Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 174 (2007) 65–81
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

Recent Advances in Real-Time Maude

Peter Csaba Ölveczky[a] and José Meseguer[b]

[a] *Department of Informatics, University of Oslo*

[b] *Department of Computer Science, University of Illinois at Urbana-Champaign*

Abstract

This paper gives an overview of recent advances in Real-Time Maude. Real-Time Maude extends the Maude rewriting logic tool to support formal specification and analysis of object-based real-time systems. It emphasizes ease and generality of specification and supports a spectrum of analysis methods, including symbolic simulation, unbounded and time-bounded reachability analysis, and LTL model checking. Real-Time Maude can be used to specify and analyze many systems that, due to their unbounded features, such as unbounded data structures or dynamic object and message creation, cannot be modeled by current timed/hybrid automaton-based tools. We illustrate this expressiveness and generality by summarizing two case studies: (i) an advanced scheduling algorithm with unbounded queues; and (ii) a state-of-the-art wireless sensor network algorithm. Finally, we give some (often easily checkable) conditions that ensure that Real-Time Maude's analysis methods are *complete*, also for dense time, for object-based real-time systems. In practice, our result implies that Real-Time Maude's time-bounded search and model checking of LTL time-bounded formulas are complete decision procedures for a large and useful class of non-Zeno real-time systems that fall outside the scope of systems that can be modeled in decidable fragments of hybrid automata, including the sensor network case study discussed in this paper.

Keywords: formal analysis, real-time systems, rewriting logic, Maude, object-oriented specification, wireless sensor networks, completeness

1 Introduction

Users of formal tools face a choice between expressiveness and generality of the modeling formalism on the one hand, and the availability of decidable and complete analysis methods on the other. For real-time systems, tools based on timed and linear hybrid automata, such as UPPAAL [13] and HyTech [12], have been successful in modeling and analyzing an impressive collection of systems. However, while their restrictive specification formalism ensures that interesting properties are decidable, such finite-control automata do not support well the specification of larger systems with different data types, communication models, and advanced object-oriented features.

Real-Time Maude [18,20] is a high-performance tool that extends the rewriting logic-based Maude system [9,10] to support the formal specification and analysis of real-time systems. Real-Time Maude emphasizes expressiveness and ease of speci-

doi:10.1016/j.entcs.2006.10.020

fication over algorithmic decidability of key properties. In Real-Time Maude, the data types of a system (which may be unbounded) are defined by *equational specifications*, instantaneous transitions are defined by *rewrite rules*, and time elapse is defined by *"tick" rewrite rules*. Real-Time Maude supports the specification of distributed *object-oriented* real-time systems in a natural way, and its flexible specification formalism allows us to easily define different communication models at various levels of abstraction.

Real-Time Maude specifications are *executable*. Our tool offers a wide spectrum of efficient analysis methods. Timed *rewriting* can simulate *one* of the many possible fair concurrent behaviors of the system. Timed breadth-first *search* and *time-bounded linear temporal logic model checking* can analyze *all* behaviors—relative to a given treatment of dense time as explained below—from a given initial state up to a certain duration. By restricting search and model checking to behaviors up to a certain duration, the set of reachable states is restricted to a finite set (unless the system exhibits pathological Zeno behaviors) that can be subjected to model checking.

The time domain may be discrete or dense. Timed automata "discretize" dense time by defining "clock regions," so that all states in the same clock region are bisimilar and satisfy the same properties [4]. In general, the clock region construction cannot be employed in the more complex systems expressible in Real-Time Maude. Real-Time Maude deals with dense time by offering a choice of different "time sampling" strategies, so that instead of searching the whole time domain, only *some* moments in time are visited. For example, one strategy offers the choice of visiting states at user-specified time intervals; another strategy allows time to advance "as much as possible" before something "interesting" happens. Real-Time Maude search and model checking algorithms analyze *all* behaviors *up to* the given strategy for advancing time. Such analyses are in general *incomplete* for dense time, since there is no guarantee that the sampling strategy covers all interesting behaviors, but, as we explain below, completeness is easy to achieve in many practical applications.

The goal of this paper, as its title suggests, is to give an overview of recent advances in Real-Time Maude. Specific advances have been individually reported elsewhere [16,22,23,19], but no overview of the state-of-the-art exists. This paper tries to fill this gap and provide answers to questions such as: (i) what is Real-Time Maude good for?; (ii) how is it different from UPPAAL and HyTech?; (iii) what logical properties can be decided?; and (iv) what applications can Real-Time Maude handle that other tools cannot handle? In particular, we emphasize expressiveness, generality, and completeness aspects, as well as challenging applications.

The generality of Real-Time Maude's formalism and analysis methods, has allowed us to model and analyze state-of-the-art systems in very different application areas. In Section 3, we summarize the modeling and analysis of a proposed optimization of the sophisticated CASH *scheduling algorithm* [7]. The CASH algorithm has advanced capacity sharing features for reusing unused execution budgets. It cannot be modeled by the above mentioned automaton-based tools, because the number of elements in the queue of unused resources can grow beyond any bound. Therefore, an unbounded data type is needed to model the queue. In joint work with Marco Caccamo, the first author specified the algorithm and used Real-Time Maude's search command to find an extremely subtle bug showing that it is indeed possible to reach states where a hard deadline is missed.

