Formal Model Engineering for Embedded Systems Using Real-Time Maude

Peter Csaba Ölveczky
Department of Informatics, University of Oslo

This paper motivates why Real-Time Maude should be well suited to provide a formal semantics and formal analysis capabilities to modeling languages for embedded systems. One can then use the code generation facilities of the tools for the modeling languages to automatically synthesize Real-Time Maude verification models from design models, enabling a formal model engineering process that combines the convenience of modeling using an informal but intuitive modeling language with formal verification. We give a brief overview six fairly different modeling formalisms for which Real-Time Maude has provided the formal semantics and (possibly) formal analysis. These models include behavioral subsets of the avionics modeling standard AADL, Ptolemy II discrete-event models, two EMF-based timed model transformation systems, and a modeling language for handset software.

1 Introduction

It is now acknowledged that one of the most promising approaches to inject formal methods into industrial model-driven engineering is to endow the domain-specific modeling languages used in industry with formal verification capabilities. This can be achieved by defining the formal semantics of the informal modeling language, and then automatically synthesize a *formal verification model* from the informal *design model*. This enables a *formal model engineering* process that combines the convenience of modeling using an informal but intuitive modeling language with formal verification. Furthermore, the synthesis of the verification model can take advantage of the code generation infrastructure provided by a number of advanced modeling tools to support the generation of deployment code from design models; this infrastructure can also be used to generate a formal model.

For the formal model engineering process to be useful in practice, the target formal language and tool must not only be expressive enough to define the formal semantics of the informal language and provide automated simulation and model checking capabilities, but should also

1. allow the users to define formal analysis commands without understanding the formal language or the formal representation of their models; and
2. provide formal analysis results, such as counterexamples in temporal logic model checking, that the user can easily understand.

Formal model engineering is crucial for real-time embedded systems (RTESs)—such as automotive, avionics, and medical systems—that are hard to design correctly, since subtle timing aspects impact system functionality, yet are safety-critical systems whose failures may cause great damage to persons and/or valuable assets. There exist a number of formal analysis tools for real-time systems (see [29] for a survey). However, there is a significant gap between the formalisms of these tools, such as timed automata [2] or timed Petri nets [9, 28], that sacrifice expressiveness for decidability, and the expressiveness of modeling languages for industrial RTESs, that were, after all, designed for modeling convenience.

In contrast to such formal tools, Real-Time Maude [22]—which extends the rewriting-logic-based Maude system [10] to support the formal specification, simulation, and model checking of real-time

Francisco Durán and Vlad Rusu (Eds.):
Second International Workshop on Algebraic Methods
in Model-Based Software Engineering 2011 (AMMSE'11)
EPTCS 56, 2011, pp. 3–13, doi:10.4204/EPTCS.56.1

systems—emphasizes expressiveness and ease of specification over algorithmic decidability of key properties. In Real-Time Maude, the state space and the functional properties of the system are defined as an equational specification, the instantaneous local transitions are modeled as rewrite rules, and time advance is modeled by tick rewrite rules. Real-Time Maude is particularly suitable for modeling real-time systems in an object-oriented style. Because of its expressiveness, Real-Time Maude has been successfully applied to a wide range of advanced state-of-the-art systems that are beyond the pale of timed automata, including wireless sensor networks algorithms [13, 24], scheduling algorithms that require unbounded queues [21], and large multicast protocols [23, 15].

Real-Time Maude's natural model of time, together with its expressiveness, should make it a suitable semantic framework for modeling languages for RTESs; such languages then also get the following formal analysis capabilities for free:

- simulation;
- reachability analysis;
- linear temporal logic (LTL) model checking for for untimed LTL properties, as well as time-bounded LTL model checking for analyzing systems with an infinite reachable state space; and
- TCTL$_{\leq,\geq}$ model checking of *timed* CTL formulas [14].

It is also worth mentioning that the use of both equations (to define the functional aspects of the system) and rewrite rules (to model its concurrent transitions) makes model checking significantly more efficient than if deterministic behaviors were also encoded as rules, since only rewrite rules contribute to the reachable state space.

The possibility to equationally define *parametric* atomic state propositions (and "state patterns" for reachability analysis) in Real-Time Maude allows us to predefine a useful set of parametric state propositions in the Real-Time Maude interpreter of a language. These propositions make it easy for the user to define his/her search and LTL model checking commands without having to understand Real-Time Maude or the formal representation of his/her model, as illustrated in this paper.

A key requirement to (i) understand the results of Real-Time Maude analyses, and (ii) being able to map them back into the original modeling formalism is to have, respectively, a small *representational distance* between the original models and their formal counterparts, and a one-to-one correspondence between these models. Since hierarchical composition and encapsulation play key roles in industrial modeling languages, the possibility of defining *hierarchical* objects enable us to achieve both goals.

