10-2008

Specifying and verifying sensor networks: An experiment of formal methods

Jin Song DONG

Jing SUN

Jun SUN
Singapore Management University, junsun@smu.edu.sg

Kenji TAGUCHI

Xian ZHANG

Citation

DONG, Jin Song; SUN, Jing; SUN, Jun; TAGUCHI, Kenji; and ZHANG, Xian. Specifying and verifying sensor networks: An experiment of formal methods. (2008). *Proceedings of the 10th International Conference on Formal Engineering Methods, ICFEM 2008, Kitakyushu-City, Japan, October 27-31*. 318-337.
Available at: https://ink.library.smu.edu.sg/sis_research/5049

Specifying and Verifying Sensor Networks: An Experiment of Formal Methods

Jin Song Dong[1], Jing Sun[2], Jun Sun[1,⋆], Kenji Taguchi[3], and Xian Zhang[1]

[1] School of Computing,
National University of Singapore
Tel.: +65 6516 4244; Fax: +65 6779 1610
{dongjs,sunj,zhangxi5}@comp.nus.edu.sg
[2] Department of Computer Science,
The University of Auckland
j.sun@cs.auckland.ac.nz
[3] Information Systems Architecture Research Division,
Grace Center, National Institute of Informatics

Abstract. With the development of sensor technology and electronic miniaturization, wireless sensor networks have shown a wide range of promising applications as well as challenges. Early stage sensor network analysis is critical, which allows us to reveal design errors before sensor deployment. Due to their distinguishable features, system specification and verification of sensor networks are highly non-trivial tasks. On the other hand, numerous formal theories and analysis tools have been developed in formal methods community, which may offer a systematic method for formal analysis of sensor networks. This paper presents our attempt on applying formal methods to sensor network specification/verification. An integrated notation named *Active Sensor Processes* is proposed for high-level specification. Next, we experiment formal verification techniques to reveal design flaws in sensor network applications.

1 Introduction

With the development of sensor technology and electronic miniaturization, sensor integration makes it possible to produce extremely inexpensive sensing devices. The sensors have been equipped with significant processing, memory, and wireless communication capabilities. Thus, they are capable of performing complex in-network computation besides sensing and communication. Wireless sensor networks have shown a wide range of promising applications in a variety of domains [12], e.g., environmental monitoring, acoustic detection, smart spaces, inventory tracking, etc.

In recent years, a number of sensor devices and sensor programming languages have been actively developing [1,4,20]. In the following, we review several features of ad hoc and sensor networks which distinguish them from ordinary systems. In general, sensor networks may be categorized as *event-based distributed reactive hybrid* systems.

[⋆] Corresponding author.

S. Liu, T. Maibaum, and K. Araki (Eds.): ICFEM 2008, LNCS 5256, pp. 318–337, 2008.

- The nature of sensors is event-based. Sensors' behaviors are often stated in terms of how they respond to internal/external events. Sensors may have continuous interfaces for sensing/actuating as well as discrete message input/output for inter-sensor communication.
- Sensor networks may be re-configurable, i.e., part of the behaviors may be updated dynamically. Deployed sensors may need to be updated with new programs to cope with new system requirements or reused for completely new tasks.
- Sensor network applications are unlikely interested in the state of an individual sensor. Rather, applications focus on the data generated by sensors. Sensor network nodes are data-centric. There may not be a unique global address (like IP in the Internet) associated with each sensor. The data generated at a sensor node is named by attributes and applications request data matching certain attribute values. Date-centric routing is favored over end-to-end routing.
- Sensor networks are application-aware. Traditional networks are designed to accommodate a wide variety of applications. Sensor networks are however application-specific, i.e., they are configured and deployed for a specific application. Thus, they are designed with the knowledge of the types of sensors, the geography, the data format generated by the sensors, etc.

