Formal Modeling and Analysis of Wireless Sensor Network Algorithms in Real-Time Maude

Peter Csaba Ölveczky and Stian Thorvaldsen
Department of Informatics, University of Oslo, Norway
{peterol, stianth}@ifi.uio.no

Abstract

Advanced wireless sensor network algorithms pose challenges to their formal modeling and analysis, such as modeling probabilistic and real-time behaviors and novel forms of communication, and analyzing both correctness and performance. In this paper, we propose using Real-Time Maude to formally model, simulate, and further analyze such algorithms. The Real-Time Maude formalism is expressive yet intuitive, and the tool provides a spectrum of analysis methods, including simulation, reachability analysis, and temporal logic model checking. We have used Real-Time Maude to formally model and analyze the sophisticated OGDC algorithm. We could perform all *the analyses performed by the OGDC developers using the simulation tool ns-2, as well as further analyses which are beyond the capabilities of simulation tools. To the best of our knowledge, this is the first time a formal tool has been applied to such a complex wireless sensor network algorithm.*

1 Introduction

Formal methods and tools have proved useful to give high-level yet precise descriptions of computer systems, and to analyze and experiment with system designs at early (and at different) stages of the system development process. Such analysis often discovers subtle but critical design errors that are not discovered during traditional testing. Given the increasing sophistication of wireless sensor network (WSN) algorithms—and the difficulty of modifying an algorithm once the network is deployed—there is a clear need to use formal methods to validate system *performance* and *functionality* prior to implementing such algorithms.

However, advanced WSN algorithms present a set of challenges to formal analysis tools, including:

1. Modeling and reasoning about *time-dependent* behavior; e.g.: longevity is often a crucial goal, WSN algorithms may use timers, and message transmission may be subject to message delays.
2. Many algorithms depend on geometric entities such as locations, distances, etc.
3. Modeling different forms of communication. For sensor nodes transmitting by radio, the appropriate model of communication may be broadcast where only nodes within a certain distance from the sender receive the signal with sufficient signal strength. In addition, the broadcast may be subject to transmission delays.
4. WSN algorithms often incorporate *probabilistic* behaviors.
5. Simulating and analyzing systems with a large number of sensor nodes scattered randomly.
6. Both correctness and, in particular, performance are critical aspects that must be analyzed.

Furthermore, the formalism should be intuitive and should support specifying the algorithm at an appropriate level of abstraction, so that a formal specification can be well understood and can provide a useful starting point for an implementation of the algorithm.

In this paper, we advocate the use of the language and tool Real-Time Maude [15, 16], which extends the rewriting logic-based Maude [4] tool to real-time systems, for the formal specification, simulation, and further analysis of WSN algorithms. The Real-Time Maude specification language emphasizes expressiveness. The data types of a system 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* systems, which is ideal for modeling a network system. The high-performance Real-Time Maude tool provides a range of analysis techniques, including: timed rewriting for simulation purposes; timed search for reachability analysis; and linear temporal logic model checking.

1-4244-0054-6/06/$20.00

In Real-Time Maude, geometric entities (challenge (2)) can be defined by the user as data types. Regarding (3), Real-Time Maude's flexible specification formalism allows us to easily define different forms of communication. We show how to model both unicast and 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 [1]. Nevertheless, for the purpose of *simulating* a system directly in Real-Time Maude, we show how probabilistic behaviors can be "sampled" using a pseudo-random number generator. For correctness analysis, we can model probabilistic behavior by nondeterminism as explained in Section 4. Regarding (5), we show how we can easily define states with any given number of nodes scattered pseudo-randomly. Finally, system correctness and performance can be analyzed by Real-Time Maude as illustrated in this paper.

Real-Time Maude has been used to model and analyze a set of advanced real-time systems, such as large communication protocols [17] and scheduling algorithms [13], that are beyond the capabilities of automaton-based tools. Such analysis has found subtle design errors not uncovered during traditional simulation and testing. Furthermore, some of the designers of the AER/NCA protocol suite told us that they found that rewrite rules were much more intuitive and helpful to network engineers than their informal use-case descriptions of the protocols [17].