In this extended abstract, I give an overview of six fairly different modeling languages for RTESs that have a Real-Time Maude semantics, and briefly explain how they have been endowed with Real-Time Maude, or in some cases Maude, simulation and model checking capabilities. Papers on the semantics of each modeling have been published elsewhere [1, 3, 4, 8, 19, 25].

2 Behavioral AADL

The *Architecture Analysis & Design Language* (AADL) [27] is an industrial standard used in avionics, aerospace, automotive, medical devices, and robotics communities to describe a performance-critical embedded real-time system as an assembly of software components mapped onto an execution platform.

In joint work with José Meseguer, I have defined the Real-Time Maude semantics of a subset of the *software components* of AADL. This subset defines the architectural and behavioral specification of a system as a set of hierarchical components with ports and connections, and with the behaviors defined by Turing-complete transitions systems. The references [19, 20] explain the Real-Time Maude semantics

for the targeted fragment of AADL in detail. In particular, the semantics of a component-based language can naturally be defined in an object-oriented style, where each component instance is modeled as an object, and where the hierarchical structure of components is reflected in the nested structure of objects.

Together with Artur Boronat, we have developed the *AADL2Maude* plug-in for the Open Source AADL Tool Environment (OSATE), which is a set of Eclipse plug-ins. *AADL2Maude* uses OSATE's code generation facility to automatically generate Real-Time Maude specifications from AADL models. However, the Real-Time Maude analysis itself is not yet integrated into OSATE.

To allow the user to conveniently define his/her search and model checking commands without knowing Real-Time Maude or the Real-Time Maude representation of AADL models, we have defined some useful functions and parametrized state propositions. For example, the term

`value of` *v* `in component` *fullComponentName* `in` *globalComponent*

gives the value of the state variable *v* in the thread identified by the full name *fullComponentName* in the system *globalComponent*. Likewise,

`location of component` *fullComponentName* `in` *globalComponent*

gives the current location/state in the transition system in the given thread.

We have applied the *AADL2Maude* code generator and Real-Time Maude to formally analyze an AADL model, developed by Min Young Nam and Lui Sha at UIUC, of a simple network of medical devices, consisting of a *controller*, a *ventilator machine* that assists a patient's breathing during surgery, and an *X-ray* device [18]. Whenever a button is pushed to take an X-ray, the ventilator machine should pause for two seconds, starting one second after the button is pushed, and the X-ray should be taken after two seconds. The initial state is `{MAIN system Wholesys . impl}`, and `MAIN -> Xray -> xmPr -> xmTh` denotes the full name of the X-ray machine thread. The ventilator machine must be pausing when an X-ray is being taken, so that the X-ray is not blurred. The following search command checks whether an *undesired* state, where the X-ray thread `xmTh` is in local state `xray` while the ventilator thread `vmTh` is *not* in state `paused`, can be reached from the initial state (the unexpected result shows a concrete unsafe state that can be reached from the initial state):

```
Maude> (utsearch [1]
         {MAIN system Wholesys . impl} =>* {C:Configuration}
          such that
           ((location of component (MAIN -> Xray -> xmPr -> xmTh)
               in C:Configuration) == xray
            and (location of component (MAIN -> Ventilator -> vmPr -> vmTh)
                   in C:Configuration) =/= paused) .)

Solution 1    C:Configuration --> ...
```

For LTL and timed CTL model checking purposes, our tool has pre-defined useful parametric atomic propositions, such as *full thread name* `@` *location*, which holds when the thread is in state *location*, and `value of` *variable* `in component` *full thread name* `is` *value*, which holds if the current value of the local transition system variable *variable* in the given thread equals *value*.

We can use time-bounded LTL model checking to verify that an X-ray must be taken within three seconds of the start of the system (this command returned a counter-example revealing a subtle and previously unknown design error):

```
Maude> (mc {MAIN system Wholesys . impl} |=t
           <> ((MAIN -> Xray -> xmPr -> xmTh) @ xray) in time <= 3 .)

Result ModelCheckResult :  counterexample( ... )
```

3 Synchronous AADL

In a number of systems targeted by AADL, such as integrated modular avionics systems and distributed control systems in motor vehicles, the system design is essentially a *synchronous design* that must realized in an asynchronous distributed setting. The key idea of the *PALS architectural pattern* [16, 17] is to reduce the design, verification, and implementation of a distributed real-time system to that of its much simpler synchronous version, provided that the network infrastructure guarantees bounds on the messaging delays and the skews of the local clocks. For a synchronous design *SD* and network bounds Γ, we then have a semantically equivalent asynchronous distributed design $PALS(SD, \Gamma)$. Not only is this important from a system design perspective, but it also makes model checking verification feasible. For example, in [16] we show that for the avionics system mentioned below, the synchronous design model has 185 reachable states and can be model checked in less than a second in Maude, whereas—even restricted to perfect local clocks and message delays 0 and 1—the corresponding asynchronous model has more than 3 million reachable states and cannot be model checked in reasonable time.

