application in specification and analysis of Virtual machines, processor architectures, protocols embedded control software and requirement capture.

In [5] ASM has been applied to real-time modeling and verification in Railroad Crossing problem. In [7] an ASM based prototype system is described for specifying electronic commerce applications. FALKO is a software system for railway simulation. The software consists of three components, namely the train supervision, interlocking system, and the process simulator recently developed using ASM language and C++ [8]. As a result of ASM today Java programs on our Java machine, can be compiled into byte code and executed with JVM machine. [8] and [9].

WHAT ARE ITS GOALS IN THE CONTEXT OF FORMAL METHODS?

The need to ensure correctness for safety, life and mission critical systems have attracted and challenged researchers to find rigorous mathematical and experimental analysis of complex computer systems looking for relevant problems beyond those of semantics of programming languages. The discussion led to the questions of what constitute proofs, the role of grounds models and how mathematical tools such as evolving algebra could be used to extend the application of ASMs to the specification and verification of complex computer systems. Evolving algebra methodology has been used in the context of designing, testing, verifying, documenting, maintaining, understanding and teaching algorithms in the fields of programming languages, architectures, distributed and real- time protocols, etc. The factors below are the salient issues in evolving algebra which set up the context of ASMs. As earlier mentioned ASMs provides freedom of abstraction by which evolving algebras use universes, dynamic functions and states as static algebras to support the software life—cycle phases from initial specifications to executable code, through stepwise refinement. The extensibility and reuse features inherent in evolving algebra make it adaptable through the simple mechanism for information hiding and defining precise abstract interfaces. Evolving algebra employs oracle functions (both dynamic and static) and externally alterable functions without putting any restrictions on them. This means that it is within the power of the designer to determine how users of a system interface it. As a result, the evolving algebra approach helps to ensure that programs, once developed, can be extended, maintained and reused as components of larger systems in a systematic and reliable way.

Another important concept which has gained widespread use in system development is the global and locality principles. The use of locality property as against the global property often to leads to modularization of code designs. The evolving algebra method provides the platform for the interaction between global and local dynamics viewing system states as static algebras and allows local updates in a uniform manner without giving rise to a cleaner programming style than is possible with standard imperative programming languages. Synchronous parallelism means that AsmL has transactional semantics. The language was motivated by Z and VDM but with the sole interest of simplicity. The language supports synchronous parallelism, finite choice, sequential composition and exception handling.

Conceptually AsmL is massively parallel. One state-change of an AsmL program may be a complex transaction involving numerous subprograms. This allows you to avoid unnecessary sequentialization imposed by conventional programming languages and to make your programs clearer.

AREAS OF APPLICATION

ASMs have been applied to both hardware and software engineering successfully in the recent past. Since 1993, it has been adapted and used to extend models through vertical and horizontal refinements of the basic ASM models for the implementation of the language Prolog. Colmerquer's Prolog, IBM's Protos-L , Lloyd's and Hill's programming language Godel, B Mi'1ller's object oriented Prolog, Concurrent Prolog, Pandora and Sauer's adaptations of the prolog models are a few to be mentioned where ASMs have been applied to extend prolog compilers. "Through the work of software engineers at IBM, BIM, Quintus and Siemens to develop commercial implementation of Prolog the usefulness of the ASM concept became apparent for supporting changing designs in an industrial standardization and development process. The flexibility gained is realized as the daily duties to rigorously model and document design decisions, building ground models, to adapt to abstract models to changing requirements and to refine them in successive steps(linking to executable codes) and then into prototypes,"[4]. The method of successive refinement of ASMs finds useful applications in other programming languages such as Smalltalk, COBOL, C and C++. "The modeling of object oriented programming language feature is taken up once more in Ann Arbor, this time using the refinement method to extend the ASM model for C to one for C++" [3]. Also Schmid has used the Production Cell ASM for generating C++ code whose structure allow to trace the specification to support the reliability of code inspection [6]. ASM are used for architecture design and virtual machines. It was used to develop the APE100 architecture a floating point intensive numerical simulator. It was used to provably correctly decompose the control unit of the zCPU. ASM based architecture verification method has been used to develop the standard pipelining scheme for Hennessey and Patterson's RISC machine DLX. For virtual machines ASM has been used to model Parallel Virtual Machines (PVS) as distributed ASM which led to ground model for designing the hardware design language called VHDL.