There are extensions of basic models for concurrent and real-time systems that add some support for user-definable data types, such as different kinds of colored (timed) Petri nets [1], TE-LOTOS [14], and the IF toolset [5]. Real-Time Maude complements such systems not only by the full generality of the specification language and the range of analysis methods, but also by its simplicity: a uniform and intuitive formalism is used to specify both the data types (by *equations*) and the dynamic and real-time behavior of the system (by *rewrite rules*). Furthermore, in contrast to formalisms based on a fixed communication model, many different models of communication can easily be expressed.

This latter feature is important when modeling advanced *wireless sensor network algorithms*. Such algorithms pose many challenges to their modeling and analysis, as explained in Section 4. In joint work with Stian Thorvaldsen, the first author recently modeled, simulated, and analyzed the sophisticated OGDC algorithm for density control in wireless sensor networks. To the best of our knowledge, our work on OGDC represents the first attempt at using a formal tool on advanced wireless sensor network algorithms. In Section 4, we show how communication in wireless sensor networks, at the level of abstraction of OGDC, can be modeled, and briefly summarize the modeling and analysis of the OGDC algorithm.

As mentioned above, the time sampling-based analysis methods provided by Real-Time Maude are in general *incomplete* for dense time, since not all times can be visited, and there is no guarantee that the behaviors obtained by the sampling technique cover all possible behaviors. In Section 5 we give some easily checkable conditions for typical object-oriented specifications that ensure that *maximal time sampling* analyses are *complete* analysis methods. The conditions are satisfied by a large and useful class object-oriented specifications which fall outside the class of systems that can be modeled as timed automata—including many of our large case studies, for example the AER/NCA active network protocol suite [21] and the OGDC wireless sensor network algorithm. In practice, our results imply that time-bounded search and model checking of time-bounded LTL formulas are complete decision procedures for a large and useful class of non-Zeno real-time systems.

2 Real-Time Maude

A Real-Time Maude *timed module* specifies a *real-time rewrite theory* [17] of the form $(\Sigma, E, \phi, IR, TR)$, where:

- (Σ, E) is a *membership equational logic* [15] theory, with Σ a signature[1] and E a set of conditional equations and memberships. The theory (Σ, E) specifies the

[1] That is, Σ is a set of declarations of *sorts*, *subsorts*, and *function symbols* (or *operators*).

system's state space as an algebraic data type. (Σ, E) must contain a specification of a sort `Time`, modeling the time domain (which may be dense or discrete).

- ϕ is a function which associates to each function symbol in Σ its *frozen* [2] argument positions [6].
- *IR* is a set of *labeled conditional instantaneous rewrite rules* specifying the system's instantaneous (i.e., zero-time) local transitions, each of which is written `crl [`l`] :` t `=>` t' `if` *cond*, where l is a *label.* Such a rule specifies a *one-step instantaneous transition* from an instance of t to the corresponding instance of t', *provided* the condition holds. The rewrite rules are applied *modulo* the equations E. [3]
- *TR* is a set of *tick (rewrite) rules*, written with syntax

  `crl [`l`] : {`t`} => {`t'`} in time` τ `if` *cond* `.`

  that model the elapse of time in a system. The operator `{_}` is a built-in constructor of sort `GlobalSystem`, and τ is a term of sort `Time` that denotes the *duration* of the rewrite.

The initial states must be ground terms of sort `GlobalSystem` and must be reducible to terms of the form `{`t`}` using the equations in the specification. The form of the tick rules then ensures uniform time elapse in all parts of the system.

In object-oriented timed modules, a *class* declaration

`class` C `|` att_1 `:` s_1`, ... ,` att_n `:` s_n `.`

declares a class C with attributes att_1 to att_n of sorts s_1 to s_n. An *object* of class C in a given state is represented as a term `<` $O : C \mid att_1 : val_1, ..., att_n : val_n$ `>`, where O (of sort `Oid`) is the object's identifier, and where val_1 to val_n are the current values of the attributes att_1 to att_n. A *message* m is a term of sort `Msg`. We can easily define *delayed messages* (see [20]) as terms of the form `dly(`m`,`τ`)`, which denotes a message m that will be "ripe" in time τ (that is, it will become m in time τ). In a concurrent object-oriented system, states have the form `{`t`}`, where t is a term of the built-in sort `Configuration`. [4] t has typically the structure of a *multiset* made up of objects, ripe messages, and delayed messages. Multiset union for configurations is denoted by a juxtaposition operator (empty syntax) that is declared associative and commutative, so that rewriting is *multiset rewriting* supported directly in Maude. The zero-time dynamic behavior of concurrent object systems is axiomatized by specifying each of its concurrent transition patterns by an instantaneous rewrite rule. For example, the rule

```
rl [l] : m(O,w)
         < O : C | a1 : x, a2 : O', a3 : z >
```

[2] Rewrites cannot take place in a frozen argument position of a function symbol, so that a term $f(t_1, \dots, t_i, \dots, t_n)$ will *not* rewrite to $f(t_1, \dots, u_i, \dots, t_n)$ when t_i rewrites to u_i if $i \in \phi(f)$.

[3] The set E of equations and memberships is a union $E' \cup A$, where A is a set of equational axioms such as associativity, commutativity, and identity, and E is Church-Rosser modulo A. Operationally, a term is reduced to its E'-normal form modulo A before any rewrite rule in *IR* or *TR* is applied.