To allow modelers to take advantage of PALS while using AADL, I have, in joint work with Kyungmin Bae, José Meseguer, and Abdullah Al-Nayeem, identified a "synchronous subset" of behavioral AADL, called *Synchronous AADL*, that is suitable to define the synchronous PALS designs in AADL [3].

To support formal model engineering, we have integrated the Real-Time Maude analysis of Synchronous AADL models into OSATE. The *SynchAADL2Maude* tool is an OSATE plug-in that uses OSATE's model traversal facility to both support the generation of a Real-Time Maude model from an Synchronous AADL instance model, and for analyzing the synthesized Real-Time Maude model from *within OSATE*. *SynchAADL2Maude* inherits the convenient predefined atomic propositions from our AADL analysis library, and also allows the modeler to define new formulas and LTL model checking commands in XML within OSATE.

We have used *SynchAADL2Maude* to verify a Synchronous AADL model of an avionics system based on a specification by Steve Miller and Darren Cofer at Rockwell-Collins [17]. Figure 1 shows the *SynchAADL2Maude* window for this example. Real-Time Maude code generation and model checking are performed by clicking on the `Code Generation` button and the `Do Verification` button, respectively. The LTL properties that the avionics system should satisfy have been entered into the tool, and are shown in the "AADL Property Requirement" table. The `Do Verification` button has been clicked and the results of the model checking are shown in the "Maude Console."

The properties to be verified are managed by the associated XML property file. For example, to add an LTL model checking command to verify a property `R1` defined as the LTL formula below, we just add the following `command` tag to the property file:

```
<command>
 <name>R1</name>
  <value type = "ltl">
   [] (noChangeAssumptionNextState
        -> O (agreeOnActiveSide \/ O (noSideFailed -> agreeOnActiveSide))) .
 </value>
</command>
```

New formulas can be defined in the property file using the `definition` tag. For example, a new constant `side1Failed`, can be defined as follows:

```
<definition>
 <name>side1Failed</name>
  <value>
```

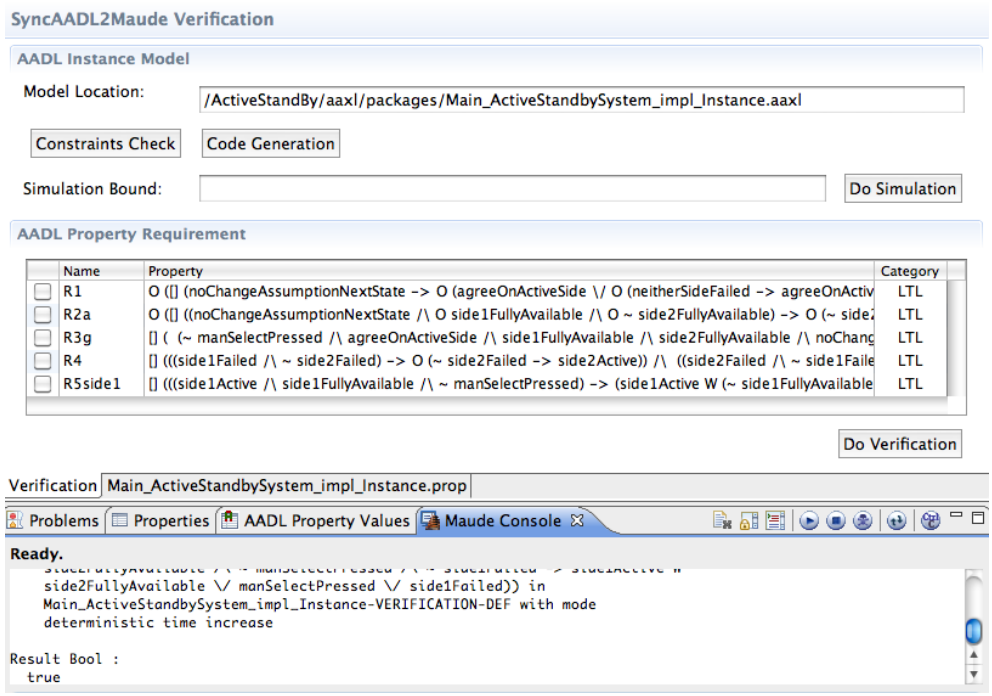


Figure 1: SynchAADL2Maude window in OSATE.

```
    value of side1Failed in component
      (MAIN -> sideOne -> sideProcess -> sideThread) is true
  </value>
</definition>
```

4 Ptolemy II Discrete-Event Models

Ptolemy II [11] is a well-established modeling and simulation tool used in industry that provides a powerful yet intuitive graphical modeling language. A Ptolemy II model consists of a set of interconnected *actors* with *input ports* and *output ports*, where communication channels pass *events* from one port to another. Such a model can be encapsulated as a *composite* actor, which may also have input and output ports. In Ptolemy II, real-time systems are modeled as *discrete-event* (DE) models. Like many graphical modeling languages, Ptolemy II DE models lack formal verification capabilities.