We therefore believe that Real-Time Maude is a promising candidate for formally modeling, simulating, and analyzing WSN algorithms. On the one hand, Real-Time Maude offers an alternative to informal specifications and testing on simulation tools by:

- providing a precise formal specification of the system which can be simulated and tested directly;
- allowing the specification to be analyzed in many different ways, not just by simulating a few behaviors of the system, but by exhaustively exploring a wide range of different scenarios; and
- allowing the user to define the appropriate forms of communication at a high level of abstraction, instead of having to use a fixed set of communication primitives.

On the other hand, the most popular *formal* tools for real-time systems are the timed/hybrid automaton-based tools Uppaal [2] and HyTech [7]. To ensure that crucial properties are decidable, the specification formalisms of these tools are quite restrictive. Real-Time Maude differs from such tools by having a much more expressive specification language which supports well the specification of "infinite-control" systems with user-definable data types, different communication models, and advanced object-oriented features. The tool IF [3] extends timed automaton techniques with UML-inspired constructions for modeling objects and communication and with some notion of data type. Real-Time Maude differs from IF not only with its flexible communication model, but also with its simplicity: A simple and intuitive formalism is used to specify both data types (by *equations*) and dynamic and real-time aspects (by *rewrite rules*).

Jennifer Hou recently suggested to us the *optimal geographical density control algorithm* (OGDC) [19] for WSNs as a challenging modeling and analysis task. The OGDC algorithm is a sophisticated state-of-the-art algorithm that tries to maintain complete sensing coverage and connectivity of an area for as long as possible by switching nodes on and off. It has been simulated in the simulation tool ns-2 [12, 5], where its performance was compared to similar algorithms. OGDC presents all the challenges (1) to (6) above, as explained in [18, 19]. The model of communication is geographically bounded broadcast with transmission delay.

We have modeled, simulated, and analyzed OGDC in Real-Time Maude [18]. 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. 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. Based on communication with Jennifer Hou, it seems that our modeling and analysis effort took significantly shorter time than the simulation effort reported in [19].

This paper gives an introduction into how WSN algorithms can be modeled and analyzed in Real-Time Maude. We focus on how to specify the communication model described above and locations and distances; on how to define large initial states; and on simulating probabilistic behaviors (Section 4). Section 5 summarizes our specification and analysis effort of OGDC.

2 Related Work

Although the need for formal analysis of WSN algorithms has been been pointed out in many papers, our work on OGDC represents—to the best of our knowledge—the first formal modeling and analysis of such a sophisticated WSN algorithm as OGDC.

Some attempts at using formal methods on WSNs

have focused on modeling TinyOS using automaton-based formalisms. In [6], TinyOS is modeled in detail as a hybrid automaton, and a network of sensor nodes is modeled as a network of hybrid automata. We focus on modeling the algorithm at the appropriate level of abstraction, and we therefore model sensor nodes as abstractly as possible. In the OGDC case, there is no need to model TinyOS. In addition, it is not clear that the restrictive hybrid automaton formalism can model more sophisticated WSN algorithms than the one in [6].

In [11], the authors use Lamport's *temporal logic of actions* [8] to model and simulate diffusion protocols for discovering routing trees for gathering and disseminating data. Their analysis focuses on the number of edges in the resulting routing trees. Therefore, their protocols and analyses are not very "wireless sensor network-specific," and they do not need to model sensor nodes in any detail. For example, time and time-dependent behavior are not modeled.

In recent work, Luo and Tsai develop a new formal model, called *space time Petri nets* (STPNs), to model WSNs [9]. STPNs extend *timed* Petri nets with *locations* associated to places, and with new kinds of language constructions, such as *broadcast transitions*. In our view, the formalism, despite being a graphical formalism, is not very intuitive. Furthermore, the lack of data types will make it almost impossible to model advanced algorithms. For example, it does not seem possible to model coverage areas, angles, etc., that are all needed in the OGDC algorithm. Finally, the model is inflexible since specific forms of communication are built in as language primitives. For other models of communication, such as relating the delay of a each message in a broadcast with the distance to the destination, this formalism probably cannot be used.