An industrial application of ASMs comes with the abstract operational ASM a new definition of industrial standards for the design language SDL for distributed systems. Engineers are able gradually use abstract models to verify the validity and correctness of their inventions through mathematical proofs[11].

WHO ADVOCATES IT?

The underlying mathematical principles were initiated by Dijkstra's basic concept of abstract machines and the fundamental idea use by Tarski structures otherwise known as abstract states. Hence the name "Abstract State Machines" Yuri Gurevich pushed the idea of ASM further in his bid to sharpen the Church—Turing thesis in 1984 when it was first revisited under the name dynamic structures and has become the father of ASM. A year later the program was formulated in a note to the American Mathematical Society from where we quote: "First, we adapt Turning's thesis to the case when only devices with bounded resources are considered. Second, we define a more general kind of abstract computational devices, called dynamic structures, and put forward the following new thesis; every computational device can be simulated by an appropriate dynamic structure of appropriately the same size— in real time: uniform family of computational devices can be uniformly simulated by an appropriate family of dynamic structures in real time. In particular every sequential computational device can be simulated by an appropriate sequential dynamic structure." [4]. Yet it did not gain popularity until 1987 when Yuri Gurevich explained the concepts of ASMs using the examples of Turning machines, stack machines, static algebras, the strcpy example and Static, dynamic and external functions.

E Borger another leading authority in ASMs also pushes forward to where it is now with 'his model for Prolog which solved several problems the ISO Prolog standardization committee had faced for many years. Egon Boorger championed this approach and the termed ground model of the system and used successive refinement methods to facilitate the reflection of arbitrary abstraction levels. Other advocates include Dean Rosenzweig, Uwe Glasser, J. Higgins J Lipton, Blass and host of others who have done extensive research into Evolving Algebra and contributed immensely to the semantics and implementation of some complex programming languages like Prolog, Protos—L, java C ,C++,VHDL, Occam , protocols and architecture and real time algorithms

HOW ASM COMPARES TO Z AND VDM

ASM, Z and VDM are all mathematically based formal methods techniques relying on first order logics. Mathematical symbols used in all cases have similar meaning. ASM, Z and VDM are specification languages normally used in the software design process. They are all language independent. They are used when a system has correctness as a concern, such as in safety critical, and security-critical systems, systems in which the cost of system failure is catastrophic and systems where governing bodies mandate their use. All the formal methods under consideration are used to prove satisfaction of requirements arising in the refinement of a design. Formal methods utilize either a property-oriented or model-oriented approach and have different levels of rigor. Model-oriented formal methods specify system behavior by the construction of a mathematical model with an underlying state (data) and a collection of operations on that state. Property-oriented formal methods define system behavior indirectly by stating a set of properties, usually in the form of axioms that the system must satisfy. ASM, VDM and Z are all of the model—oriented type [2]. They all have application tools with types, syntax and proof checkers. ASM, Z and VDM describe objects of systems in term of functions and the relation defined on them. But ASM differs from Z and VDM because it is purely based on structures which exclude relation. Unlike the other formal methods, such as Z and VDM, often used primarily for requirements specification, ASM goes beyond requirements specification to actual design. An important feature of the ASM is its ability to simultaneously execute more than one update. For example the normal sequential processes involved in swapping using Z and VDM will definitely require an intermediate storage but this is not applicable in ASM. The swapping will take place simultaneously eliminating the introduction of temporal location to hold one of the variables. Although Z and VDM are used primarily to describe the characteristics of any system under consideration, ASM goes beyond the properties to specify coding specifics and how the coding is to be implemented. It is a widely known fact that all formal methods techniques cannot guarantee infallible correctness and safety but evolving algebra proves relative correctness promising engineering and industry. The mathematical methods underlying ASMs are open framework where other systems can be integrated easily. ASM has very high degree of freedom of proof and language. This flexibility allows ASMS to be integrated into existing systems which others methods such as Z and VDM will rigidly require a total change.

