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

(Invited Paper)

Peter Csaba Ölveczky
Department of Informatics, University of Oslo
Email: peterol@ifi.uio.no

Stian Thorvaldsen
Department of Informatics, University of Oslo
Email: 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 analyze the sophisticated OGDC algorithm. 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. We have modeled the OGDC algorithm in Real-Time Maude at a suitable level of abstraction, and 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. Furthermore, we believe that modeling and simulating the OGDC algorithm in Real-Time Maude require significantly less effort than implementing it on a simulation tool.

This paper shows how typical features of wireless sensor networks can be modeled in Real-Time Maude, and briefly summarizes our modeling and analysis of the OGDC algorithm.

I. 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 in the system development process. Such analysis often discovers subtle but critical design errors that may not be discovered during traditional testing. Given the increasing sophistication of wireless sensor network 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.

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

1) 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.

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. The details that need to be modeled depend on the algorithm and its level of abstraction.

4) Wireless sensor network algorithms often incorporate probabilistic behaviors.

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

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 [1], [2], which extends the rewriting logic-based Maude [3] tool to real-time systems, for the formal specification, simulation, and further analysis of wireless sensor network algorithms. The Real-Time Maude specification language emphasizes expressiveness. The data types of a system are defined in a “functional programming style” 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 state space exploration; and linear temporal logic model checking.

In Real-Time Maude, geometric entities (challenge (2)) can be defined by the user as 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 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 [4]. 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 IV-C. 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 [5], [6] and scheduling algorithms [7] that are beyond the capabilities of automaton-based tools. Apart from finding subtle design errors not uncovered during traditional simulation and testing, some of the designers of the AER/NCA protocol suite [8] 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 [5].

We therefore believe that Real-Time Maude is a promising candidate for formally modeling, simulating, and analyzing wireless sensor network algorithms. On the one hand, Real-Time Maude offers an alternative to informal specifications and testing on simulation tools such as ns-2 [9] with the wireless extension [10], GloMoSim [11], and JIST [12], 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 [13] and HyTech [14]. To ensure that crucial properties are decidable, the specification formalisms of such automaton-based tools are quite restrictive. Real-Time Maude complements 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 [15] extends timed automaton techniques with UML-inspired constructions for modeling objects and communication and with some notion of data type. Real-Time Maude complements 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).

To investigate the suitability of analyzing wireless sensor network algorithms using Real-Time Maude, Jennifer Hou recently suggested to us the optimal geographical density control algorithm (OGDC) [16] for wireless sensor networks 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, where its performance was compared to the performance of similar algorithms. OGDC presents all the challenges (1) to (6) above, as explained in [16], [17]. In addition, it requires modeling and computing with coverage areas. The model of communication is geographically bounded broadcast with transmission delay.

We have modeled, simulated, and analyzed OGDC in Real-Time Maude [17]. 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 also 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 [16]. Formalizing the algorithm at a fairly high level of abstraction in Real-Time Maude should be a much less time-consuming task than implementing it on a simulation tool. We included transmission delays in our model and thereby discovered an (insignificant) weakness in the algorithm. As for the results of our analyses, our simulations generally resulted in a larger number of active nodes (and, hence, better coverage and shorter lifetime) than the ns-2 simulation results reported in [16]. Possible reasons for this difference are discussed in Section V-C.3.

This paper gives an introduction into how wireless sensor network algorithms can be modeled and analyzed in Real-Time Maude. In particular, we focus on how to specify the communication model described above and locations and distances; how to define large initial states; and on simulating probabilistic behaviors (Section IV). Section V summarizes our specification and analysis effort of OGDC.

II. RELATED WORK

Although the need for formal analysis of wireless sensor network 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 wireless sensor network algorithm as OGDC.

Some attempts at using formal methods on wireless sensor networks have focused on modeling TinyOS using automaton-based formalisms. In [18], TinyOS is modeled in detail as a hybrid automaton. Results from power analysis of one sensor node are used to model a network of sensor nodes as a network of hybrid automata in order to analyze the power dissipation over the network. In our work, 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. As for networks of hybrid automata, it is not clear that the restrictive hybrid automaton formalism can model more sophisticated wireless sensor network algorithms than the one in [18].
In [19], the authors use Lamport’s temporal logic of actions [20] 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 depth. 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 wireless sensor networks [21]. STPNs extend timed Petri nets with locations associated to places, and with new kinds of language constructions, such as broadcast transitions. In our view, one problem with the formalism is that it, 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.