On one hand, early stage analysis of sensor networks is critical as, once deployed, sensors may not be easily accessible and updated with corrections. The unique features of wireless sensor networks present unique challenges for specification, verification and synthesis. On the other hand, numerous formal theories (e.g., broadcasting messages [25], higher-order processes [30], mobile processes [5]) and tools (e.g., UPPAAL, SPIN) have been developed. We believe that wireless sensor networks are a fruitful application domain of formal methods, which shall provide methodologies as well as tools for systematic sensor networks specification and verification. This paper presents our initial attempt on applying formal methods to sensor networks.

In order to apply the rich set of formal methods and theories, the very first task is to construct a formal description of sensor networks. A formal system description requires construction of a high-level mathematical model of the system, which can later be used for a variety of system analysis tasks such as simulation, verification, performance evaluation and synthesis. In this paper, we adapt ideas from multiple existing formal specification languages [23,25,6] and propose a simple integrated notation for formal sensor network specification, namely *Active Sensor Processes* (shortened as ASP). ASP is based on classic process algebras with extensions solely for hybrid broadcasting systems. Next, we demonstrate how to verify sensor network applications using state-of-the-art verification techniques. Verification based on ASP (instead of concrete implementation) allows us to focus on the key aspects of the application without being disturbed by irrelevant details. For instance, it is desirable to prove the soundness of the high-level specification of a wireless routing protocol (i.e., a package will reach the destination provided that it is feasible) assuming reliable a link layer protocol. Performed early in the design stage, such modeling and analysis offers the promise of a systematic approach. We show that using existing system analysis tools, previously unknown design flaw can be revealed. Nonetheless, we show that systematic sensor network verification may require verification capability beyond existing techniques and tools.

As for related works, the research on sensor networks has been influenced by both the traditional network community and the database community. The former group tends to focus on finding efficient communicating protocol [32,17]. The latter focuses on in-network data aggregation and data querying [8,22]. Other works include building sensor hardwares and their middleware support [20]. There have also been proposals for domain specific languages which present programming models for writing sensor network programs [4,22]. We believe formal methods community can contribute to the development of sensor network systems by providing new modeling and design techniques capturing high level system behaviors. To our best knowledge, there have been few formal languages proposed for generic sensor network modeling [24,31]. The previous closest to ASP is the notion of SensorML [3], which offers a modeling language for sensor processes. Comparing to SensorML, our language models the dynamic behaviors of sensor networks. ASP has a formal semantics, which is essential for formal analysis. For instance, by translating a large subset of ASP to equivalent timed automata, we may reuse existing model checkers for formal verification. There have been many languages proposed for modeling hybrid or mobile reactive systems, e.g., Ambients, KLAIM, TCOZ, various extensions to π-calculus or automata, etc. However, applying existing formalisms may not be possible or optimal because of the unique features of sensors. For instance, the classic CSP, CCS, π-calculus and Ambient calculus rely on single communication mechanism (e.g., lock-step synchronization between processes), which is not suitable for inter-sensor communication. Naturally, there are different ways of communication in sensor networks, i.e., the sensing/actuating may be continuous rather then discrete; the messaging between sensor nodes are asynchronous as well as broadcasting; and there may be lock-step synchronization between processes running in parallel which reside at the same sensor node. Moreover, the network topology in sensor networks is highly dynamic, which depends on a lot on the geographic location of the sensor and its own characteristics like radio range. Existing formal specification languages, like π-calculus and Ambient calculus, support network dynamic reconfiguration by explicitly changing the channel names or the residing location of a process and thus are in-effective for modeling data-centric sensor network routing. Nonetheless, we believe wireless sensor networks are a fruitful domain for the scattered works on theoretical development on communication as well as process semantics (e.g., broadcasting semantics, higher-order processes and others as evidenced in [24,25]).

The remainder of the paper is organized as follows. Section 2 explains the constructs of ASP using illustrative examples. Section 3 explains the semantics of the language. Section 4.1 models and verifies (using existing tools) a network code propagation algorithm. Section 4.2 studies modeling and verification of a class of sensor network routing protocols. Section 5 concludes the paper and reviews related works and future works.