In [10] J. Huggins asserts that the determination of the correctness of a specification executing the specification directly helps a great deal. A specification methodology which is executable allows one to test for errors in the specification to verify the correctness of a system by experimenting with various safety properties. Methods such as VDM, Z, or process algebras are not directly executable.

LANGUAGE FEATURES

The language features of AsmL were chosen to give the user a familiar programming paradigm. For instance, AsmL supports classes and interfaces in the same way as C# or Java do. In fact all .NET structuring mechanisms are supported: enumerations, delegates, methods, events, properties and exceptions. Nevertheless, AsmL is primarily a specification language. Users familiar with the specification language literature will find familiar data structures and features, like sets, sequences, maps, pattern matching, bounded quantification, and set comprehension. But the crucial features of AsmL, intrinsic to ASMs, are massive synchronous parallelism and finite choice. These features necessarily changing the entire system in any way at any level of abstraction. This object oriented approach find useful applications in modeling parallel or distributed systems (PVM). Another feature of evolving algebra is the success obtained by tools for program development and verification which can be customized to suit a particular application domain. This is made possible by separating specifications from verifications. The flexibility of evolving algebras makes easy integration into existing development systems for the purposes of practical design method. Evolving algebra supports freedom to choose basic objects and actions to be performed on states based on the ground model and methods of abstraction and apply stepwise refinement (vertical or horizontal) to fine tune the system. This is why ASM is language independent in computer system design. The primary reasons why software project fail is the poor method of requirement gathering and how computer solutions could be used to model the real world situation. Evolving algebra is able to provide satisfactory links to application domains by appropriate ground models usually flexible, simple, and concise and falsifiable. It describes the interfaces between the designer and the users on the discussions on the “whats” but not the “hows” Evolving algebra has the potential to scale up to large complex systems including hardware and software co-design because of the techniques for crossing abstraction levels. This is not realized with other formal method techniques. Finally evolving algebras use only standard mathematical notation hence one is spared of the overhead of learning new symbols, syntax and semantics of any particular formal system. This means that any average computer scientist could easily learn and implement even though the method has a rigorous mathematical foundation. This indicates the more reason it has high scalability in industry.

RATIONAL

In this paper we have pointed out the strengths and weaknesses of the Abstract State Machine. Most of the time, this document is very solid because it contains a completely defined architecture and detailed descriptions of the features of the language. In this presentation, we proposed a number of comments concerning the advantage and difficulties of using ASM, Z and VDM methods. We tried to explain why the introduction of such methods are accepted or rejected by some industrial players.

In this paper, we have presented a brief survey of a comparative study of the formal methods ASM, Z and VDM, which relies on refinement and proofs to rigorously transform an abstract high level model into a correct implementation. If ones need is motivated by the interest of simplicity then the synchronous parallelism, finite choice, sequential composition and exception handling are great urge worth considering. The need to ensure correctness for safety, life and mission critical systems should redirect software engineers to this rigorous mathematical for analysis of complex computer systems with capabilities way beyond those of semantics of other programming languages. ASM is used in the context of designing, testing, verifying, documenting, maintaining, understanding and teaching algorithms in the fields of programming languages, architectures, distributed and real- time protocols. The mathematical rigorousity underpinning ASM ensures that programs, once developed, can be extended, maintained and reused as components of larger systems in a systematic and reliable way.

It should be noted that the language also allows you to avoid unnecessary sequentialization imposed by conventional programming languages and to make your programs clearer. Engineers are able to use the approach to verify the validity and correctness of their inventions. The stepwise refinements of the basic ASM models has greatly supported the implementation of various flavours of the language Prolog. The method of successive refinement of ASMs finds useful applications also in other programming languages paradigms. Suffice to say that the techniques is also good for architecture design and virtual machines. It is a widely known fact that all formal methods techniques cannot guarantee infallible correctness and safety but the underpinning mathematical models for ASML proves relative correctness promising engineering and industry.

CONCLUSION

The specification methodology used in ASM is precise in syntax and semantics. This makes less difficult of understand and apply in the description of the system under construction. ASMs use classical mathematical structures to describe states of a computation; structures are we1l—understood precise models. This is a distinguishing factor from informal methodologies. ASM programs use an extremely simple syntax, which makes reading and writing specification as easy as writing or reading pseudo-code. This is why the ASM has gained popularity among even inexperienced computer scientists and they prefer to use it than other specification methods, which use complicated syntax whose semantics are more difficult to read and write.

