

|  |
| --- |
| FORMAL METHODS BCS 2213 |
| SITI NURBAIZURA BT SUKRI CB12042 |
| Section 01B |

ABSTRACT

With each new generation of aircraft, the requirement for digital avionics systems become increasingly complex and their development and validation consumes an ever increasing percentage of the total development cost of aircraft.

Formal methods apply theoretical computer science fundamentals to solve difficult problems especially in software engineering and hardware engineering. There is a large diversity of formal method for system, with a sound mathematical foundation and which support reasoning about properties of the systems.

Traditional methods of software verification rely on testing to verify behaviour and robustness. But testing can only show the presence of errors, not their absence. In contrast, formal methods use mathematics to prove certain facts and properties. But verification technique based on formal methods can conclusively prove certain attributes of software.

INTRODUCTION

Software is being used more and more in almost aspect of daily life e.g. in transportation, finance, health care, government and telecommunication and the reliability of such software critical for us, especially when failures may lead to catastrophes where people die or values/money are lost. For instance, when we go by train, it is correct.

As another example, when we use a home banking system to make a bank transaction over the internet, it is vital for us that the software controlling this is correct and secure such that transaction is executed as we have specified and nobody is able to misuse the data we are sending.

Such kind of software is rather complex and it is not an easy task to make it correct. Experience from software development projects also shows that software full of bugs leading to delays, cost overrun and usability problem.

LITERATURE REVIEW/BACKGROUND

Abstract State Machine (ASM) is a finite set of transition rules over abstract state. It is no restriction on the abstraction level, complexity nor means of function definition. ASM has been proved to and well founded in industrially viable method for the design and analysis of complex systems which is has been applied successfully to programming languages, protocols, embedded systems, architecture and requirement engineering.

The analysis covers both verification and validation, using mathematical reasoning or experimental simulation. As a sequence, a scientifically rigour to it, calls for a smooth integration into traditional hardware/software engineering procedure and notations, of multiple ways to achieve various degrees of certifiable system trust worthiness and quality assurance.

Among such approaches the ASM method is characterized by providing a simple practical framework where is a coherent and uniform way the system engineer can adopt a divides and conquer approach as will become clear in the following sections, not a single ingredient of the ASM method and the freedom it offers the practitioner to choose for each problem an appropriate combination of concepts, notations and techniques which is are interested by the framework in a coherent way as element as a uniform mathematical background.

HOW ABSTRACT STATE MACHINE WORKS

The ASM methods which bridge the gap between the two ends of system development which is the human understanding and formulation of real world problems and the deployment of their algorithmic solutions by code-executing machines on changing platforms.

There are 3 basic concepts for the method build

* ASM: a precise from pseudo-code generalizing finite state machines to operate over arbitrary date structures.
* Ground model which is a rigorous form of blueprints, serving as authoritative reference model for the design
* Refinement, a most general scheme for stepwise instantiations of model abstractions to concrete system elements, providing controllable links between the more and more detailed descriptions at the successive stages of system development.

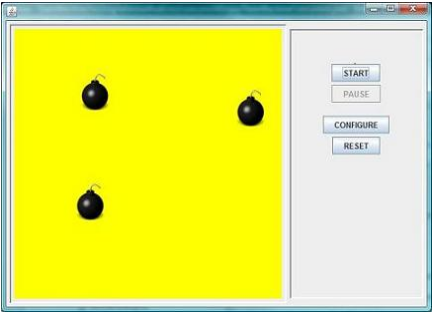
The original conception of ASMs, a single [agent](http://en.wikipedia.org/wiki/Software_agent) executes a program in a sequence of steps, possibly interacting with its environment. This notion was extended to capture [distributed computations](http://en.wikipedia.org/wiki/Distributed_computing), in which multiple agents execute their programs concurrently. Due to the algorithmic and mathematical nature of these three concepts, ASM models and their properties of interest can be analyzed using any rigorous form of [verification](http://en.wikipedia.org/wiki/Verification_and_Validation_(software)) or [validation](http://en.wikipedia.org/wiki/Verification_and_Validation_(software)).

The simple mathematical foundation of ASMs as FSMs working over arbitrary data types makes it easy for practitioners to understand and work with the concept. It also allows one to exploit for the ASM method the uniform conceptual and methodological framework of traditional mathematics, where one can consistently relate standard notions, techniques and notations to express any system features or views.5 Having as background for the ASM method not just one a priori chosen formal language and associated proof calculus, but the full body of usual mathematic.

SAMPLE/ IMPLEMENTATION OF ASM

**The Bouncing Bomb Application**

The Bouncing Bomb Application uses an FSM to manage the enabling/disabling of buttons in the right button panel. The effect is that allowable events are constrained according to the current application state. Each button initiates an action event when pressed; those events, when propagated to the FSM, result in state transitions. The set of actions (application events) and states are predefined (enumerated), and allowable state transitions are determined according to entries in a Map collection. The map defines what states are reachable from any other state, which indirectly determines the allowed button actions.



CONCLUSION

Various prototypical tools have been developed for designing and executing ASMs in the world. More advanced and industrially satisfactory tool support is needed for defining, simulating and visualizing, debugging, transforming (refining, implementing, where possible through code generation), analyzing (testing and verifying) ASMs. The tool environment should support to capture design knowledge in a rigorous and electronically available way. It should be linked as much as possible to establish design owns and exploit their achievements for an encouraging successful project. Together with an advanced tool environment a model and a proof theory of ASMs are needed which in particular define practical refinement principles, ideally together with corresponding proof schemes for a good start.

The codeless form of programming" ordered by ASMs helps porting application programs from one platform or language to another and could lead to fruitful applications for plug-and-play software technology. Paradigmatic and parameterized ASM components and (de)composition techniques for constructing them have to be defined and to be made available in libraries. ASMs should also be put to use to enhance current (mostly signature oriented) software architecture description techniques by adding semantically content to the structural definitions.

This is particularly promising at the levels of what in is called conceptual architecture and module interconnection architecture. Conceptual architecture refers to the ground model level where domain specials play a major role the module interconnection architecture level reflects implementation decisions which are independent of any particular programming language, such a the logical/functional decomposition, layers with allowable import/export relations, interface constraints.

REFERANCES

1. <http://web.mit.edu/16.35/www/lecturenotes/FormalMethods.pdf>
2. <http://staff.itee.uq.edu.au/pal/SVRC/tr98-25.pdf>
3. <http://www.eecs.umich.edu/gasm/intro.html>
4. <http://www.di.unipi.it/~boerger/ASMTutorialEtaps.html>
5. <http://research.microsoft.com/pubs/77452/issta02.pdf>
6. <http://www.di.unipi.it/~boerger/Papers/Methodology/BcsFacs07.pdf>
7. <http://en.wikipedia.org/wiki/Abstract_state_machines>
8. <http://clip.dia.fi.upm.es/~mcarro/Slides/Formal_Methods/Formal_Methods_Intro/formal-methods_6.pdf>
9. <http://en.wikipedia.org/wiki/Model_checkers>
10. <http://en.wikipedia.org/wiki/Formal_methods>
11. <https://elearning.fh-hagenberg.at//pluginfile.php/94507/mod_resource/content/0/ReadingMaterial/BcsFacs07.pdf>
12. <https://today.java.net/article/2009/09/06/finite-state-machine-supporting-concurrent-states>