3 Real-Time Maude

A Real-Time Maude *timed module* specifies a *real-time rewrite theory* (Σ, E, IR, TR), where:

- (Σ, E) is a *membership equational logic* [10] theory with Σ a signature[1] and E a set of conditional equations. The theory (Σ, E) specifies the system's state space as an algebraic data type. (Σ, E) must contain a specification of a sort `Time` modeling the (discrete or dense) time domain.
- 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 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.
- TR is a set of *tick (rewrite) rules*, written

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

  that model the elapse of time in a system. `{_}` 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 specifications. The form of the tick rules then ensures uniform time elapse in all parts of the system.

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 [16]) 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, the state has the form `{`t`}`, where t is a term of the built-in sort `Configuration`. It has typically the structure of a *multiset* made up of objects, ripe messages, and delayed messages, where multiset union is denoted by juxtaposition. 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 >
        =>
         < 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`. The transitions have the effect of altering the attribute `a1` of the object `O` and of sending a new message `m'(O')` with delay `x`. "Irrelevant" attributes (such as `a3`, and the *right-hand side occurrence* of `a2`) need not be mentioned in a rule. There is typically only one tick rule, which usually has the form

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

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

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 action must take place. These functions distribute over the elements in a configuration and must be defined for single objects. The tick rule advances time nondeterministically by *any* amount `T` less than or equal to `mte(C)`. Before executing the system, a *time sampling strategy* guiding the application of such tick rules must be defined (see Section 5).

Timed modules are *executable* under reasonable assumptions. Real-Time Maude provides a spectrum of analysis commands, including:

- *timed "fair" rewriting*, which simulates *one* behavior of the system from a given initial state *up to a given duration*;
- *timed search*, which 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*;
- (time-bounded) *linear temporal logic model checking*, which checks whether each behavior "up to a certain time" satisfies a temporal logic formula;
- the `find latest` command, which finds how long it takes, in the worst case, to reach a desired state.

4 Modeling Wireless Sensor Networks in Real-Time Maude

This section presents some techniques for modeling typical WSN features such as locations, broadcast communication, probabilistic behavior, timers, etc., in Real-Time Maude. We also show how large initial states can be defined. These techniques are used in the OGDC case study.

Locations. If the sensor nodes are located on a two-dimensional surface, we can represent a location as a pair $x\,.\,y$ of rational numbers x and y. Using the built-in sort `Rat` of rational numbers, such pairs can be represented as terms of the following sort `Location`:[2]

```
sort Location .  op _._ : Rat Rat -> Location .
```

The following function defines the *square* of the distance between two locations:[3]

```
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')) .
```

[2]Underbars in the declarations of operators such as `_._` denote the places of arguments for "mix-fix" function symbols.

[3]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.

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

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

Modeling Sensor Nodes. A sensor node can be represented as an object of a class `WSNode`. A sensor node usually does not have an explicit identifier, but can be identified by its location. We let object identifiers be locations by giving the subsort declaration `subsort Location < Oid .`

The attributes of a sensor node depend on the algorithm to be modeled. For illustration purposes, we assume that we have an attribute `remainingPower` denoting the remaining power in the node, an attribute `status` of a sort `OnOff` with values `on` and `off` which denotes whether the node is switched off or on, and two timers `timer1` and `timer2`. A timer is modeled as an attribute of the built-in sort `TimeInf` which adds the infinity value `INF` to the time domain. The value of a timer attribute denotes the time remaining until the timer expires (a timer with value `INF` is turned off). In this case, the class `WSNode` could be declared as follows:

```
class WSNode | remainingPower : Rat, status : OnOff,
               timer1 : TimeInf, timer2 : TimeInf .

sort OnOff .   ops on off : -> OnOff .
```