[4] In general, the operator `{_}` is declared to take arguments of the sort `System`. In object-oriented modules, the sort `Configuration` is automatically defined to be a subsort of `System` [20].

```
        =>
         < O : C | a1 : x + w, a2 : O', a3 : z >
         dly(m'(O'),x) .
```

defines a family of transitions in which a message m, with parameters O and w, is read and consumed by an object O of class C. These transitions have the effect of altering the attribute a1 of the object O and of sending a new message m'(O') with delay x. Attributes, such as a3, whose values do not change and do not affect the next state of other attributes, need not be mentioned in a rule. Attributes, like a2, whose values influence the next state but are themselves unchanged, may be omitted in right-hand sides of rules. The above rule could therefore be given as

```
rl [l] : m(O,w)  < O : C | a1 : x, a2 : O' >
        =>
         < O : C | a1 : x + w >  dly(m'(O'),x) .
```

An object-oriented specification usually has only one tick rule, which typically has the form

```
var C : Configuration .   var T : Time .
crl [tick] : {C} => {δ(C, T)} in time T  if T <= mte(C) [nonexec] .
```

The function δ defines the effect of time elapse on the objects and messages in a configuration, and the function mte defines the maximum amount of time that can elapse before some instantaneous action must take place. These functions distribute over the elements in a configuration according to the following equations:

```
vars NeC NeC' : NEConfiguration .     var R : Time .
eq δ(none, R) = none .
eq δ(NeC NeC', R) = δ(NeC, R) δ(NeC', R) .

eq mte(none) = INF .
eq mte(NeC NeC') = min(mte(NeC), mte(NeC')) .
```

The functions δ and mte must then be defined on *individual* objects by application-specific equations. The tick rule advances time nondeterministically by *any* amount T less than or equal to mte(C), and is therefore non-executable ([nonexec]) until T is given a concrete value. Our tool deals with such rules by offering a choice of different "time sampling" strategies for setting the value of T. For example, the *maximal* time sampling strategy advances time by the maximum possible time elapse mte(C) in tick rules of the above form (unless mte(C) equals the infinity value INF). Another time sampling strategy always advances time by a user-given time increment Δ in time-nondeterministic tick rules. All applications of such tick rules—be it for rewriting, search, or model checking—are performed using the selected time sampling strategy. This means that some behaviors in the system, namely those obtained by applying the tick rules differently, are not analyzed.

Timed modules are *executable* under reasonable assumptions, and Real-Time Maude provides a spectrum of formal analysis capabilities. Real-Time Maude's *timed "fair" rewrite* command simulates *one* behavior of the system *up to a certain* *duration*. It is written with syntax

```
(tfrew t in time <= τ .)
```

where t is the initial state and τ is a ground term of sort `Time`.

Real-Time Maude's timed *search* command uses a breadth-first strategy to search for states that are reachable from the initial state within a given time, match a *search pattern*, and satisfy a *search condition*. The command which searches for *one* state satisfying these search criteria has syntax [5]

```
(tsearch [1] t =>* pattern such that cond <= τ .)
```

Real-Time Maude also provides an *untimed* command to search for a state reachable in any amount of time. Such search, while not guaranteed to terminate, is sometimes more efficient than timed search because it does not have to keep track of durations.

Real-Time Maude extends Maude's *linear temporal logic model checker* [10] to check whether each behavior "up to a certain time," as explained in [20], satisfies a propositional temporal logic formula. Propositions may also be *clocked*, in that they take the elapsed time into account. In this way, the temporal logic does not need to be complicated with explicit real-time syntax, but can express many important real-time properties. The key point is that global system states have a *global clock* component, and may also contain timers and other time-related sub-components, and these time aspects can be tested by atomic propositions.

3 Modeling and Analyzing the CASH Scheduling Algorithm

The increasing sophistication of scheduling algorithms makes it challenging, if not impossible, to model such algorithms using automaton-based formalisms. In joint work with Marco Caccamo, we have modeled and analyzed the state-of-the-art *CASH* scheduling algorithm [7] and a proposed improvement of this algorithm. This work is reported in [16]; here we give an overview.

The CASH algorithm attempts to maximize system performance while guaranteeing that critical tasks are executed in a timely manner. The idea is that when a task instance (or *job*) needs less than its allocated execution time (which is usually its average-case execution time), it can make the remaining execution time available to other jobs by placing the unused execution time in a global *capacity sharing* (CASH) queue of unused budgets. A job can then reuse execution times from unused budgets with earlier deadlines than the job's own deadline, allowing the job to execute for more than its allocated execution time, if needed, thereby increasing system performance. If the job does not need all that execution time, it places the remaining budget in the CASH queue. When the system is idling (i.e., no job is ready to execute), the spare capacity with the *earliest* deadline must be discharged according to the elapsed time. The crucial schedulability result that can be guar-

[5] Search commands using the arrows `=>+` and `=>!` instead of `=>*` search for, respectively, states that are reachable in one or more steps, and states that cannot be further rewritten.

anteed off-line is that each job can execute its allocated execution time (including the spare budget it uses) before its deadline, as long as the sum of the bandwidths of the servers is less than or equal to 1.

One of the developers of the CASH algorithm, Marco Caccamo, wanted to use Real-Time Maude to analyze a potential optimization of the CASH algorithm, where, instead of discharging the spare budget with the *earliest* deadline when idling, the budget with the *latest* deadline is discharged. In this way, the more "valuable" resources with earlier deadlines could be reserved for executing jobs (since jobs can only use budgets having earlier deadlines than the job's deadline). The key question is then whether the schedulability result also holds for the modified scheduling algorithm.