III. REAL-TIME MAUDE

A Real-Time Maude timed module specifies a real-time rewrite theory [22] of the form \((\Sigma, E, IR, \tau R)\), where:

- \((\Sigma, E)\) is a membership equational logic [23] theory with \(\Sigma\) a signature\(^1\) and \(E\) a set of conditional equations. The theory \((\Sigma, E)\) specifies the system’s state space as an algebraic data type. \((\Sigma, E)\) must contain a specification of a sort Time modeling the time domain (which may be dense or discrete).

- IR is a collection of labeled conditional instantaneous rewrite rules specifying the system’s instantaneous (i.e., zero-time) local transitions, each of which is written \(\text{crl} [l] : t \rightarrow t’ \text{ if } \text{cond}\), 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 \(E\).\(^2\)

- \(\tau R\) is a set of tick (rewrite) rules, written with syntax \(\text{crl} [l] : (t) \rightarrow (t’) \text{ in time } \tau \text{ if } \text{cond}\). that model the elapse of time in a system. \(\_\) is a built-in constructor of sort \(\text{GlobalSystem}\), and \(\tau\) is a term of sort \(\text{Time}\) that denotes the duration of the rewrite.

The initial states must be ground terms of sort \(\text{GlobalSystem}\) and must be reducible to terms of the form \((t)\) using the equations in the specifications. The

1That is, \(\Sigma\) is a set of declarations of sorts, subsorts, and function symbols (or operators).

2The set \(E\) of equations is a union \(E’ \cup A\), where \(A\) is a set of equational axioms such as associativity, commutativity, and identity, so that deduction is performed modulo \(A\). Operationally, a term is reduced to its \(E’\)-normal form modulo \(A\) before any rewrite rule is applied.

form of the tick rules then ensures uniform time elapse in all parts of the system.

In object-oriented timed modules, a class declaration

\[
\text{class } C & \mid \text{ att}_1 : s_1, \ldots, \text{ att}_n : s_n .
\]

declares a class \(C\) with attributes \(\text{att}_1\) to \(\text{att}_n\) of sorts \(s_1\) to \(s_n\). An object of class \(C\) in a given state is represented as a term \(<O : C \mid \text{ att}_1 : \text{val}_1, \ldots, \text{ att}_n : \text{val}_n >\) where \(O\) (of sort \(\text{Oid}\)) is the object’s identifier, and where \(\text{val}_1\) to \(\text{val}_n\) are the current values of the attributes \(\text{att}_1\) to \(\text{att}_n\).

A message \(m\) is a term of sort \(\text{Msg}\). We can easily define delayed messages (see [2]) as terms of the form \(\text{dly} (m, \tau)\), which denotes a message \(m\) that will be “ripe” in time \(\tau\) (that is, it will become \(m\) in time \(\tau\)). In a concurrent object-oriented system, the state has the form \((t)\), where \(t\) is a term of the built-in sort \(\text{Configuration}\). It 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

\[
\text{rl} [1] : m (O, w) \\
< O : C \mid a_1 : x, a_2 : O’, a_3 : z > \\
\rightarrow \\
< O : C \mid a_1 : x + w, a_2 : O’, a_3 : z > \\
\text{dly} (m’ (O’), x).
\]

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 \(a_1\) of the object \(O\) and of sending a new message \(m’ (O’)\) with delay \(x\). Attributes, such as \(a_3\), whose values do not change and do not affect the next state of other attributes, need not be mentioned in a rule. Attributes, like \(a_2\), whose values influence the next state but are themselves unchanged, may be omitted from right-hand sides of rules. There is typically only one tick rule, which usually has the form

\[
\text{var } C : \text{Configuration} . \quad \text{var } T : \text{Time} . \\
\text{crl} [\text{tick}] : (C) \Rightarrow (\delta (C, T)) \text{ in time } T \\
\text{if } T <= \text{mtime}(C) \text{ [nonexec]}.
\]

The function \(\delta\) defines the effect of time elapse on the objects and messages in a configuration, and the function mtime 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 \(\text{mtime}(C)\). Before executing the system, a time sampling strategy guiding the application of such tick rules must be defined (see Section V).