ASMs are very useful in describing systems at several different levels of abstraction. This makes applicable to both medium to large scale industrial applications. ASM is useful in many application areas: sequential, parallel, and distributed systems; abstract-time and real-time systems; finite-state and infinite—state domains. Some methodologies such as finite model checking, timed input-output automata, and other temporal logics are confined to their specific application domains. ASMs top them all by being useful in all of these domains. As noted above in the forgoing discussion ASM has made tremendous strides into the fields of engineering and computing in the design and analysis of complex computer systems. ASM provides reliable, less expensive and very fast method of system design process for industrial applications. At the moment there are no known weakness found yet. What is known is that advocates stand the risk of isolating themselves from computer science and the fact that if it is not conscientiously applied to solve industrial problem it will die out prematurely. Hence young vibrant researchers need to be attracted to it and sustained. Above all it can also be said that ASM is not the panacea of all problems in the industry. It is yet to be fully explored and the best cases of its application and proofs are all based on relative correctness.

REFERENCES

[1] Yuri Gurevich, Benjamin Rossman and Wolfram Schulte SemanticEssence of AsmL: Extended Abstract Microsoft Research
[2] An Analysis of Two Formal Methods: VDM and Z 13 August 1997 Authored by: Thomas McGibbon
[3] C. Wallace. The Semantics of the C++ Programming language. In E Borger editor, Specification and Validation Methods pages 131, Oxford University Press 1995.
[4] Egon Borger. The Origins and the Development of the ASM Method for High Level System Design and Analysis (University di Pisa, Italy)
[5] C. Heitmeyer and D. Mandroli. Formal Methods for Real- Time Computing Volume Trends in

Software 5 1996.
[6] J Schmid. Compiling Astract State Machines to C++ Journal of Universal Computer Science. www.tydo.de/ProductionCell
[7] B. Fordham S. Abiteboul and Y. Yesha. Evolving Databases An application to Electronic Commerce. I proceedings of International database Engineering and Applications Symposium (IDEAS), August 1997
[8] Leiter: Prof. Dr. Helmuth Partsch Refinement and Implementation Techniques for Abstract State Machines Dissertation zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.) im Fachbereich Informatik der Universit"at Ulm Vorgelegt on,Joachim Schmid aus Tuttlingen Abteilung f 'ur Programmiermethodik und Compilerbau hflpz//Vts.uni— lm.de/docs/2002/1578/Vts l578.pdf
[9] Yuri Gurevichyand James K. I-Iugginsy The Railroad Crossing Problem: An Experiment with Instantaneous Actions and Immediate Reactions_EECS Department, University of Michigan, Ann Arbor, MI, 48109-2122, USA. February 7, 1997
[10] Jim Huggins Abstract State Machines http://www.eecs.umich.edu/gasm/intro.html
[11]J.-R. Abrial. Formal methods in industry: Achievements, problems, future. In Proc.ICSE'06, Shanghai (China), May 2006. ACM,
[12] Jean.-Raymond Abrial. Formal Methods: Theory Becoming Practice, ETHZ, Zurich, Switzerland, 2007.http://www.cs.man.ac.uk/~banach/CSIComm-May07-FM-Papers/030Abrial.pdf
[13]Egon Boerger, Why use evolving algebras for hardware and software engineering? SOFSEM'95: Theory and Practice of Informatics, 1995
[14] Egon Borger. The Abstract State Machines Method for High-Level System Design and Analysis in Formal Methds: State of the Art and New Directions P. P. Boca. J.P. Bowen, J.I>Siddiqi pages 79-116 Springer-Verlag London 2010
[15] Christian Stary, S-BPM ONE - Scientific Research: 4th International Conference, S-BPM ONE 2012 vienna Austria, Proceedings Springer-Verlag Berlin Heidelberg 2012
[16] Maria S. Rukadikar. Data Structure and Algorithm, Shroff Publishers and Distributors Pvt. Ltd; 2011. ISBN-10: 9350235552