An Object-Oriented Algebraic Steam-Boiler Control Specification

Extended abstract

Peter Csaba Ölveczky[1], Piotr Kosiuczenko[2,3], and Martin Wirsing[2]

[1] Dept. of Informatics
University of Bergen, Norway

[2] Institut für Informatik
Ludwig-Maximilians-Universität München, Germany

[3] Instytut Matematyki
Politechnika Warszawska, Warszawa, Poland

1 Introduction

The aim of this paper is to use the steam-boiler problem [1] to illustrate how specifications for hybrid systems can be given in an object-oriented and algebraic style. The solution is written in Timed Maude. Timed Maude is a specification language under development where the static parts of the specified system are described by equational specifications, whereas the behavior of a process is described by timed term rewriting. Timed Maude is based on Meseguer's Maude language [5], and its underlying logic is timed rewriting logic, an extension of rewriting logic to deal with hard real-time and hybrid systems.

The steam-boiler control specification problem has been proposed as a challenge for different methods for real-time, safety-critical, and fault-tolerant systems. Such a system consists of a *plant*, a *control program* and some *transmission medium* through which plant and program communicate. The problem is to give a formal specification of the control program and to verify that it behaves in a certain way when composed with its *environment*, the plant and the transmission medium.

As the informal specification of Abrial this paper focuses on the control program which is designed as the parallel composition of several objects which communicate using messages. The transmission of such internal messages is assumed to be instantaneous whereas the communication with the environment can be time consuming.

The extended abstract is organized as follows: Section 2 presents a short introduction to Timed Rewriting Logic and Timed Maude. Section 3 gives an outline of our solution of the steam-boiler specification problem. Section 4 contains some techniques used for validating the specification. Section 5 gives a short resumé of our approach.

2 Timed Rewriting Logic and Timed Maude

The underlying logic of our specification is *timed rewriting logic* (TRL) [2]. TRL extends algebraic specification techniques and tools to handle hybrid systems like the steam-boiler. In TRL, the static parts of a system are described by equations while the transitions are described by timed rewrite rules. TRL has well-defined initial dynamic algebra semantics.

Timed rewriting logic (TRL) is based on Meseguer's rewriting logic [4] and allows us to reason about time elapse in real-time systems. In TRL, a time stamp is added to each rewrite step. Timed rewriting logic is a logic about *change* in real-time systems. Processes are considered as terms and proofs as behaviors of a process. Basically, $t \xrightarrow{r} t'$ means that a system could rewrite from state t to state t' in time r.

Our specification is written in the real-time specification language Timed Maude (see also [3]). Timed Maude is based on TRL and Meseguer's language Maude [4, 5], from which Timed Maude borrows its object-oriented features. As in Maude, a Timed Maude program makes computational progress by rewriting its global state, its *configuration*, which is a multiset of objects and messages. In Timed Maude, a configuration can not proceed in time if it contains messages, This feature forces messages to be read as soon as they are received, and will be used extensively in our solution.

Rewriting in Timed Maude is nothing but deduction in TRL, which means that a Timed Maude specification defines a logical theory that, due to the operational nature of the logic, at the same time can be seen as being half way between a more abstract, non-executable specification and an actual implementation. We believe that the order-sorted object-oriented approach using equations and timed rewrite rules yields a specification which is easily understandable for the customer, yet detailed enough to point out loose ends in the informal specification.

3 Outline of the Solution

Our solution focuses on the control program for which an operational specification in an object-oriented algebraic style is given. The solution also provides a specification of the initial behavior of the control program. For the environment only a simple specification will be given which can be used for validation and simulation issues. The environment and the control program communicate by exchanging messages. We loosen the restriction in [1] that all messages from the environment in a round are received simultaneously. This will instead be a special case of environment behavior. The fact that messages from the environment can arrive at arbitrary times in a steam-boiler cycle, together with the assumption that the time a message is received reflects the time the message-triggering action took place in the environment, turns a merely reactive steam-boiler system into a really hybrid system.