Timed modules are executable under reasonable assumptions, and Real-Time Maude provides a spectrum of analysis capabilities, some of which are summarized below.

Real-Time Maude’s timed “fair” rewrite command simulates one behavior of the system up to a certain duration. It
is written with syntax

\[(t \text{search } 1 \text{ in time } \leq \tau .)\]

where \(t\) is the initial state and \(\tau\) 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 and match a search pattern and satisfy a search condition. The command which searches for one state satisfying the search criteria has syntax

\[(t \text{search } 1 \text{ in time } \leq \tau .)\]

Real-Time Maude extends Maude's linear temporal logic model checker [3] to check whether each behavior “up to a certain time,” as explained in [2], satisfies a temporal logic formula. Finally, the find latest command finds how long it takes, in the worst case, to reach a desired state.

IV. MODELING WIRELESS SENSOR NETWORKS IN REAL-TIME MAUDE

This section presents some techniques for modeling typical wireless sensor network 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.

A. 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 \(\text{Rat}\) of rational numbers, such pairs can be represented as terms of the following sort Location:

\[
\text{sort Location. op } \_._: \text{Rat Rat} \rightarrow \text{Location.}
\]

The following function defines the \(\text{square}\) of the distance between two locations:

\[
\text{op distanceSq : Location Location } \rightarrow \text{Rat.}
\]

\[
\text{vars X X' Y Y' : Rat. eq distanceSq(X , Y , X' , Y) = } \((X - X') * (X - X') + (Y - Y') * (Y - Y')\).
\]

Given a constant \(\text{transRange}\) denoting the transmission range of a sensor node, we can check whether a sensor node is within the transmission range of another sensor node:

\[
\text{vars L L' : Location. op } \_\text{withinTransRangeOf}_.: \text{Location Location } \rightarrow \text{Bool. eq L withinTransRangeOf L' = distanceSq(L, L') \leq \text{transRange} * \text{transRange}.}
\]

B. Modeling Sensor Nodes

Each sensor node can suitably be represented as an object of a class \(\text{WSNode}\). 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

\[
\text{subsort Location < Oid.}
\]

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

\[
\text{class WSNode | remainingPower : Rat, status : OnOff, timer1 : TimeInf, timer2 : TimeInf.}
\]

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

\[
\text{eq mte( L : WSNode | remainingPower : R, status : S, timer1 : TI, timer2 : TI', T) =}
\]

\[
\text{if S == on then R monus (idlePower * T)}
\]

\[
\text{else R monus (sleepPower * T)}\text{ if, timer1: TI monus T, timer2: TI' monus T ).}
\]

C. 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 use the following function \(\text{random}\), which generates a sequence of numbers pseudo-randomly and satisfies the criteria of a “good” random function given in [24]:

\[
\text{op random : Nat } \rightarrow \text{Nat. eq random(N) = (104 * N) + 7921.}
\]

The state of the system will also contain an object of a class \text{RandomNGen} with an attribute \text{seed} which stores the ever-
changing “seed” to random. A rule of the following form can model the case where state $t$ goes to $t_1$ with probability $p$, and to $t_2$ with probability $1 - p$ (rem is the remainder function):

$$\text{rl} \text{ if } (\text{random}(N) \text{ rem } 100) + 1 \leftrightarrow p * 100 \text{ then } t_1 \text{ else } t_2 \text{ fi}$$

Probabilistic behavior of kind (ii) above can 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 replace 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).

**D. Modeling Communication in Wireless Sensor Networks**

Should communication in wireless sensor networks be modeled as broadcast or as unicast (or both)? Should transmission delays be modeled? If so, should the transmission delay be a constant or a function of the distance between sender and receiver? Should power consumed by transmitting (or receiving) 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 generality and 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. For example, if the algorithm to be analyzed takes transmission delays into account, our model should reflect that, or the results from the analysis may be misleading. Real-Time Maude provides a flexible specification formalism in which many different forms of communication can easily be defined.