2 Specifying Sensor Networks

The objective is to construct a concise and precise model of the sensor nodes. In order to facilitate later system analysis, the language is designed to be lightweight. In this section, we present the notation named *Active Sensor Processes*. We remark that ASP

$P ::= \mathit{Stop} \mid \mathit{Skip}$	– deadlock and termination
$\mid \; \mathit{Idle}(d)$	– delay
$\mid \; e \rightarrow P$	– event-prefix
$\mid \; x := \mathit{exp} \rightarrow P$	– assignment
$\mid \; P;\ Q$	– sequential composition
$\mid \; P \,\square\, Q$	– choice
$\mid \; P \lhd b \rhd Q$	– conditional branching
$\mid \; P \triangledown Q$	– interrupt
$\mid \; P \,\vert[\, X \,]\vert\, Q$	– local synchronous composition
$\mid \; s \odot x \rightarrow P$	– sensing
$\mid \; s \otimes v \rightarrow P$	– actuating
$\mid \; c?X \rightarrow P$	– inter-sensor message input
$\mid \; c!X \rightarrow P$	– inter-sensor message output
$\mid \; P = Q$	– process referencing

Fig. 1. ASP Process Syntax

adapts features from Timed CSP [27] and TCOZ [23] and then extends it with language constructs which are dedicated to sensor networks.

For simplicity, we distinguish three types, i.e., the set of all data values $\mathcal{V}$, the set of all events $\mathcal{E}$ and the set of all processes $\mathcal{P}$. The specification of a sensor node contains two parts. One is the mapping of a set of data variables to their values. The other is the mapping of a set of process variables to process expressions. Unlike traditional CSP/CCS, processes are allowed to be re-defined dynamically so as to model sensors which are designed to be reconfigurable. This is achieved by assigning a process expression to process names dynamically, in the same way new values are assigned to data variables. Data variables identify the context of the node, which may be categorized into two groups. The first group is a set of pre-defined control variables which determines the network topology. For instance, one way to capture the network topology is through two predefined variables, namely $\mathit{location}$ and range. Variable $\mathit{location}$ identifies the location of the sensor node. Variable range identifies its radio range, which in terms identifies all connected sensor nodes together with the variable $\mathit{location}$. The second group is a set of local data variables, e.g., control variables, sensed data store, etc. A process expression is formed by the syntax summarized in Figure 1. In the following, we will briefly review the process constructs using illustrative examples. A number of process constructs introduced in CSP and Timed CSP are reused. Note that some have different semantics.

The process Skip terminates successfully. The process Stop deadlocks. A sensor node which behaves as Stop (if the battery runs out or the antenna is broken) disappears from the network since it can not communicate with the rest of the system any more. The process $\mathit{Idle}(d)$ where d is a positive real number idles for exactly d time units. A sensor node behaving as $\mathit{Idle}(d)$ may go to sleep mode or power off (to save energy). In contrast to sensors which behave as Stop, the sensor wakes up after exactly d time units or responding to some interrupts. Note that clocks, as data variables, are always local to a sensor node. The process $e \rightarrow P$ is called event-prefixing. It engages e and then behaves as P, where e is an abstract event. If the event is shared by multiple processes running in parallel residing at the same sensor node, the event acts like a synchronization barrier. Local variables or processes may be updated by assignments of the form $x := exp$ where exp is an expression.

Diversity of behaviors are specified using choices, which are often guarded with events or Boolean conditions. Process $a \rightarrow P \ \Box\ b \rightarrow Q$ will proceed as specified by P if a is firstly engaged. A conditional branch is written as $P \lhd b \rhd Q$, where b is a Boolean expression over the sensor's context. It behaves as specified by P if b evaluates to true, otherwise it behaves as specified by Q. We view interrupt as a biased choice. The process $P \triangledown e \rightarrow Q$ behaves as P until the moment e is engaged and then behaves as Q afterward. Alphabetized parallel composition is written as $P \,|[\, X \,]|\, Q$, where X is a set of events. Events in X must be synchronized by P and Q. X is omitted if it is exactly the set of common events of P and Q. Recursion is defined through process referencing. Its semantics is defined as Tarski's weakest fixpoint as in CSP [16].