We need to define the behavior in time for `WSNode`s; that is, we need to define the functions δ and `mte` on such objects. The function δ decreases the remaining power and the timer values with the elapse of time:

```
var L : Oid .   var S : OnOff .   var T : Time .
vars TI TI' : TimeInf .           var R : Rat .

eq δ(< L : WSNode | remainingPower : R, status : S,
                    timer1 : TI, timer2 : TI' >, T) =
     < L : WSNode | remainingPower :
                     if S == on then
                        R monus (idlePower * T)
                     else R monus (sleepPower * T) fi,
                    timer1 : TI monus T,
                    timer2 : TI' monus T > .
```

The constants `idlePower` and `sleepPower` denote the amount of power the node consumes per time unit when the node is, respectively, active and inactive. The function `monus` is defined by x `monus` $y = \max(x - y, 0)$. The function `mte` should be defined so that it does not allow time to advance past the moment when some action must be taken. Typically, we define `mte` to allow time to elapse until the next timer expires (or until the power supply is exhausted):

```
eq mte(< L : WSNode | remainingPower : R, status : S,
                      timer1 : TI, timer2 : TI' >) =
     if S == on then min(TI, TI', R / idlePower)
     else min(TI, TI') fi .
```

Probabilistic Behaviors. The OGDC algorithm exhibits probabilistic behaviors in that (i) some actions are performed with probability p, and (ii) some values are supposed to be set to "random values, drawn from a uniform distribution ..." As mentioned, Real-Time Maude does not provide explicit support for specifying probabilistic behavior. Instead, for simulation purposes, we define a function `random`, which generates a sequence of numbers pseudo-randomly, and add to the state an object of a class `RandomNGen` with an attribute `seed` which stores the ever-changing "seed" to `random`. Probabilistic behaviors can then be modeled by "sampling" a value from the given interval using the `random` function. The disadvantage with this approach is that the Real-Time Maude specification no longer correctly specifies the informal algorithm and that all possible behaviors of the system can no longer be explored. For the purpose of specifying all possible behaviors, we can model probabilistic behavior by nondeterministic behavior by (i) allowing a probabilistic action to be performed as long as the probability of it being performed is greater than 0, and (ii) by letting the "random" value be a new variable, only occurring in the *right-hand* side of the rewrite rule, which can be given any value in the desired interval (and which makes the rule nonexecutable).

Modeling Communication. Should communication in WSNs be modeled as broadcast or unicast (or both)? Should transmission delays be modeled? If so, should the delay be a function of the distance between sender and receiver? Should power consumed by transmitting a packet be modeled? Should packet collisions be taken into account?

The answer to these questions differs from algorithm to algorithm, depending on their focus and level of abstraction. On the one hand, for ease of specification and analysis, it is important to abstract from as much detail as possible. On the other hand, essential functionality and assumptions of the algorithm must be captured in the model. It is the problem at hand that should guide the formalization—not the specification formalism or the simulation tool. Real-Time Maude provides a flexible formalism in which many different forms of communication can easily be defined.

In the OGDC algorithm, the informal description of the algorithm given in [19] says that nodes *broadcast* messages within the radio range. (Furthermore, a node does not know its neighbors.) Most time related parameters in OGDC are set according to the transmission time of a message, which is a clear indication that transmission delays must be captured in the model. In OGDC, the transmission delay does not depend on the distance between sender and receiver. We have *not* modeled packet collisions.

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" `broadcast` m `from` l, where m is the message content, into the configuration. This broadcast message is defined to be *equivalent* to a set of single 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 defines 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 `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:

```
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 .
```

If the transmission delay between two nodes l and l' is a function of the distance between them, say $f(l, l')$, we can just replace Δ with $f(\mathtt{L}, \mathtt{L'})$ in the last equation.

Section 5.2 gives examples of rules for, respectively, broadcasting and receiving a packet in this setting.