Simulating the CASH algorithm using *timed rewriting* showed that the number of spare budgets in the CASH queue can grow beyond any bound. This implies that "unbounded" data structures are needed to model the CASH algorithm, something that is impossible to do with timed automata.

3.1 Modeling the CASH Algorithm

We have specified (both versions) of the CASH algorithm, for *all* possible task sets, by allowing a job to arrive at *any* time and to execute for *any* non-zero amount of time. We have specified the algorithm in an object-oriented style, where each task server is modeled by an object of a class `Server`. The specifications are given in detail in [16]. In what follows, we just give a flavor for the specification.

We represent a *spare capacity* as a term `deadline:` d `budget:` b, where d is its *relative* deadline and b is its remaining budget. The cash queue of spare capacities is represented by a term `[CASH:` $c_1 \ldots c_n$ `]`, where $c_1 \ldots c_n$ is a queue of spare capacities. The sorts and operators for this data type are specified as follows:

```
sorts Capacity CapacityQueue .      subsort Capacity < CapacityQueue .

op deadline:_budget:_ : Time Time -> Capacity [ctor] .
op emptyQueue : -> CapacityQueue [ctor] .
op __ : CapacityQueue CapacityQueue -> CapacityQueue
                             [ctor assoc id: emptyQueue] .
sort Cash .     subsort Cash < Configuration .
op `[CASH:_`] : CapacityQueue -> Cash [ctor] .
```

To illustrate the specification style, we present the following rule that models the case when a new job (whose arrival is modeled implicitly by changing the `state` attribute of the server from `idle` to either `executing` or `waiting`) arrives at the task server `O`, while another task server `O'` is executing a job. The CASH algorithm is based on *earliest deadline first* (EDF) scheduling, which means that the job with the earliest deadline must always execute. Therefore, depending on whether or not the new relative deadline of this new job (given by the previous relative deadline `T` plus the period `NZT` of `O`) comes before or after the relative deadline `T'` of the currently executing job, `O` either preempts `O'` and starts executing, or goes into the `waiting` state and allows `O'` to continue executing:

```
vars O O' : Oid .   vars T T' : Time .  var NZT : NzTime .

rl [idleToActive] :
   < O : Server | period : NZT, state : idle, timeToDeadline : T >
   < O' : Server | state : executing, timeToDeadline : T' >
  =>
   if (T + NZT) < T' then   --- start to execute and preempt O':
      (< O : Server | state :  executing, timeToDeadline : T + NZT,
                      timeExecuted : 0, usedOfBudget : 0 >
       < O' : Server | state :  waiting >)
   else                     --- cannot preempt O'; start waiting:
      (< O : Server | state : waiting, timeToDeadline : T + NZT,
                      timeExecuted : 0, usedOfBudget : 0 >
       < O' : Server | >)
   fi .
```

To make our analysis more convenient, we add a constant `DEADLINE-MISS` and a rule which rewrites an object whose remaining budget is larger than its relative deadline to `DEADLINE-MISS`:

```
var STATE : ServerState .
op DEADLINE-MISS : -> Configuration [ctor] .

crl [deadlineMiss] :
    < O : Server | state : STATE, usedOfBudget : T, timeToDeadline : T',
                   maxBudget : NZT >
   =>
    DEADLINE-MISS
   if (NZT - T) > T' /\ (STATE == waiting or STATE == executing) .
```

3.2 Formal Analysis of the CASH Algorithms

The main purpose of our analysis is to investigate whether or not it is possible to reach a state where the execution of the remaining budget cannot be done within the current deadline. Since the time domain is discrete, we select the time sampling strategy '`def 1`' which increments time by one time unit in each application of a tick rewrite rule. In this way, *all possible task sets* can be explored. The following term `init2` defines an initial state with two servers. Since the sum of their bandwidths is $\frac{34}{35} \leq 1$, it should *not* be possible to reach a missed deadline if the algorithm is correct:

```
op init2 : -> GlobalSystem .
eq init2 =
    {< s1 : Server | maxBudget : 2, period : 5, timeExecuted : 0,
                     state : idle, usedOfBudget : 0, timeToDeadline : 0 >
< s2 : Server | maxBudget : 4, period : 7, timeExecuted : 0,
                state : idle, usedOfBudget : 0, timeToDeadline : 0 >
[CASH: emptyQueue]   AVAILABLE-PROCESSOR} .
```

We use time-bounded search to check whether a missed deadline can be reached from state `init2`. The pattern `{DEADLINE-MISS C:Configuration}` matches any state containing the constant `DEADLINE-MISS`, since the variable `C:Configuration` matches the rest of the configuration. Time-bounded search among states reachable within time 12 found a missed deadline:

```
Maude> (tsearch [1] init2 =>* {DEADLINE-MISS C:Configuration}
                                    in time <= 12 .)

Solution 1
C:Configuration <- ... ;  TIME_ELAPSED:Time <- 12
```

The underlying trace facilities for search commands in Maude were used to exhibit the sequence of rewrite steps leading from state `init2` to the missed deadline. It is also worth remarking that extensive *Monte Carlo simulation* of the CASH algorithm did not discover the missed deadline [16], strongly suggesting that this is a very subtle bug unlikely to be found by either testing or simulation.

4 Modeling and Analysis of Wireless Sensor Network Algorithms

A wireless sensor network consists of a set of small, cheap, and low-power sensor nodes that use wireless technology to communicate with each other [3]. Most often, it is assumed that sensor nodes communicate by broadcasting using a radio transmitter with an undirected antenna. Sensor nodes tend to have limited power supply (provided by a battery) that is virtually impossible to replace.