Sensor nodes may communicate with its local continuous environment via sensing or actuating. A sensor node can sense data from its ever changing environment so as to detect certain phenomenon or to be aware of its context. The process $s \odot x \rightarrow P$ reads the value x from a sensing channel s from the environment. If the external environment is specified as a continuous function, the object 'monitors' the value of the continuous function through s. Sensor nodes with the same sensing device may receive the same datum from the external environment. There may be multiple sensing channels on one sensor node. Different sensing channels may be dedicated to different phenomenons. The data sensed by a sensor node may come from the external environment or another sensor node in the system. Sensor nodes influence its environment through actuating. Process $s \otimes v \rightarrow P$ actuates value v continuously to the environment. A sensing channel and an actuating channel match if they share the channel name.

Example 1. A light sensor of a camera detects the light condition and outputs the light level continuously. The camera reads the light level regularly so as to update the screen[1].

LightSensor

$$\mathit{Main} = \mathit{Idle}(2);\ \mathit{daylight} \odot x \rightarrow \mathit{illumination} \otimes x \rightarrow \mathit{Main}$$

where $\mathit{daylight}$ is a sensing channel and x is the sensed value. The process Main identifies the behaviors of the object after initialization (as in TCOZ [23]). After sensing the day light level from the environment, its value is actuated on channel $\mathit{illumination}$. The process repeats after 2 time units.

Display

$$\begin{aligned}
&\mathit{value} = 0\\
&\mathit{Main} = \mathit{illumination} \odot x \rightarrow\\
&\qquad\quad (\mathit{Main} \lhd \mathit{value} = x \rhd \mathit{refresh} \rightarrow \mathit{value} := x \rightarrow \mathit{Main})
\end{aligned}$$

where value is a data variable recording the current light level. Its initial value is 0. $\mathit{refresh}$ is the event of refreshing the screen. The above defines a local display device.

[1] We use a Z-schema like syntax to group components of a sensor node. It by no means implies that we adapt the Z syntax and semantics.

It reads the value of $illumination$ through sensing. If the value is different from the previous one, the screen is refreshed to show the new value, else the screen is not refreshed. Thus, the illumination displayed is in real-time. Initially, 0 is displayed. Channel $daylight$ connects the system with the external environment as there is no matching sensing channel named $daylight$. □

As shown above, the specification of one sensor node contains a set of mapping from data/process variables to their value. The initial values identify the initial state. The process $Main$ identifies the behavior of the node. Sensor nodes which form a network must communicate with each other, e.g., typically through radio transmission. There are unique characteristics about inter-sensor messaging. First of all, messaging between different nodes is almost always asynchronous because processing time per bit communicated is plentiful in sensor networks, i.e., CPUs are fast and bandwidths are low. Depending on the communication media and the transfer rate, it may take considerably long time. Secondly, there may not be a global identity for each sensor and thus end-to-end communication is unlikely in sensor networks. Thus, broadcasting is favored in sensor networks instead of messaging through channels shared by a pre-determined set of processes (as in CSP [16]).

$c!X \rightarrow P$ – output
$c?X \rightarrow P$ – input

where c is a channel. The message content X, which is called a $gradient$, could be an abstract event or a data message or even a process itself. We assume that the output is broadcasted so that all sensors within the range may receive it. Yet only those intended ones may process it. A receiving sensor node may only listen on channels of its own interest. A $gradient$ is always treated as a single parameter, and its syntax and how it is evaluated should be defined by a designer of the system and be incorporated into the operational semantics.

