The TASM Toolset: Specification, Simulation, and Formal Verification of Real-Time Systems
(Tool Paper)

Martin Ouimet and Kristina Lundqvist

Embedded Systems Laboratory
Massachusetts Institute of Technology
Cambridge, MA, 02139, USA
{mouimet,kristina}@mit.edu

Abstract. In this paper, we describe the features of the Timed Abstract State Machine toolset. The toolset implements the features of the Timed Abstract State Machine (TASM) language, a specification language for reactive real-time systems. The TASM language enables the specification of functional and non-functional properties using a unified language. The toolset incorporates features to create specifications, simulate specifications, and verify formal properties of specifications. Properties that can be verified using the toolset include completeness, consistency, worst-case execution time, and best-case execution time. The toolset is being developed as part of an architecture-based framework for embedded real-time system engineering. We describe how the features of the toolset were used successfully to model and analyze case studies from the aerospace and automotive communities.

1 Introduction

The Timed Abstract State Machine (TASM) specification language is a specification language for reactive real-time systems. The TASM language aims to capture the three key aspects of real-time system behavior, namely, functional behavior, timing behavior and resource consumption. TASM is based on the theory of Abstract State Machines (ASM), a method for system design that can be applied at various levels of abstraction [1]. The TASM language has formal semantics, which makes its meaning precise and enables executable specifications. The time semantics of the language revolve around the concept of durative actions.

The TASM toolset implements the features of the TASM language through three main components - an editor, an analyzer, and a simulator. The toolset can be used during the early phases of development to understand behavior before the system is built, or it can be used throughout the development of the system to guide implementation. The type of analysis that can be performed with the toolset include verifying completeness and consistency of the specification [2] and verifying timing properties of the specification such as the absence of deadlocks and Worst-Case Execution Time (WCET). The philosophy of the toolset is to reuse the state of the art in analytical engines to perform formal verification. The TASM toolset integrates the UPPAAL tool suite [3] to verify timing properties of TASM specifications and uses the SAT4J SAT Solver [4] to verify completeness and consistency of TASM specifications [5]. The TASM toolset serves as the basis of a specification framework for real-time system engineering [6].

2 The Timed Abstract State Machine (TASM) Language

The TASM language is based on the theory of Abstract State Machines (ASM), a method for high-level system design [1]. In the ASM formalism, behavior is specified as the computation steps of an abstract machine and the effect of the computation steps on the global state. The TASM language extends the ASM language by providing constructs and semantics for time and resource consumption. In the TASM language, time is attached to the steps of the abstract machine in such a way that a finite amount of time elapses before the effect of the computation step is reflected on the global state. The semantics of *durative* actions are used to reflect the reality that actions are typically not instantaneous. In a similar fashion, resource consumption is attached to durative steps to denote the resources used by the machine to complete the computation step. Listing 1 shows a sample rule of a TASM machine from the production cell system describing the action of the robot picking up a block from the press. The execution of the rule takes between 1 and 3 time units to complete and consumes exactly 2000 units of power.

Listing 1. TASM Rule Describing the Robot Picking up a Block from the Press

```
R1: Pickup from Press
{
  t     := [1, 3];
  power := 2000;

  if armbpos = atpress and armb = empty and press_block = available then
    press_block := notavailable;
    press       := empty;
    armb        := loaded;
}
```

The TASM language also contains facilities for hierarchical composition, parallel composition, and synchronization channels. In the TASM language, *completeness* is defined as a machine having a rule enabled for all classes of inputs [5]. *Consistency* is defined as a machine having no more than one rule enabled for all classes of inputs [5]. Furthermore, because actions are durative in TASM, execution time refers to the time that it takes to reach a certain reachable state from a start state. Worst-Case Execution Time (WCET) is the maximum amount of time that the machine will take to reach a state. Conversely, Best-Case Execution Time (BCET) is the minimum amount of time between any two states.

3 The TASM Toolset

The TASM toolset uses literate and graphical facilities to edit, simulate, and verify TASM specifications. The toolset includes facilities for creating and editing TASM specifications, through the TASM Editor. The editor enables the specification of functional and non-functional behavior, with standard facilities for syntax highlighting and syntax checking. By definition, TASM specifications are executable. The execution semantics of the TASM language have been defined in [7]. The TASM Simulator enables the graphical visualization of the dynamic behavior expressed in the specification in a step-by-step fashion. Because time and resources can be specified using intervals, that is, using a lower bound and an upper bound, the simulation can use different semantics for time durations and resource consumption. For example, a given simulation can use the worst-case time (upper bound) for all steps, to visualize the system behavior under the longest running times. Other options include best-case time, average-case time, and using a time randomly selected within the specified interval. The same semantics can be selected for the resource consumption behavior.