In each iteration of a DE model, all components with input execute *synchronously*. Since connections are instantaneous and the components execute in lock-step, we must compute the *fixed point* of the input for each component in the round before its execution; this input comes from the output of another actor's execution in the same synchronous round. Formalizing the Ptolemy II DE modeling language is fairly challenging as it combines a synchronous fixed-point semantics with time and hierarchical models, as well as a powerful expression language.

Real-Time Maude code generation and verification has been integrated into Ptolemy II by Kyungmin Bae, so that a large subset of Ptolemy II DE models can be verified from within Ptolemy II. The paper [4] explains the Real-Time Maude semantics and formal analysis support for Ptolemy II DE models in detail.

Figure 2 shows a hierarchical Ptolemy II model of a fault-tolerant traffic light system at a pedestrian

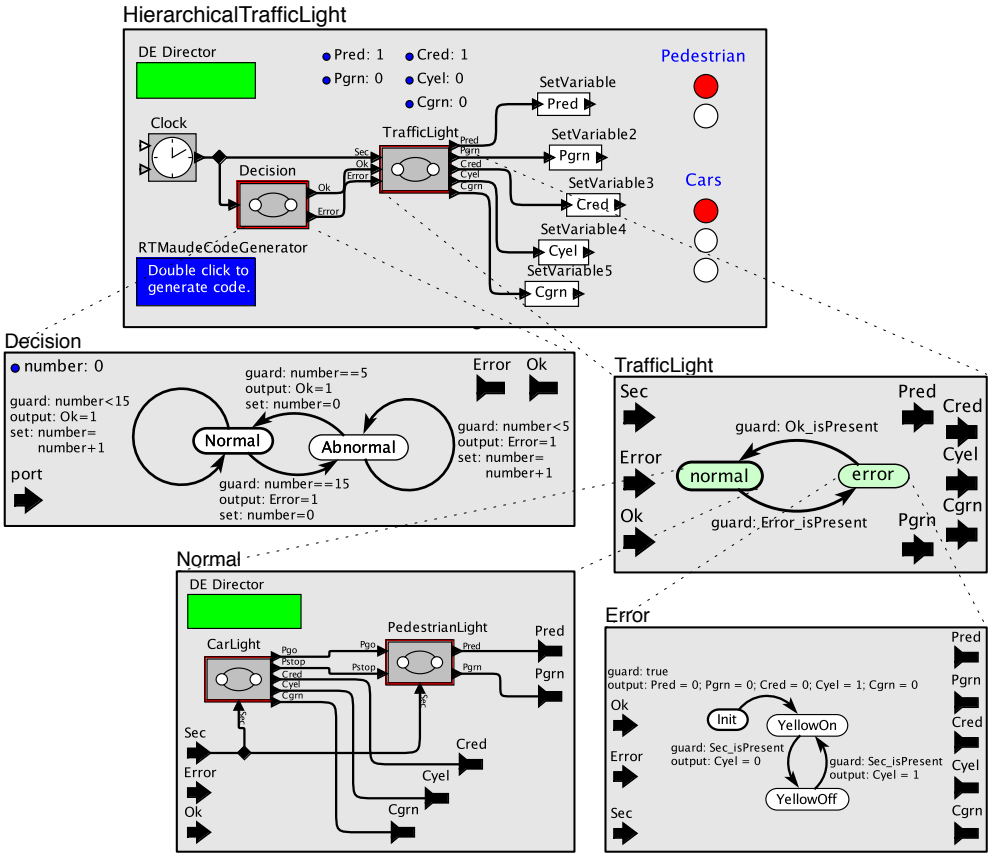


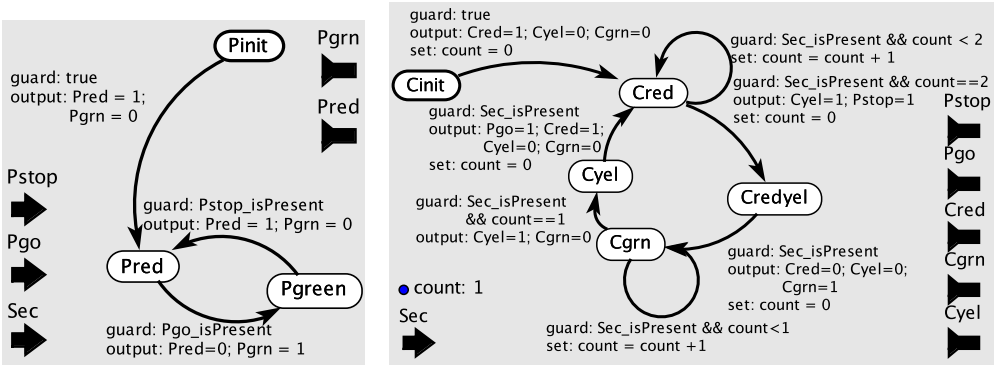
Figure 2: A hierarchical fault-tolerant traffic light system in Ptolemy II.

crossing, consisting of one car light and one pedestrian light. Each light is represented by a set of *set variable* actors (`Pred` and `Pgrn` represent the pedestrian light, and `Cred`, `Cyel` and `Cgrn` represent the car light). A light is *on* iff the corresponding variable has the value 1. The state machine actor `Decision` "generates" failures and repairs by alternating between staying in location `Normal` for 15 time units and staying in location `Abnormal` for 5 time units. Whenever the model operates in `error` mode, all lights are turned off, except for the yellow light of the car light, which is blinking. We refer to [4] for a thorough explanation of the model.

We have used Ptolemy II's code generation infrastructure to integrate both the synthesis of a Real-Time Maude model from a Ptolemy II design model as well as Real-Time Maude model checking of the synthesized model into Ptolemy II itself. When the blue `RTMaudeCodeGenerator` button in a Ptolemy II DE model is double-clicked, Ptolemy II opens a dialog window (shown in Fig. 4) which allows the user to start code generation and to give model checking commands to formally analyze the generated model.

We have also predefined in our model checker useful atomic propositions similar to those in our AADL model checker. For example, the proposition

$$actorId \quad | \quad var_1 = value_1, \ldots, var_n = value_n$$