Example 2. Suppose the sensors have been deployed around a volcano (e.g., they are thrown from an aircraft) so as to monitor the volcano activity. The sensors report through radio whenever the sensed temperature is above certain threshold. Due to the rising of temperature, we now want those sensors which sensed high temperature to report more frequently. Thus, a station is set up near the volcano, which is modeled as follows,

$Station$
$Main = c!set(5, 20) \rightarrow Main$

where c is a channel, set is a message tag and $(5, 20)$ is a compound message in a pre-agreed format. The first number identifies the intended receivers and the second is used to update the frequency. A channel can be implemented on a separate port or radio frequency or as simple as a flag bit in the message package. The station repeatedly broadcasts the message. The sensors are designed as follows,

$Node$

$$\begin{array}{l}
delay = 5 \\
data = 0 \\
Routine = Idle(delay);\ temperature \odot x \rightarrow data := x \rightarrow Routine \\
Update = c?set(threshold, x) \rightarrow (delay := x \rightarrow Update \\
\qquad\qquad \lhd\ data > threshold \rhd Update) \\
Main = Routine \parallel Update
\end{array}$$

where $delay$ is a local variable used to control sensing frequency and $data$ is used to store the most recent sensed data. The $Main$ process is composed two sub-processes running in parallel. Process $Routine$ executes a routine task, i.e., after idling for $delay$ time units, collects temperature data from the environment and then store the sensed data into variable $data$. Process $Update$ awaits for messages from the station. Once a message arrives on channel c, if the newly sensed data is above the $threshold$, the $newdelay$ is adapted. Otherwise, process $Update$ is repeated.

The above design relies on a pre-defined message format, which is acceptable since sensor networks may be application specific. Alternatively, instead of updating one parameter of a process in a pre-determined way, a flexible sensor may be designed to be reconfigurable so that its behavior can be changed dynamically to cope with different tasks. The following shows such an approach based on the notion of higher-order processes, i.e., a message can be a process itself.

$ReconStation$

$$Sender = c!set(110, Idle(20)) \rightarrow Sender$$

The $sender$ sends a piece of program, i.e., $Idle(20)$, to directly change the behaviors of the sensors. A re-configurable sensor is designed as follows,

$ReconNode$

$$\begin{array}{l}
data = 0 \\
Recon = Idle(5) \\
Routine = Recon;\ temperature \odot x \rightarrow data := x \rightarrow Routine \\
Update = c?set(threshold, X) \rightarrow \\
\qquad\qquad (Recon := X \rightarrow Update \lhd data > threshold \rhd Update) \\
Main = Routine \parallel Update
\end{array}$$

where $Recon$ is defined as a process variable. $Routine$ is to execute $Recon$ first followed by sensing. Notice that whenever a process reference (e.g., $Recon$) is invoked, it is replaced by its current value. Once a new piece of program is received, the value of $Recon$ is updated by an assignment. □

There are two levels of concurrency in sensor networks. Firstly, local processes of a sensor node or computation devices connected by wire where communication delay is ignored can execute concurrently with possible barrier synchronization. Secondly, different sensor nodes run independently with each other and communicate through message passing. The parallel composition of local processes located at the same sensor node is denoted as $P \mid[\, X \,]\mid Q$ where common events of Σ_P and Σ_Q are synchronized. All sensor nodes are implicitly executing in parallel and the communication between different sensor nodes is through asynchronous message passing[2].

Example 3. Because inter-sensor messaging may take considerably long time, it is often desirable to have separate processes for local computation and interfacing so that local computation carries on without being delayed by message sending or receiving. The following shows such a design,

$SampleNode$

$$\begin{array}{l}
average = 0 \\
n = 0 \\
Accumulate = data \odot x \rightarrow average := (average \times n + x)/(n+1) \rightarrow \\
\qquad\qquad n := n + 1 \rightarrow Idle(2);\ Accumulate \\
Interface = sink!average \rightarrow Interface \\
Main = Interface \parallel Accumulate
\end{array}$$

where n is a counter. Process *Interface* handles communication (with a data sink). It repeatedly reads the *average* and then sends it out. Process *Accumulate* collects data from the environment. It gets the value of *data* through sensing. It then computes the new average. After idling for 2 time units, process *Accumulate* is invoked again. □