To make the problem of specifying the control program (which can receive more than 2^{21} different message combinations in each round) more manageable

and to increase parallelism, the control program is defined as the parallel composition of different objects. In general, each physical unit is represented by a corresponding object in the control program. Such an object keeps track of what the control program knows about the corresponding physical unit (based on previously received messages), but does not model the *physical* state of the unit, of which the control program knows virtually nothing.

With the exception of two rules specifying the overall dynamics of the system, each object is specified independently of the other objects. In this way we obtain a very modular specification which also allows for a high degree of concurrency. However, it also means that whenever an object needs to know the value of some other object's attribute, this must be sent (sometimes upon request) by the other object.

The steam-boiler system proceeds in cycles of length Δt. Messages from the environment are assumed to be received in the open time interval $(n\Delta t, (n+1)\Delta t)$ for $n = 0, 1, 2, ...$

During one such time interval, the system alternates between the following two behaviors:

- While no new messages are received from the environment, the system evolves according to the dynamics of the system and updates the minimal and maximal expected water and steam levels. This is due to the fact that in our setting messages from the environment can be received at any time, and that a level value message received at time t is supposed to indicate the level at time t, not the level at the end of the round.
- Whenever a message from the environment is sent, the system treats this immediately (since time cannot proceed when a configuration contains a received but unread message). A message is read by the appropriate object (e. g. the reported water level is read by the water object), which then
  - updates some of its attributes (e. g. if the received water level is within the current value of the estimated minimal and maximal water level attributes, these attributes are updated accordingly) and/or
  - sends messages to other control program objects.

At the end of each cycle (time $n\Delta t$), timers count down to zero and thereby blocks the system from proceeding in time. Then

- the objects check if all messages that must arrive from the environment in each cycle have arrived,
- the system computes which messages to send to the environment, and
- when an object has finished all computations for this cycle, it resets its timer to Δt. When all timers are reset, the system can again proceed in time and a new cycle can begin.

4 Validation Issues

In this section some validation techniques are outlined.

To give a flavor of reasoning within timed rewriting logic, we prove that if the control program in initializing mode receives a *steam_boiler_waiting* message, and the steam level it different from 0, then the configuration *could* rewrite to a configuration where the $mode(emergency)$ message is sent.

To validate the specification further, a number of properties that the system is expected to satisfy could be formalized and proven formally. The formalism of timed rewriting logic only allows for proving $t \xrightarrow{r} t'$, but does not allow to prove $t \not\xrightarrow{r} t''$, absence of deadlock, etc., within the logic itself.

As an example of reasoning about the system outside the logic, it is shown, by using techniques to prove termination of sequential order-sorted rewriting that Zeno behavior (in this case: an infinite number of zero-time state changes in a row) cannot occur.

5 Concluding Remarks

In this paper we have presented a Timed Maude solution to the specification problem posed in [1]. The specification is operational; it describes how to design a steam-boiler control program. Timed Maude is a specification language which extends algebraic and object-oriented specification techniques to handle real-time systems.

We believe that the fact that the solution consists of equations and timed rewrite rules, and that objects can be specified almost independently of each other, makes the specification easy to read and understand and enhances confidence in that it provides a faithful formalization of the informal specification.

References

1. J. R. Abrial. Steam-boiler control specification problem. Distributed to the participants of a Dagstuhl seminar on "Methods for Semantics and Specification". Available via http://www.informatik.uni-kiel.de/~procos/dag9523/dag9523.html, 1994. See also "Additional Information Concerning the Physical Behaviour of the Steam-boiler" at the same address.
2. P. Kosiuczenko and M. Wirsing. Timed rewriting logic, 1995. Working material for the 1995 Marktoberdorf International Summer School "Logic of Computation".
3. P. Kosiuczenko and M. Wirsing. Timed rewriting logic with an application to object-oriented specification. To appear in *Science of Computer Programming*, 1996.
4. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.
5. J. Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In Gul Agha, Peter Wegner, and Akinori Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT Press, 1993.

This article was processed using the LaTeX macro package with LLNCS style