Defining Initial States. To simulate large sensor networks with different initial seeds, we define a function `genInitConf`, where `genInitConf(`n`,`*seed*`)` defines a configuration with n sensor nodes scattered at

pseudo-random locations within the sensing area, as well as a `RandomNGen` object. (An initial state must also add the operator `{_}`.) We can therefore easily generate initial states with any number of nodes, and place them in different locations, by just changing the parameters n and *seed*. In the definition below we assume that the initial values of the attributes `status`, `powerRemaining`, `timer1`, and `timer2` are, respectively, `on`, P, $t1$, and $t2$.

```
op genInitConf : Nat Nat -> Configuration .

ceq genInitConf(N, M) =
     if N == 0 then   --- generate RandomNGen object:
      < Random : RandomNGen | seed : M >
     else   --- more nodes to generate:
      < L : WSNode | status : on, powerRemaining : P,
                     timer1 : t1, timer2 : t2 >
      --- and generate the remaining N-1 nodes:
      genInitConf(N - 1, random(random(M)))
     fi
    if L := ((random(M) rem (Xsize + 1)) .
             (random(random(M)) rem (Ysize + 1))) .
```

5 Modeling and Analyzing the OGDC Algorithm in Real-Time Maude

This section gives a brief overview over how we have modeled the OGDC algorithm using the specification techniques described in Section 4, and how we could simulate and further analyze the algorithm in Real-Time Maude. Details are given in [18].

5.1 Overview of the OGDC Algorithm

The OGDC algorithm [19] is a state-of-the-art *density control* algorithm developed by Zhang and Hou. It aims at maintaining *sensing coverage* and *connectivity* of the sensing area for as long as possible by periodically selecting nodes to be *active* and *inactive*.

The network lifetime is divided into *rounds*, where each round is divided into a *node selection phase* and a *steady state phase*. The node selection phase begins with each node having status "undecided" and *probabilistically* choosing 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*. When an "undecided" 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, it sets its backoff timer depending on how close the node is to the *optimal position* w.r.t. the nodes that are currently active. The timer value is set to a gradually larger value as the *distance increases* and the *direction deviates*. When the backoff timer of a node expires, the node becomes active and broadcasts a power-on message that may cause other nodes to reset their backoff timers or to become inactive. 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.

5.2 Modeling the OGDC Algorithm

The brief description of the OGDC algorithm should make it clear that its modeling is a challenging task. Features such as coverage areas, angles, probabilistic behaviors, and power consumption have to be modeled. Real-Time Maude provides the user the flexibility to model any (computable) data type as an equational specification, allowing us to define the above features in a fairly intuitive way (see [18]). The dynamic behavior of OGDC is specified by 11 rewrite rules.

The following rule models the case where 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 then broadcasts a power-on message that contains the node's location and a *random direction*:[4]

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

The following rule models the case when an `undecided` sensor node receives a power-on message and is within distance $2\,r_s$ from the sender. The node adds the sender to its neighbor list, and checks whether all its neighbors' coverage disks completely cover the node's own coverage disk (modeled by the `bitmap` attribute). If so, the node sets its status to `off`:

```
crl [recPowerOnMsgAndSwitchOff] :
    (msg (powerOnWithDirection D) from L' to L)
    < L : WSNode | status : undecided,
                   neighbors : NBS, bitmap : BM >
   =>
    < L : WSNode | status : off, neighbors : NBS L',
                   bitmap : updateBitmap(L, BM, L'),
                   backoffTimer : INF >
   if (L withinTwiceTheSensingRangeOf L')
      /\ sensingAreaCovered(updateBitmap(L, BM, L')) .
```

[4] The direction is a parameter of the power-on message; its name does not imply that *directed* broadcast is used.

5.3 Simulation and Formal Analysis

We have subjected the OGDC algorithm to the following kinds of formal analyses in Real-Time Maude:

- *Monte Carlo simulation*, where probabilistic behavior is simulated using our pseudo-random number generator, using *timed fair rewriting*.
- Time-bounded *reachability analysis* and *temporal logic model checking* of all possible behaviors from an initial state *with respect to the values generated by the* `random` *function*. That is, our analysis is *incomplete* since we only analyze those behaviors that can take place with the specific choice of pseudo-random numbers used to simulate probabilistic behaviors. Nevertheless, such analysis covers *many* different behaviors from a given state.