Complex process constructs can be composed from the simple ones. For instance, a process that times out after some time units is written as $P \rhd_d Q$. The process Q takes control from the process P if P has not made a move after d time units elapsed, i.e., $P \rhd_d Q$ is equivalent to $P \,\Box\, (Idle(d);\ Q)$. A process that is interrupted after executing exactly d time units is written as $P \ \triangledown_d\ Q$. The process Q takes control from P after d time units, i.e., $P \ \triangledown_d\ Q$ is equivalent to $P \ \triangledown\ (Idle(d);\ Q)$. $[b] \bullet P$ is a guarded process which behaves as P when b is evaluated to true, i.e., $[b] \bullet P \mathrel{\widehat{=}} P \lhd b \rhd Stop$. Since they are considered as syntax sugars, we skip the rest for brevity.

3 Operational Semantics

In this section, the operational semantics of ASP is explained. The configuration of a node is composed of two components, i.e., the current process expression and the binding of data/process variables to their current values. Given a set of sensor nodes, the local state of each sensor node constitutes the global configuration.

Let B be a binding, which maps data variables to a data values or process variables to process expressions. Let P be a process. A configuration of a node is a pair of the form (P, B). The global state of a network of n nodes is $\{(P_1, B_1), (P2, B_2), \cdots, (P_n, B_n)\}$, i.e., the configuration of all sensor nodes. For simplicity, we write $(P_1, B_1) \mid\mid\mid (P_2, B_2)$ to denote that (P_1, B_1) and (P_2, B_2) are part of

[2] Nearby sensors may interact through sensing/actuating.

the global state[3], i.e., there are a node with configuration (P_1, B_1) and a node with configuration (P_2, B_2) running in parallel. In the previous examples, the initialization consists of a set of equations of the form $name = initial\_value$. The initial state of a node is $(Main, BInit)$ where $Main$ is the main process and $BInit$ is the binding of all variables with their initial values. The following action prefixes are defined as abbreviations.

$$\begin{aligned}
&\alpha ::= s \odot v \mid s \otimes v \\
&\beta ::= c?v \mid c!v \\
&\lambda ::= \; e \mid \tau \mid \checkmark \\
&a ::= \alpha \mid \lambda
\end{aligned}$$

where τ is the event of idling and $\checkmark$ is the event of termination. The operational semantics is presented using a set of transition rules associated with the language constructs. A transition is of the form $(P, B) \overset{a}{\hookrightarrow} (P', B')$, which denotes that a process P with a binding B performs an action a and evolves to a new process P' and a new binding B'. A new binding of a variable x to v in B is denoted by $B \oplus \{x \mapsto v\}$, i.e., the old value of x is replaced by v.

Inspired by the operational semantics defined in [26] for Timed CSP, we extend the rules with a component B to cope with our setting, as presented in Appendix A. We remark that $\Box$, $|[\, X \,]|$ and $|||$ are symmetric. Because of the shared variables, rules in [26] are extended to capture semantics of operators which deal with variables. Figure 2 presents those transitions rules. Rule $Assign$ states that an assignment replaces the value of variable x with $eval(B, exp)$ which is the value of exp evaluated against binding B. We remark that x may be a process variable and exp may be a process expression. Rule $Con1$ and $Con2$ capture the semantics of conditional branching. If the the condition b is true, written as $B \models b$, the system proceeds as P. Otherwise, it behaves as Q. Timing information is important in modeling (and verifying) sensor networks. We adapt a simple explicit-time approach, i.e., a variable $time$ is used to represent the current time [18]. Rule $Idle1$ and $Idle2$ captures how process $Idle(d)$ behaves. Notice that the behaviors of the clock are modeled implicitly as a process which updates the variable $time$. Rule $ProcDef$ deals with process referencing. The idea is to load the definition of P dynamically from B.