Advanced wireless sensor network algorithms present a set of challenges to formal tools, including:

(i) Modeling and reasoning about *time-dependent* behavior. For example, longevity of the network is often a crucial goal, in which case power consumption must be modeled. In addition, wireless sensor network algorithms may use timers, message transmission may be subject to message delays, and so on.

(ii) Many algorithms depend on spatial features such as locations, distances, coverage areas, etc.

(iii) Modeling different forms of communication. For sensor nodes transmitting by radio, the appropriate model of *direct* communication may be broadcast, where only nodes within a certain distance from the sender receive the signal with sufficient strength.

(iv) Wireless sensor network algorithms often incorporate *probabilistic* behaviors.

(v) Simulating and analyzing systems with a large number of sensor nodes scattered randomly in a sensing area.

(vi) Both correctness and, in particular, performance are critical aspects that must be analyzed.

In Real-Time Maude, spatial features (challenge (2)) can be defined by the user with suitable data types. Regarding challenge (3), Real-Time Maude's flexible specification formalism allows us to easily define different forms of communication. We show in this paper how to model geographically bounded broadcast with transmission delays. Real-Time Maude does not provide explicit support for modeling and reasoning about probabilistic behaviors (challenges (4) and, partially, (6)), which are supported by another extension of Maude called PMaude [2]. Nevertheless, for the purpose of *simulating* a system directly in Real-Time Maude, probabilistic behaviors can be "sampled" using a pseudo-random number generator. For correctness analysis, we can model probabilistic behavior by nondeterminism as explained in [22]. Regarding (5), we can easily define states with any given number of nodes scattered pseudo-randomly as shown in [22]. Finally, system correctness and performance can be analyzed by Real-Time Maude as explained in [22,23].

Jennifer Hou recently suggested to us the state-of-the-art *optimal geographical density control* (OGDC) algorithm [24] for wireless sensor networks as a challenging modeling and analysis task. OGDC has been simulated in the simulation tool ns-2, where its performance was compared to the performance of similar algorithms. OGDC presents all the challenges (1) to (6) above.

We have modeled, simulated, and analyzed OGDC in Real-Time Maude [23]. To the best of our knowledge, our work on OGDC represents the first formal modeling and analysis effort of sophisticated wireless sensor network algorithms. We were able to do in Real-Time Maude *all* the analyses that the developers of OGDC performed using the wireless extension of ns-2 [11]. In addition, we have subjected the algorithm to time-bounded reachability analysis and temporal logic model checking. Such analyses normally explore all possible behaviors from a certain state, but in our case they were also relative to the sampling techniques used for simulating probabilistic behaviors.

This paper intends to give some high-level ideas on how to model wireless sensor network algorithms. The paper [22] explains in more detail how sensor network algorithms in general can be modeled and analyzed in Real-Time Maude. The report [23] describes the OGDC case study in detail.

4.1 Modeling Communication in Wireless Sensor Networks

In [22] we have explained how various aspects of sensor networks can be modeled. For example, for *simulating* probabilistic behaviors, we include in the state an object of class `RandomNGen`, which carries the *seed* of the pseudo-random generator which is used to "sample" probabilistic behaviors. In this section, we show how to model communication in wireless sensor networks at the level of abstraction of the OGDC algorithm.

We first need to define *locations*. If the sensor nodes are located in a two-dimensional surface, we can represent a location as a pair $x\,.\,y$ of rational numbers.

Using the built-in sort `Rat` of rational numbers, such pairs can be represented as terms of the following sort `Location`:

```
sort Location .
op _._ : Rat Rat -> Location [ctor] .
```

The following function defines the *square* of the distance between two locations: [6]

```
op distanceSq : Location Location -> Rat .
vars X X' Y Y' : Rat .
eq distanceSq(X . Y, X' . Y) =
      ((X - X') * (X - X')) + ((Y - Y') * (Y - Y')) .
```

Given a constant `transRange` denoting the transmission range of a sensor node, we can check whether a sensor node is within the transmission range of another sensor node:

```
vars L L' : Location .
op _withinTransRangeOf_ : Location Location -> Bool .
eq L withinTransRangeOf L' = distanceSq(L, L') <= transRange * transRange .
```

Each sensor node can suitably be represented as an object of, say, a class called `WSNode`. A wireless sensor node usually does not have an explicit identifier, but can be identified by its location. In Real-Time Maude, we let object identifiers be locations by giving the subsort declaration `subsort Location < Oid .`

In what follows, we model broadcast where a packet must reach all nodes within the radio range of the sender, and where the transmission is subject to a transmission delay Δ. The idea is that the sender l sends a "broadcast message" of the form `broadcast` m `from` l, where m is the message content, into the configuration. This broadcast message is defined by equations to be *equivalent* to a set of single, addressed messages `dly(msg` m `from` l `to` l'`,` Δ`)` with delay Δ, one for each sensor node l' within the radio range of l. The messages are declared as follows:

```
sort MsgCont .           --- Message content
msg broadcast_from_ : MsgCont Location -> Msg .
msg msg_from_to_ : MsgCont Location Location -> Msg .
```

The following equation captures the desired equivalence:

```
var C : Configuration .  var MC : MsgCont .
eq {< L : WSNode | >  (broadcast MC from L)  C} =
     {< L : WSNode | >  distributeMsg(L, MC, C)} .
```