As mentioned in Section 3, a *time sampling strategy* must be chosen before the analysis can take place. Since all events in the OGDC algorithm happen at specific times, we have shown in [14] that we can "fast forward" between these events without losing any interesting behaviors. Therefore, in our analysis, we use the *maximal* time sampling strategy which advances time as much as possible (as defined by `mte`).

Simulation Using Timed Rewriting. In [19], Zhang and Hou use the simulation tool ns-2 [12] with the wireless extension [5] to simulate OGDC and measure the following *performance metrics*:

- The number of *active* nodes and the percentage of coverage they provide at the end of the first round.
- The percentage of coverage and the total amount of remaining power for the whole system throughout the network's lifetime.
- The total time during which at least α percent of the sensing area is covered.

We cannot use Real-Time Maude's timed rewrite command *directly* to perform the corresponding analyses, since these performance metrics should be measured at different points in time *throughout* the lifetime of the system, and since the metrics themselves do not appear explicitly in the state.[5] Therefore, we add to the initial state a new construction called *analysis message*. An analysis "message" is defined so that, at the same time in each round of the algorithm, it computes the appropriate performance metric of the current state and stores the value in a list. The analysis message remains in the state throughout the execution and can be reviewed afterwards. We have defined in [19] three analysis messages: `activeNodes`, which records the number of active nodes in the state; `coverage%`, which records the percentage of the sensing area that is covered by the active nodes; and `totalPower`, which records the amount of power remaining in the system.

The following timed fair rewrite simulates 50 rounds of the algorithm (`in time <= roundTime * 50`) with 75 nodes in a $25m \times 25m$ sensing area. The initial state also contains the analysis messages `coverage%` and `totalPower` that compute their metrics at time `roundTime` $-\,1$, i.e., just before the end of each round:[6]

```
Maude> (tfrew {genInitConf(75, 313)
               dly(coverage%(nil), roundTime - 1)
               dly(totalPower(nil), roundTime - 1)}
           in time <= roundTime * 50 .)

Result ClockedSystem :
  {dly(coverage%(100 ++ ... ++ 100 ++ 92 ++ 100 ++ 99
                 ++ ... ++ 51 ++ 0 ++ ... ++ 0), 999999)
   dly(totalPower(146337556845 ++ 140676548774
                  ++ ... ++ 0 ++ ... ++ 0), 999999)
   ... } in time 50000000
```

The result messages show that the nodes can provide 100% coverage for 25 rounds, with a decrease of coverage in certain intermediate rounds. We have also simulated one round of the protocol with up to 400 sensor nodes to measure the number of active nodes [18].

Further Formal Analysis. Due to the large states involved, we restricted our search and model checking analyses to systems with 5 to 6 nodes in a $15m \times 15m$ sensing area. Executing the `find latest` command with many different initial states, we found that the system will reach the steady state phase in *at most* 1647 ms. One round of the OGDC algorithm is 1000 *seconds*, which means that the network spends most of its lifetime performing its sensing task.

We used Real-Time Maude's *temporal logic model checker* to check whether the system remains in the steady state phase throughout the first round once it has reached this phase. That was indeed the case. Finally, we used time-bounded search to show that the entire sensing area is covered in the steady state phase in the first round (see [18] for details).

Summary of the Analysis. Using analysis messages, rewriting could simulate the OGDC algorithm with several hundred sensor nodes and could measure *all* performance metrics measured by the OGDC developers using ns-2. We generally got a larger number of active nodes than reported in [19], and, consequently

[5] In principle, one *could* use Real-Time Maude's *tracing* capabilities to trace the state at these different points in an execution, but this is not practical, given the large states and the large number of rewrites involved.