(a) `Pedestrianlight` (b) `Carlight`

Figure 3: The state machine actors for pedestrian lights and car lights

holds in a state if the value of the parameter var_i of the actor *actorId* equals $value_i$ for each $1 \leq i \leq n$, where *actorId* is the *global actor identifier* of a given actor. Similarly, *actorId* `|` `port` *p* `is` *value* and *actorId* `|` `port` *p* `is` *status* hold if, respectively, the port *p* of actor *actorId* has the value *value* and status *status*. For state machine actors, the proposition *actorId* `@` *location* is satisfied if and only if the actor *actorId* is in "local state" *location*.

In the traffic light system, the following *timed* CTL property states that the car light will turn yellow, *and only yellow*, within 1 time unit of a failure:

```
AG (('HierarchicalTrafficLight . 'Decision | port 'Error is present)
   => AF[<= 1] ('HierarchicalTrafficLight | 'Cyel = 1, 'Cgrn = 0, 'Cred = 0))
```

Model checking this property revealed a previously unknown error: after a failure, the car light may show red or green in addition to blinking yellow.

5 Timed Model Transformations in MOMENT2

The MOMENT2 [6, 7] formal model transformation framework is based on a formalization of MOF meta-models in rewriting logic. The static semantics of a system is given as a class diagram (or meta-model) describing the set of valid system states (or models) that are represented as object diagrams, and the dynamics of a system is defined as an *in-place model transformation*. In MOMENT2, a model transformation is defined as a set of production rules. Each such rule

```
rl l { nac dl nacl { NAC } such that cond ;...
       lhs { dl { L } };   rhs { dl { R } };   when cond; ... }
```

has a left-hand side *L*, a right-hand side *R*, a set of (possibly conditional) negative application conditions *NAC*, and a condition with the *when* clause. *L*, *R*, and *NAC* contain model patterns, where nodes are object patterns and unidirectional edges are references between objects. MOMENT2 provides a range of analysis methods in Maude, including a checking whether a model conforms to its meta-model as well as model checking of model transformations.

In joint work with Artur Boronat, I have extended MOMENT2 to add timed behaviors to model transformations by providing a small and simple but useful set of built-in types for defining *clocks*, *timers*, and what we call *timed values*, which are clocks that increase with a given rate [8]. As a consequence,