In the OGDC algorithm, the informal description of the algorithm given in [16] 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 $\Delta$. The idea is that the sender $l$ sends a “broadcast message” of the form $\text{broadcast}_m$ 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 $\text{dly}(\text{msg } m \text{ from } l \text{ to } l', \Delta)$ with delay $\Delta$, one for each sensor node $l'$ within the radio range of $l$. The messages are declared as follows:

```plaintext
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:

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

```plaintext
op distributeMsg : Location MsgCont Configuration
                    -> Configuration [frozen (3)].
```

In case 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 $\Delta$ with $f(l, l')$ in the last equation.

In this setting, broadcasting a message $m$ from sensor node $l$ is modeled by a rule of the form

```plaintext
rl < L : WSNode | powerRemaining : R, ... > =>
    < L : WSNode | powerRemaining : R minus tP, ... > (broadcast m from L).
```

where $tP$ is the power needed to broadcast a message. A rule modeling the reception of a message has the form

```plaintext
rl (msg m from l' to l) < L : WSNode | ... > =>
    < L : WSNode | ... > ...
```

**E. Defining Initial States**

To simulate different large sensor networks with different initial seeds, we can define a function $\text{genInitConf}$, where $\text{genInitConf}(n, seed)$ 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 $(_\_)$ around the set of objects.) We can therefore easily generate initial states with any number of sensor nodes, and place them in different locations, by just changing the parameters $n$ and $seed$ in $\text{genInitConf}$. In the definition below we assume that the initial values of...
the attributes status, powerRemaining, timer1, and timer2 are, respectively, on, \( P \), \( t_1 \), and \( t_2 \).

\[ \text{op genInitConf : Nat Nat -> Configuration .} \]

\[ \text{ceq genInitConf(N, M) = } \]

\[ \text{if N == 0 then } \]

\[ \text{--- generate RandomNGen:} \]

\[ \text{< Random : RandomNGen | seed : M >} \]

\[ \text{else } \]

\[ \text{--- more nodes to generate:} \]

\[ < L : \text{WSNode} | \text{status : on, powerRemaining : P, timer1 : t1, timer2 : t2} > \]

\[ \text{--- and generate the remaining N-1 nodes:} \]

\[ \text{genInitConf(N - 1, random(random(M)))} \]

\[ \text{fi} \]

\[ \text{if L := } \]

\[ \text{(random(M) rem (Xsize + 1)) .} \]

\[ \text{(random(random(M)) rem (Ysize + 1))) .} \]

V. 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 IV, and how we could simulate and further analyze the algorithm in Real-Time Maude. Details about these efforts are given in [17].

A. Overview of the OGDC Algorithm

The OGDC algorithm [16] is a sophisticated 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 intersection of the boundaries of the coverage areas\(^5\) of two active nodes is called a crossing. The OGDC algorithm tries to select the set of active nodes such that they provide the minimum amount of overlap while leaving no crossing uncovered. The optimal position denotes the location where a node ideally should be placed, with respect to the active nodes that are selected so far. The OGDC algorithm tries to select the node that is closest to a given optimal position.

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 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 if 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:\(^6\)

\[ \text{r1 [startingNodePowerOn] :} \]

\[ \text{< L : \text{WSNode} | remainingPower : P, backoffTimer : 0, hasVolunteered : true >} \]

\[ \text{< Random : RandomNGen | seed : M >} \]

\[ \text{==>} \]

\[ \text{< L : \text{WSNode} | remainingPower : P monus tP, backoffTimer : INF, status : on >} \]

\[ \text{< Random : RandomNGen | seed : random(M) >} \]

broadcast (powerOnWithDirection randomDirection(M)) from L.

When an undecided sensor node receives a power-on message and is within a distance of 2\( r_s \) from the sender, it adds the sender to its neighbor list, and checks whether or not all its neighbors’ coverage disks completely cover the node’s own coverage disk. If so, the node sets its status to off. This setting is modeled by the following rule:

\[ \text{crl [recPowerOnMsgAndSwitchOff] :} \]

\[ \text{(msg (powerOnWithDirection D) from L’ to L) } \]

\[ \text{< L : \text{WSNode} | status : undecided, neighbors : NBS, bitmap : BM >} \]

\[ \text{==>} \]

\[ \text{< L : \text{WSNode} | status : off, neighbors : NBS L’, bitmap : updateBitmap(L, BM, L’), backoffTimer : INF >} \]

\[ \text{if (L withinTwiceTheSensingRangeOf L’)} \]

\[ \text{\( /\) sensingAreaCovered(updateBitmap(L, BM, L’)).} \]