The TASM Analyzer is the component of the TASM toolset that performs analysis of specifications. The analyzer can be used to verify basic properties of TASM specifications such as consistency and completeness [2]. In the TASM language, completeness ensures that for all classes of monitored variable values, a rule will be enabled. Consistency ensures that for all classes of monitored variable values, one and only one rule is enabled. In other words, verifying consistency means verifying that the rules of a given machine are mutually exclusive. Both completeness and consistency are verified at the machine specification level. The analysis of completeness and consistency is achieved by translating machine rule guard expressions into a boolean formula in conjunctive normal form [5]. The boolean formula can then be verified for satisfiability using a SAT solver. The TASM toolset uses the SAT4J solver, an open source SAT solver [4]. The completeness and consistency problem is formulated in such a way that an incomplete or inconsistent specification leads to a satisfiable boolean formula. Formulating the problem this way ensures that the SAT solver can automatically generate a counterexample if the specification is inconsistent or incomplete.

The TASM analyzer is also used to verify execution time of TASM specifications. The execution time is verified by mapping TASM specifications to the timed automata formalism of UPPAAL. The UPPAAL tool suite is used in conjunction with an approach we call *iterative bounded liveness*, to verify BCET and WCET. The approach uses the bounded liveness temporal logic pattern [3] in an iterative fashion to converge to an upper bound and to a lower bound from an initial time obtained through reachability analysis. For TASM specifications, execution times are bounded. Since the reachability problem is decidable for timed automata, verifying the execution times of TASM specifications is guaranteed to converge. The toolset is available, free of charge, from the TASM web site (http://esl.mit.edu/tasm).

4 Case Studies

The TASM language and toolset have been used to model and analyze three case studies. The toolset has been used to model an Electronic Throttle Controller (ETC) and to analyze the completeness and consistency of the mode switching logic of the controller [8]. The production cell, partially illustrated in Listing 1, was modeled and analyzed in the toolset. The production cell was analyzed to measure the minimum amount of time for the system to process 5 blocks. Using the model and BCET analysis, the optimal solution to process 5 blocks was automatically derived. The toolset was also used to model the Timeliner System, a scripting environment currently in use on the international space station. The model was used to analyze the WCET of one pass of the Timeliner system. The Timeliner system shares processor usage with other tasks using a fixed timeslice scenario. The execution time was analyzed to ensure that the assigned timeslice is adequate but not overly estimated, to ensure optimal processor usage. Future case studies will include a modular redundant avionics system, modeled and analyzed to understand the end-to-end latency of the system.

5 Conclusion and Future Work

The toolset and language have been used successfully to model and analyze embedded real-time systems as found in the avionics and automotive communities. Using specifications expressed in the TASM language, the toolset can verify properties of specifications such as completeness and consistency. The analysis is performed by translating the specification and using existing solvers. For completeness and consistency, this is achieved through a translation to boolean formulas and using the SAT4J SAT solver to automatically verify the property. Furthermore, the execution times of specifications can be analyzed using a translation of TASM specifications to UPPAAL's timed automata.

Future work on the toolset will investigate the use of theorem provers to verify properties of the models that cannot be handled because of state explosion problems. Furthermore, the toolset will be used to generate test cases based on TASM specifications. This will most likely be achieved by reusing the translation to boolean formulas and the translation to timed automata and using established algorithms to to generate test cases.

References

1. Börger, E.: The Origins and the Development of the ASM Method for High Level System Design and Analysis. Journal of Computer Science, vol. 8(5) (2001)
2. Heimdahl, M.P.E., Leveson, N.G.: Completeness and Consistency in Hierarchical State-Based Requirements. Software Engineering 22(6), 363–377 (1996)
3. Behrmann, G., David, A., Larsen, K.G.: A Tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, vol. 3185, Springer, Heidelberg (2004)
4. Leberre, D.: SAT4J: A Satisfiability Library for Java. presentation available from `http://sat4j.objectweb.com`
5. Ouimet, M., Lundqvist, K.: Automated Verification of Completeness and Consistency of Abstract State Machine Specifications using a SAT Solver. In: Proceedings of the 3rd International Workshop on Model-Based Testing (MBT '07), Satellite Workshop of ETAPS '07 (April 2007)
6. Ouimet, M., Lundqvist, K.: The Hi-Five Framework and the Timed Abstract State Machine Language. In: Proceedings of the 27th IEEE Real-Time Systems Symposium - Work in Progress Session, December 2006, IEEE Computer Society Press, Los Alamitos (2006)

7. Ouimet, M., Lundqvist, K.: The Timed Abstract State Machine Language: An Executable Specification Language for Reactive Real-Time Systems. In: Proceedings of the 15th International Conference on Real-Time and Network Systems (RTNS '07) (March (2007)
8. Ouimet, M., Berteau, G., Lundqvist, K.: Modeling an Electronic Throttle Controller using the Timed Abstract State Machine Language and Toolset. In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, Springer, Heidelberg (2007)