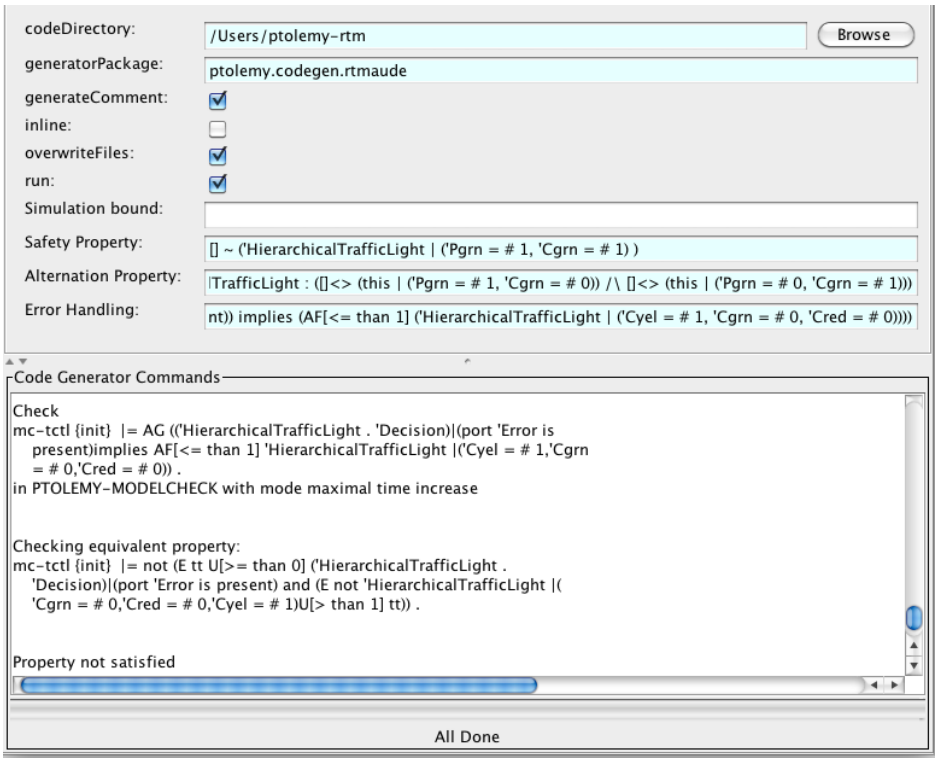


Figure 4: Dialog window for the hierarchical traffic light code generation.

a software engineer can use MDA standards and Eclipse Modeling Framework (EMF) technology to formally analyze RTESs.

This approach, where timed constructs have to be added to the original model to express timed behaviors requires changing the structural design of the application. However, using MOMENT2's support for *multi-model* transformations, one can also define a timed system in a *non-intrusive* way, in which the user-defined meta-model of the system is *not* modified. See [8] for details.

The Real-Time Maude semantics of timed model transformations is a fairly straight-forward extension of the rewriting logic semantics of model transformations given in [5]. Although MOMENT2 currently uses Maude as its execution engine, we can also easily perform Real-Time Maude inspired *time-bounded* analyses by just adding a single unconnected `Timer`—whose initial value is the time bound—to the initial state. When this timer expires, time will not advance further in the system, since no rule resets or turns off the timer. This ability to perform time-bounded analysis makes (time-bounded) LTL model checking analysis possible for systems with an infinite reachable state space, that can otherwise not be subjected to LTL model checking.

6 Domain-Specific Visual Languages in e-Motions

The e-Motions model transformation framework [26] for domain-specific visual languages is also based on EMF, with Ecore meta-models defining the abstract syntax of the language, and where the concrete syntax maps elements of the abstract syntax onto graphical objects. Although specification of system behaviors is based on in-place model transformations, the approach to timed behaviors is different from

the one taken in MOMENT2. Instead of adding external "timed constructs," e-Motions tries to make the definition of timed model transformations as intuitive and high-level as possible by providing different kinds of *timed model transformation rules*.

A model transformation rule is equipped with a time interval that defines the range of the possible *durations* of the rule, which is applied eagerly. A rule can, however, be declared to be *soft*, which means that it is not triggered eagerly, and/or may be declared to be *periodic*, in which case it is triggered periodically as long it is enabled. In addition to atomic rules, there are also *ongoing rules* that model continuous actions. These are powerful high-level constructs that typically imply that many different timed rules are being applied simultaneously to an object.

e-Motions has a Real-Time Maude semantics. e-Motions models can be simulated visually in the e-Motions tool, which is available as an Eclipse plug-in. For reachability and LTL model checking analyses, however, the tool generates the corresponding Maude model, and the analysis has to be done at the Maude level. However, the operator `<<_;_>>` is useful to define analysis commands without having to know the Maude representation of an e-Motions model. The term << *ocl-expr* ; *model* >> evaluates the OCL expression *ocl-expr* in the model *model*. Therefore, to search for a model reachable from an initial model `myModel` that satisfies the OCL expression `ocl-expr`, we use the Maude command

```
search [1] init(myModel) =>* {MODEL:@Model} in time T:Time
  such that << ocl-expr ; MODEL:@Model >> .
```

7 A Modeling Language for Handset Software

In [1], Musab AlTurki and researchers at DOCOMO USA Labs describe a simple but powerful specification language, called $\mathscr{L}$, that is claimed to be well suited for describing a spectrum of behaviors of various software systems. The language provides flexible SDL-inspired timing constructs that yield a more expressive language for timed behaviors than Erlang [12], since some nested timing patterns, which can be expressed in $\mathscr{L}$, are not expressible in Erlang [1].