[6] Parts of the Real-Time Maude output are replaced by '...'.

we got better coverage and shorter network lifetime. One explanation could be the location of the sensor nodes. A more plausible explanation is the following: In OGDC, if two nodes are fairly close to one another, the difference between their backoff timer values is often smaller than the transmission delay. If transmission delays are ignored during the simulations, potentially because the simulation tool makes it inconvenient to simulate transmission delays, then only one of the neighbors will become active. However, if, as in our case, we capture transmission delays, then the backoff timer of the "worse" node will expire *before* it receives the power-on message from the "better" node, and, hence, *both* nodes will become active.

We have specified coverage areas as "bitmaps," and have emphasized ease and elegance over computational efficiency when defining bitmaps and functions on bitmaps. A more sophisticated representation of coverage areas should allow model checking systems with much more than six nodes.

6 Concluding Remarks

We have proposed the general-purpose high-performance Real-Time Maude tool as a formal tool for modeling, simulating, and further analyzing advanced wireless sensor network algorithms. Real-Time Maude emphasizes generality and ease of specification (at the expense of decidability of key properties), and allows us to specify advanced systems, with various data types and communication forms, in an intuitive and uniform formalism that has proved useful to network engineers.

We have tested our tool on the challenging OGDC density control algorithm. Our model is essentially "just" a formalization of OGDC at the level of abstraction of its informal specification. We have been able to perform all the analyses performed using ns-2 in [19], as well as additional analyses of many behaviors from an initial state. We have good reasons to believe that modeling and simulating OGDC in Real-Time Maude require significantly less effort than implementing and simulating the algorithm on a simulation tool.

Much work remains. We cannot yet model and analyze probabilistic behaviors as such, although we can do Monte Carlo simulations using pseudo-random numbers. It would be useful to further validate Real-Time Maude as a simulation tool by showing that differences in performance between two algorithms as measured by a simulation tool carry over to their Real-Time Maude simulations. Finally, an important challenge consists of making network algorithm developers use our tool.

Acknowledgments. We are grateful to Jennifer Hou for suggesting the OGDC algorithm as a challenging modeling task and to José Meseguer for discussions on modeling communication in sensor networks.

References

[1] G. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based specification language for probabilistic object systems. In *QAPL'05*, 2005.

[2] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In *SFM-RT 2004*, volume 3185 of *LNCS*. Springer, 2004.

[3] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF toolset. In *SFM'04*, volume 3185 of *LNCS*. Springer, 2004.

[4] M. Clavel, F. Dúran, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude Manual*, April 2005. `http://maude.cs.uiuc.edu`.

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

[6] S. Coleri, M. Ergen, and T. J. Koo. Lifetime analysis of a sensor network with hybrid automata modelling. In *WSNA '02*. ACM, 2002.

[7] 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.

[8] L. Lamport. The temporal logic of actions. *ACM Toplas*, 16(3):872–923, 1994.

[9] Y. Luo and J. J. P. Tsai. A graphical simulation system for modeling and analysis of sensor networks. In *Intl. Symposium on Multimedia*, 2005. To appear.

[10] J. Meseguer. Membership algebra as a logical framework for equational specification. In *Proc. WADT'97*, volume 1376 of *LNCS*. Springer, 1998.

[11] S. Nair and R. Cardell-Oliver. Formal specification and analysis of performance variation in sensor network diffusion protocols. In *MSWiM '04*. ACM, 2004.

[12] ns-2 network simulator. `www.isi.edu/nsnam/ns`.

[13] P. C. Ölveczky and M. Caccamo. Formal simulation and analysis of the CASH scheduling algorithm in Real-Time Maude. In *FASE'06*, 2006. To appear.

[14] P. C. Ölveczky and J. Meseguer. Abstraction and completeness for Real-Time Maude. In *WRLA'06*.

[15] P. C. Ölveczky and J. Meseguer. Specification and analysis of real-time systems using Real-Time Maude. In *FASE 2004*, volume 2984 of *LNCS*. Springer, 2004.

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

[17] P. C. Ölveczky, J. Meseguer, and C. L. Talcott. Specification and analysis of the AER/NCA active network protocol suite in Real-Time Maude. `http://www.ifi.uio.no/RealTimeMaude`, 2004.

[18] 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`, 2005.

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