It is the task of `distributeMsg` to create an addressed message for each sensor object in the configuration `C` that is within the transmission range of `L`. The use of the operator `{_}` enables the equation to grab the *entire* state to ensure that *all* appropriate nodes in the system will get the message. The function `distributeMsg` is defined recursively over the elements in a configuration:

[6] Real-Time Maude also provides a built-in data type of floating-point numbers, with functions such as square root, but we prefer to stay within the rational numbers whenever possible.

```
op distributeMsg : Location MsgCont Configuration -> Configuration
                                                      [frozen (3)] .
var MSG : Msg .  var O : Oid .

eq distributeMsg(L, MC, none) = none .
eq distributeMsg(L, MC, MSG C) = MSG distributeMsg(L, MC, C) .
eq distributeMsg(L, MC, < O : RandomNGen | > C) =
     < O : RandomNGen | > distributeMsg(L, MC, C) .

eq distributeMsg(L, MC, < L' : WSNode | >  C) =
     < L' : WSNode | > distributeMsg(L, MC, C)
     (if L withinTransRangeOf L'
      then dly(msg MC from L to L', Δ)
      else none fi) .
```

The first equation says that the configuration resulting from distributing the message with content `MC` to the empty configuration (denoted by `none`) is just the empty configuration. The second equation says that the configuration resulting from distributing the message with content `MC` to a configuration consisting of a message `MSG` together with a configuration `C` is the configuration consisting of `MSG` together with the result of distributing `MC` to `C`. The third equation just states that the "extra" `RandomNGen` object(s) which might be included for simulation purposes should not get the message. The crucial equation is the last equation, which says that distributing `MC` from sender `L` to a configuration consisting of a node at location `L'` and the rest of the configuration `C` is the same as recursively distributing the message `MC` to `C`, keeping the object `L'`, and also have the message `dly(msg MC from L to L',`Δ`)` if `L'` is within the transmission range of `L`. That is, we have added this message to the system. If the transmission delay between two nodes l and l' is a function of the distance between them, say $f(l, l')$, we just replace Δ by $f($`L,L'`$)$.

In this setting, broadcasting a message m from sensor node l is modeled by a rule that sends a `broadcast` message into the configuration in the usual Maude style explained in Section 2. In Section 4.2 we show an example of such a rule. Likewise, reading a message is done by reading a `msg` m `from` ... message in usual Maude style.

4.2 Modeling and Analyzing the OGDC Wireless Sensor Network Algorithm

In a two-dimensional plane, a node with *sensing range* r_s can *sense* events in a circular *coverage area* with radius r_s. It is desirable that the coverage areas of the *active* nodes cover the entire area to be monitored (the "sensing area") for as long as possible. A large number of nodes is often deployed to extend the lifetime of a wireless sensor network, so that some nodes can be intentionally "put to sleep" to save power. The OGDC algorithm [24] is a state-of-the-art algorithm that periodically chooses the nodes that can be put to sleep while maintaining coverage of the sensing area.

In OGDC, the network lifetime is divided into *rounds*. A round begins with each node entering a *volunteering process* where it *probabilistically* chooses whether or not to volunteer to be a *starting node*. Each node that volunteers sets its *backoff timer* to a small value. The node then becomes *active* when its backoff timer expires, and broadcasts a *power-on* message which contains the location of the node and a *random direction* (see rule `startingNodePowerOn` below). When a node receives a power-on message, it checks if its entire coverage area is covered by the surrounding active nodes, in which case the node becomes inactive. Otherwise, the node sets its backoff timer depending on how close the node is to the optimal position. When the backoff timer of a node expires, the node becomes active and broadcasts a power-on message. The network enters the steady state phase when each node is either active or inactive. When a round is over, the density control process starts over again.

We present below one of the 11 rewrite rules in our Real-Time Maude specification of OGDC. As mentioned, a node becomes *active* (its `status` attribute is set to `on`) when its backoff timer expires (i.e., has value `0`) and the node has volunteered to be a starting node. The node becomes active and *broadcasts* a power-on message that contains the node's location and a *random direction*. This action is modeled by the following rewrite rule:

```
var L : Location .   var P : NzNat .   var M : Nat .

rl [startingNodePowerOn] :
   < L : WSNode | powerLeft : P, backoffTimer : 0, hasVolunteered : true >
   < Random : RandomNGen | seed : M >
  =>
   < L : WSNode | powerLeft : P - tP, backoffTimer : INF, status : on >
   < Random : RandomNGen | seed : random(M) >
   broadcast (powerOnWithDirection randomDirection(M)) from L .
```

As mentioned in Section 2, a *time sampling strategy* must be chosen before the analysis can take place. We show in Section 5 that we can "fast forward" between the events in OGDC without losing any interesting behaviors. Therefore, in our analysis, we used the *maximal* time sampling strategy, which advances time as much as possible (as defined by `mte`).

In [24], Zhang and Hou use the simulation tool ns-2, with the wireless extension [11], to simulate OGDC and measure the number of *active* nodes, the *percentage of coverage* provided by those nodes, and the *total amount of power* in the system throughout the network's lifetime.

We added to the initial state a new construction called *analysis message* to compute the appropriate performance metric of the state at the end of each round. Timed rewriting could then simulate the OGDC algorithm with several hundred sensor nodes and could measure *all* performance metrics measured by the OGDC developers using ns-2. In our simulations, we generally got a larger number of active nodes than reported in [24]. A plausible explanation is given in [23] and essentially means that, in the OGDC algorithm, more nodes will become active if transmission delays are taken into account in the simulation (as was done in our case) than if such delays are ignored, which *may* have been the case in the simulations in [24]. Since transmission delays play a significant role in the definition of the OGDC algorithm, they should be taken into account in simulations. Therefore, our formal model may provide a more appropriate simulation setting for OGDC than ns-2 with the wireless extension.