The language has an expression language, imperative features for describing sequential computations, and asynchronously communicating processes that can be dynamically created or destroyed. It is worth remarking that just the dynamic process creation places $\mathscr{L}$ outside the class of systems that can be represented as timed automata; so does its expression language and imperative features which make $\mathscr{L}$ Turing-complete.

The semantics of of $\mathscr{L}$ is defined in Real-Time Maude, and Real-Time Maude therefore also provides a prototype analysis tool for $\mathscr{L}$. However, there does not seem to be any support for automatically translating $\mathscr{L}$ specifications into Real-Time Maude models.

8 Concluding Remarks

This paper has motivated the use of Real-Time Maude as a target formalism and tool that should be well suited to support the formal model engineering of real-time embedded systems, and has given a brief overview of the use of Real-Time Maude for such purposes.

Much work remains as always, including covering other languages and larger subsets of the languages mentioned in the paper. Furthermore, although we can predefine useful propositions so that the user can easily define his/her own queries, and although there is a one-to-one correspondance between the informal and the formal model, results from the formal analyses should still be mapped back into the original formalism.

Acknowledgments. I would like to thank the organizers of AMMSE 2011 for inviting me to give a survey talk about the use of Real-Time Maude to support the formal model engineering of embedded systems. Much of the work reported in this paper is joint work. I am particularly grateful to José Meseguer for initiating, encouraging, and collaborating on most of the research presented in this paper in which I have been involved. I am also grateful to Kyungmin Bae for our joint work on Synchronous AADL and Ptolemy II, and to other collaborators on the projects summarized in this paper, including Abdullah Al-Nayeem, Artur Boronat, Edward Lee, Steve Miller, Min Young Nam, Lui Sha, and Stavros Tripakis. I also gratefully acknowledge financial support from The Research Council of Norway.

References

[1] M. AlTurki, D. Dhurjati, D. Yu, A. Chander & H. Inamura (2009): *Formal Specification and Analysis of Timing Properties in Software Systems*. In: *FASE'09*, *Lecture Notes in Computer Science* 5503, Springer, doi:10.1007/978-3-642-00593-0_18.

[2] R. Alur & D. L. Dill (1994): *A theory of timed automata*. *Theoretical Computer Science* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[3] K. Bae, P. C. Ölveczky, A. Al-Nayeem & J. Meseguer (2011): *Synchronous AADL and its Formal Analysis in Real-Time Maude*. In: *Proc. ICFEM'11*, Lecture Notes in Computer Science. To appear.

[4] K. Bae, P. C. Ölveczky, T. H. Feng, E. A. Lee & S. Tripakis (2011): *Verifying hierarchical Ptolemy II discrete-event models using Real-Time Maude*. *Science of Computer Programming* doi:10.1016/j.scico.2010.10.002. To appear.

[5] A. Boronat, R. Heckel & J. Meseguer (2009): *Rewriting Logic Semantics and Verification of Model Transformations*. In: *Proc. FASE'09*, *Lecture Notes in Computer Science* 5503, Springer, doi:10.1007/978-3-642-00593-0_2.

[6] A. Boronat & J. Meseguer (2008): *An Algebraic Semantics for MOF*. In: *Proc. FASE'08*, *Lecture Notes in Computer Science* 4961, Springer, doi:10.1007/978-3-540-78743-3_28.

[7] A. Boronat & J. Meseguer (2009): *Algebraic Semantics of OCL-Constrained Metamodel Specifications*. In: *TOOLS-EUROPE'09*, *LNBIP* 33, Springer, doi:10.1007/978-3-642-02571-6_7.

[8] A. Boronat & P. C. Ölveczky (2010): *Formal Real-Time Model Transformations in MOMENT2*. In: *Proc. FASE'10*, *Lecture Notes in Computer Science* 6013, Springer, doi:10.1007/978-3-642-12029-9.

[9] A. Cerone & A. Maggiolo-Schettini (1999): *Time-Based Expressivity of Time Petri Nets for System Specification*. *Theoretical Computer Science* 216(1-2), pp. 1–53, doi:10.1016/S0304-3975(98)00008-5.

[10] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer & C. Talcott (2007): *All About Maude - A High-Performance Logical Framework*. *Lecture Notes in Computer Science* 4350, Springer, doi:10.1007/978-3-540-71999-1.