\(^{5}\)The coverage area of a node denotes the disk-shaped area which is within the sensing range \( r_s \) of the node.

\(^{6}\)The direction is a parameter to the power-on message; its name does not imply that directed broadcast is used.
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 some initial state with respect to the particular values generated by the pseudo-random generator. That is, our analysis is incomplete since we do not analyze all possible behaviors in a given sensor network topology, but only those behaviors that can take place with the specific choice of pseudo-random numbers used to simulate the probabilistic behavior. Nevertheless, such analysis covers many possible behaviors from a given state, and thereby either discovers subtle errors or significantly increases our confidence in the correctness of the algorithm.

As mentioned in Section III, 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 [25] 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).

1) Simulation Using Timed Rewriting: In [16], Zhang and Hou use the simulation tool ns-2 with the wireless extension to simulate OGDC and measure the following performance metrics:

- The number of active nodes and the percentage of coverage provided by those nodes 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 \( \alpha \) percent of the sensing area is covered. (This analysis can be done in the same way as the first two, and is not treated here.)

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.\(^7\) 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 [16] three analysis messages: activeNodes, which records the number of active nodes in the state; coveragePercentage, which records the percentage of the entire sensing area that is covered by the active nodes; and totalPower, which records the total amount of power remaining in the system.

2) Future Formal Analysis: The following formal analyses analyze not just one, but all, possible behaviors from an initial state, relative to the treatment of probabilistic behaviors. Due to the large states involved, we restrict such analyses to systems with 5 to 6 nodes in a \( 15m \times 15m \) sensing area.

The following find latest command finds the latest possible time the network enters the steady state phase, and thereby also finds out whether this phase is always reached within the end of the round.

\[\text{Maude} \geq \text{find latest} \]
\[
\{\text{genInitConf}(5, 1)\} \Rightarrow \{\text{C:Configuration}\}
\]
\[
\text{such that steadyStatePhase}(\text{C:Configuration}) \}
\]
\[
\text{in time} \leq \text{roundTime} .
\]

Result: \{ ... \} in time 815

The system will reach the steady state phase in at most 815 ms. Experimenting with many other initial seeds, the longest time we found was 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 [17] for details).

3) Summary of the Real-Time Maude 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. In our simulations, we generally got a larger number

---

\(^7\)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.

\(^8\)Parts of the output of Real-Time Maude executions that are omitted in the exposition for lack of space are replaced by ‘...’.
of active nodes (for the same number of nodes in the same sensing area) than reported in [16], and, consequently we got better coverage and shorter network lifetime. We do not know the reason for this discrepancy. 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 impossible or 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.

To avoid this scenario, Jennifer Hou suggested to us that the OGDC algorithm be modified by adding time stamps to the power-on messages.

We have, as suggested in an earlier version of [16], specified coverage areas as “bitmaps,” and have emphasized ease and elegance over computational efficiency when defining bitmaps and functions on bitmaps. The large bitmaps made exhaustive exploration of the reachable state space and temporal logic model checking infeasible for more than six nodes. A more sophisticated representation of coverage areas should allow model checking much larger networks.

VI. 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 in other projects.

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 [16], 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, perhaps the greatest and most important challenge consists of making wireless sensor network algorithm developers use our tool or some extension of it.

ACKNOWLEDGMENTS.

We are grateful to Jennifer Hou for suggesting the OGDC algorithm as a challenging modeling task, to José Meseguer for discussions on modeling communication in sensor networks, and to Gerardo Schneider for helpful comments on earlier versions of this paper.

REFERENCES