We have also model checked the specification to explore *all* behaviors from the initial state, *relative to our treatment of probabilistic behavior*, to find out how long it takes, in the worst case, to reach the steady state phase, and whether the entire sensing area is covered in this phase in the first round.

5 Completeness and Abstraction in Real-Time Maude

The previous sections show the expressive power and generality of the Real-Time Maude formalism. The price to pay for such expressiveness is that Real-Time Maude analyses are in general *incomplete*[7] for *dense time*, because all moments in time cannot be visited. For *discrete time*, completeness can be achieved by exhaustively visiting *all* time instants. This makes breadth-first search for violation of an invariant a complete semi-decision procedure. Furthermore, *time-bounded* LTL model checking of systems in which only a finite set of states can be reached within a certain time becomes a complete decision procedure. However, visiting all discrete times typically leads to a state space explosion that renders many formal analyses unfeasible.

In recent work, reported in detail in [19], we have investigated under what conditions Real-Time Maude's *maximal time sampling strategy* yields complete analysis methods. We only consider satisfaction with respect to the set of *timed fair paths.* (A timed *unfair* path is an infinite, pathological path in which the total duration is bounded by some time value τ, *even though* time could have advanced beyond time τ in infinitely many tick rule applications.) Furthermore, we only consider LTL formulas *without the next operator* $\bigcirc$, since formulas with $\bigcirc$ are sensitive to "stuttering" tick steps. For example, if time can advance by two time units in one tick step, then time can most often also advance by one time unit in each of two consecutive tick steps.

For the very useful and general class of flat object-oriented specifications with a tick rule of the form given in Section 2 (see [19] for details), it is sufficient to prove the following five simple and natural conditions, for all objects t and time values τ and τ', to ensure completeness of LTL model checking:

(i) $\texttt{mte}(\delta(t,\tau)) = \texttt{mte}(t)\ \texttt{monus}\ \tau$, for all $\tau \leq \texttt{mte}(t)$. (The function `monus` is defined by $x\ \texttt{monus}\ y = x - y$ if $x \geq y$, and 0 otherwise.) This requirement states that if time can advance on an object (in state) t by at most time $\texttt{mte}(t)$, and the object advances in time by an amount $\tau \leq \texttt{mte(t)}$, then time can advance the resulting (state of the) object $\delta(t,\tau)$ by at most $\texttt{mte}(t)\ \texttt{monus}\ \tau$.

[7] We call an analysis method *complete* if the fact that a counterexample is not found using the method means that no counterexample can exist for the analysis in question [19].

(ii) $\delta(t, 0) = t$. That is, the state of an object should not be changed as a result of time "advancing" by time 0.

(iii) $\delta(\delta(t, \tau), \tau') = \delta(t, \tau + \tau')$, for $\tau + \tau' \leq \texttt{mte}(t)$. That is, if time advances first by time τ and then by τ', then this should have the same effect as if time advances by $\tau + \tau'$ in one step.

(iv) $\texttt{mte}(\sigma(l)) = 0$ for each ground instance $\sigma(l)$ of a left-hand side of an instantaneous rewrite rule. This requirement says that time should not be able to advance (by more than 0) in a state where an instantaneous rule is enabled.

(v) For each atomic proposition (or search pattern) p in the property to be analyzed, $\texttt{\{}t\texttt{\}}$ satisfies p if and only if $\texttt{\{}\delta(t, \tau)\texttt{\}}$ satisfies p, for all terms t, τ with $\tau < \texttt{mte}(t)$.

These proof obligations require proving equalities, which in general is undecidable. In practice, however, the proof obligations are trivial even for large and complex specifications such as OGDC and AER/NCA (which has over 60 rewrite rules), and can usually be proved easily either "by inspection," or by using decision procedures or theorem provers such as Maude's ITP [8].

Indeed, given the complexity of the OGDC algorithm, it is remarkable how easy it is to prove these requirements. For example, it follows directly that `mte` of an instance of the left-hand side of an instantaneous rule is `0` (For rewrite rule `startingNodePowerOn`, `mte` of an object with timer value `0` is `0`.) Proving the other requirements amounts to proving properties like $(x \texttt{ monus } y) \texttt{ monus } z = x \texttt{ monus } (y + z)$ and $x \texttt{ monus } 0 = x$. The last requirement with respect to the the propositions and search patterns used in the analyses in [23] follows trivially, since δ does not modify any of the attributes that define these patterns and propositions.

Our results imply that *time-bounded* search and *time-bounded* LTL model checking using the maximal time sampling strategy are decision procedures for a very useful and general class of *non-Zeno* dense-time object-oriented specifications. This class includes OGDC[8] and AER/NCA, and falls outside of the class of dense-time systems for which well known decision procedures exist. (In the case of the CASH algorithm, actions may happen at any time, which means that the maximal strategy does not give complete analysis methods. Instead, in CASH we exploit the fact that the time domain is discrete for scheduling algorithms, and use the time sampling strategy that advances time by one time unit to cover all possible behaviors.)