[11] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs & Y. Xiong (2003): *Taming Heterogeneity—the Ptolemy Approach*. *Proceedings of the IEEE* 91(2), pp. 127–144, doi:10.1109/JPROC.2002.805829.

[12] *Erlang home page*. `http://www.erlang.org/`.

[13] M. Katelman, J. Meseguer & J. Hou (2008): *Redesign of the LMST Wireless Sensor Protocol through Formal Modeling and Statistical Model Checking*. In: *Proc. FMOODS'08*, *Lecture Notes in Computer Science* 5051, Springer, doi:10.1007/978-3-540-68863-1_10.

[14] D. Lepri, P. C. Ölveczky & E. Ábrahám: *Timed CTL Model Checking in Real-Time Maude*. Submitted for publication.

[15] E. Lien & P. C. Ölveczky (2009): *Formal Modeling and Analysis of an IETF Multicast Protocol*. In: *Proc. SEFM'09*, IEEE Computer Society, doi:10.1109/SEFM.2009.11.

[16] J. Meseguer & P. C. Ölveczky (2010): *Formalization and Correctness of the PALS Architectural Pattern for Distributed Real-Time Systems*. In: *Proc. ICFEM'10*, *Lecture Notes in Computer Science* 6447, Springer, doi:10.1007/978-3-642-16901-4_21.

[17] S. P. Miller, D. D. Cofer, L. Sha, J. Meseguer & A. Al-Nayeem (2009): *Implementing Logical Synchrony in Integrated Modular Avionics*. In: *Proc. DASC'09*, IEEE, doi:10.1109/DASC.2009.5347579.

[18] P. C. Ölveczky (2008): *Towards Formal Modeling and Analysis of Networks of Embedded Medical Devices in Real-Time Maude*. In: *Proc. SNPD'08*, IEEE, doi:10.1109/SNPD.2008.42.

[19] P. C. Ölveczky, A. Boronat & J. Meseguer (2010): *Formal Semantics and Analysis of Behavioral AADL Models in Real-Time Maude*. In: *Proc. FMOODS/FORTE'10*, *Lecture Notes in Computer Science* 6117, Springer, pp. 47–62, doi:10.1007/978-3-642-13464-7_5.

[20] P. C. Ölveczky, A. Boronat, J. Meseguer & E. Pek (2010): *Formal Semantics and Analysis of Behavioral AADL Models in Real-Time Maude*. Report, `http://www.ifi.uio.no/RealTimeMaude/AADL/`.

[21] P. C. Ölveczky & M. Caccamo (2006): *Formal Simulation and Analysis of the CASH Scheduling Algorithm in Real-Time Maude*. In: *Proc. FASE'06*, *Lecture Notes in Computer Science* 3922, Springer, doi:10.1007/11693017_26.

[22] P. C. Ölveczky & J. Meseguer (2008): *The Real-Time Maude Tool*. In: *Proc. TACAS'08*, *Lecture Notes in Computer Science* 4963, Springer, doi:10.1007/978-3-540-78800-3_23.

[23] P. C. Ölveczky, J. Meseguer & C. L. Talcott (2006): *Specification and Analysis of the AER/NCA Active Network Protocol Suite in Real-Time Maude*. *Formal Methods in System Design* 29(3), pp. 253–293, doi:10.1007/s10703-006-0015-0.

[24] P. C. Ölveczky & S. Thorvaldsen (2009): *Formal Modeling, Performance Estimation, and Model Checking of Wireless Sensor Network Algorithms in Real-Time Maude*. *Theoretical Computer Science* 410(2-3), pp. 254–280, doi:10.1016/j.tcs.2008.09.022.

[25] J. E. Rivera (2010): *On the Semantics of Real-Time Domain Specific Modeling Languages*. Ph.D. thesis, Universidad de Málaga.

[26] J. E. Rivera, F. Durán & A. Vallecillo (2010): *On the Behavioral Semantics of Real-Time Domain Specific Visual Languages*. In: *Proc. WRLA'10*, *Lecture Notes in Computer Science* 6381, Springer, doi:10.1007/978-3-642-16310-4_12. See also the e-Motions web page `http://atenea.lcc.uma.es/index.php/Main_Page/Resources/E-motions`.

[27] SAE AADL Team (2009): *AADL Homepage*. `http://www.aadl.info/`.

[28] J. Srba (2008): *Comparing the Expressiveness of Timed Automata and Timed Extensions of Petri Nets*. In: *Proc. FORMATS'08*, *Lecture Notes in Computer Science* 5215, Springer, doi:10.1007/978-3-540-85778-5_3.

[29] F. Wang (2004): *Formal Verification of Timed Systems: A Survey and Perspective*. *Proceedings of the IEEE* 92(8), pp. 1283–1307, doi:10.1109/JPROC.2004.831197.