In the OGDC algorithm, which is parametric in the time domain, our results mean that, for dense time, maximal time sampling analysis is complete. If the time domain were discrete, then the *abstraction* given by using maximal time sampling instead of the "default" time sampling that visits all discrete times, gives enormous savings, since time is measured in milliseconds (or fractions thereof) in OGDC, while one round of the algorithm lasts for 1,000 *seconds*. An analysis based on visiting all the 1,000,000 time instants in each behavior would be unfeasible.

Our result can be compared to the timed automaton case, in the sense that there

[8] Completeness of OGDC is relative to the treatment of probabilistic features as explained in [19].

are some requirements on a finite automaton extended with dense-time clocks that ensure that such an automaton has a finite quotient (the "region graph"). These requirements can be given by *syntactic* means, and are reflected in the syntactic restrictions on the clock constraints in a timed automaton. In the more expressive Real-Time Maude setting, the requirements ensuring that a dense-time system has a "discrete" (stuttering) bisimulation are *semantic*, although, as mentioned, they are usually fairly easy to check.

6 Concluding Remarks

In this paper, we have given an overview of some recent work on Real-Time Maude. On the one hand, we have illustrated the expressive power, flexibility, and analytic power of the tool by modeling and analyzing new state-of-the-art algorithms developed by leading researchers in quite different fields. In the case of the rapidly emerging field of wireless sensor networks, our work is, to the best of our knowledge, the first formal analysis undertaken on a sophisticated wireless sensor network algorithm. Furthermore, we managed to measure all the performance metrics measured during the OGDC developers' simulations, as well as performing further model checking analyses not possible in typical network simulators.

On the other hand, and most importantly, we have shown that, despite Real-Time Maude's expressiveness, we can still give simple conditions that make model checking and reachability analysis using *maximal time sampling* into *complete* analysis methods for dense time for a large class of object-oriented specifications, including the OGDC case study.

Acknowledgments

We are grateful to Stian Thorvaldsen for the collaboration on modeling and analyzing the OGDC algorithm, to Jennifer Hou for suggesting this algorithm as a challenging formal analysis task, to Marco Caccamo for our collaboration on the CASH algorithm, and to the anonymous referees and Denise West for many helpful comments on earlier versions of this paper. Partial support of this research by ONR Grant N00014-02-1-0715 and by NSF Grant CNS 05-24516 is gratefully acknowledged.

References

[1] W. M. P. van der Aalst. Interval timed coloured Petri nets and their analysis. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 453–472. Springer, 1993.

[2] G. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based specification language for probabilistic object systems. In *Proc. 3rd Workshop on Quantitative Aspects of Programming Languages (QAPL'05)*, 2005.

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: Survey. *Computer Networks*, 38:393–422, 2002.

[4] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[5] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF toolset. In *Proceedings of SFM'04 (Bertinoro, Italy)*, volume 3185 of *Lecture Notes in Computer Science*. Springer, 2004.

[6] R. Bruni and J. Meseguer. Generalized rewrite theories. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP 2003)*, volume 2719 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 2003.

[7] M. Caccamo, G. Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proc. IEEE Real-Time Systems Symposium, Orlando*, December 2000.

[8] M. Clavel. The ITP tool home page. http://maude.sip.ucm.es/itp/.

[9] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 285:187–243, 2002.

[10] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude Manual (Version 2.2)*, December 2005. http://maude.cs.uiuc.edu.

[11] CMU monarch extensions to ns. http://www.monarch.cs.cmu.edu/.

[12] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[13] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.

[14] G. Leduc, L. Leonard, D. de Frutos, and J. Quemada. Time extended LOTOS. Revised working draft on Extended LOTOS. ISO/IEC JTC1/SC21/WG7, July 1995.

[15] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. Parisi-Presicce, editor, *Proc. WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.

[16] P. C. Ölveczky and M. Caccamo. Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In L. Baresi and R. Heckel, editors, *Fundamental Approaches to Software Engineering (FASE'06)*, volume 3922 of *Lecture Notes in Computer Science*, pages 357–372. Springer, 2006.

[17] P. C. Ölveczky and J. Meseguer. Specification of real-time and hybrid systems in rewriting logic. *Theoretical Computer Science*, 285:359–405, 2002.

[18] P. C. Ölveczky and J. Meseguer. Specification and analysis of real-time systems using Real-Time Maude. In T. Margaria and M. Wermelinger, editors, *Fundamental Approaches to Software Engineering (FASE 2004)*, volume 2984 of *Lecture Notes in Computer Science*, pages 354–358. Springer, 2004.

[19] P. C. Ölveczky and J. Meseguer. Abstraction and completeness for Real-Time Maude. *Electronic Notes in Theoretical Computer Science*, 2006. To appear.

[20] P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *Higher-Order and Symbolic Computation*, 2006. To appear.

[21] P. C. Ölveczky, J. Meseguer, and C. L. Talcott. Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. *Formal Methods in System Design*, 2006. To appear.

[22] P. C. Ölveczky and S. Thorvaldsen. Formal modeling and analysis of wireless sensor network algorithms in Real-Time Maude. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*. IEEE Computer Society Press, 2006.

[23] S. Thorvaldsen and P. C. Ölveczky. Formal modeling and analysis of the OGDC wireless sensor network algorithm in Real-Time Maude. Manuscript. http://www.ifi.uio.no/RealTimeMaude/OGDC, October 2005.

[24] H. Zhang and J. C. Hou. Maintaining sensing coverage and connectivity in large sensor networks. *Wireless Ad Hoc and Sensor Networks: An International Journal*, 1, 2005.