In order to capture the semantics of sensing/actuating, we introduce discard actions. Discard actions are defined as α :. The discard action $(P, B) \overset{\alpha:}{\hookrightarrow} (P, B)$ means that P discards the action α. Figure 3 shows rules associated with sensing and actuating. These are based on CBS (Calculus of Broadcasting Systems) in [25] except rule $DiscardSensing$. Rule $DiscardSensing$ means that any node with the same sensor name may not receive the data from the environment or some corresponding actuator. This rule can mimic the locality of sensors. We regard the set of rules which includes *Sensing*, *DiscardSensing*, *SAParallel* and other *discard* rules (which only take the discard sensing action) the sensor process calculus, which captures the behaviors of sensors receiving data only from the environment. This set of rules is semantically weak but models how sensors behave in general.

[3] The operator $|||$ denotes interleaving in CSP. Here it implies there is no barrier synchronization among different sensor nodes.

$$\frac{eval(B, exp) = v}{(x := exp \to P, B) \overset{e}{\hookrightarrow} (P, B \oplus \{x \mapsto v\})} \quad [\ Assign\ ]$$

$$\frac{B \models b \qquad (P, B) \overset{\lambda}{\hookrightarrow} (P', B')}{(P \lhd b \rhd Q, B) \overset{\lambda}{\hookrightarrow} (P', B')} [\ Con1\ ] \quad \frac{B \not\models b \qquad (Q, B) \overset{\lambda}{\hookrightarrow} (Q', B')}{(P \lhd b \rhd Q, B) \overset{\lambda}{\hookrightarrow} (Q', B')} [\ Con2\ ]$$

$$\frac{\{time \mapsto n\} \subseteq B \qquad d > m}{(Idle(d), B) \overset{\tau}{\hookrightarrow} (Idle(d - m), B \oplus \{time \mapsto n + m\})} \quad [\ Idle1\ ]$$

$$\frac{\{time \mapsto n\} \subseteq B}{(Idle(d), B) \overset{\checkmark}{\hookrightarrow} (Stop, B \oplus \{time \mapsto n + d\})} \quad [\ Idle2\ ]$$

$$\frac{\{P \mapsto Q\} \subseteq B \qquad (Q, B) \overset{\lambda}{\hookrightarrow} (Q', B')}{(P, B) \overset{\lambda}{\hookrightarrow} (Q', B')} \quad [\ ProcInst\ ]$$

$$\frac{P = Q \qquad (Q, B) \overset{\lambda}{\hookrightarrow} (Q', B')}{(P, B) \overset{\lambda}{\hookrightarrow} (Q', B')} \quad [\ ProcDef\ ]$$

Fig. 2. Basic Operational Rules

Similarly, we define the rules for inter-sensor communication. All the rules for inter-sensor messaging, presented in Appendix B, are based on CBS but we need to take care of *gradient*. The syntax of gradients and how they will be evaluated should be pre-defined by a specifier of a sensor network system.

4 Case Studies

In this section, we demonstrate how ASP can be applied to model real-world sensor network applications concisely. Next, we discuss sensor network verification based on the ASP models. We show that using existing model checkers, bugs which are not previously known may be detected. Yet existing verification tools and techniques may not be sufficient in general.

4.1 The Trickle Algorithm

Communication in sensor networks may be extremely costly (in terms of time and battery). For some applications, sending a data of tens of kilobytes can have the same cost as days of operation. Thus, code propagation by flooding is undesirable. The Trickle algorithm [21] is a self-regulating algorithm for code (or large datum) propagation and

$$\frac{}{(s \odot x \rightarrow P, B) \overset{s\odot v}{\hookrightarrow} (P, B \oplus \{x \mapsto v\})}\ [\ \mathit{Sensing}\ ]$$

$$\frac{}{(s \otimes v \rightarrow P, B) \overset{s\bigotimes v}{\hookrightarrow} (P, B)}\ [\ \mathit{Actuating}\ ]$$

$$\frac{}{(s{\otimes}v \rightarrow P, B) \overset{s\bigotimes v:}{\hookrightarrow}}\ [\ \mathit{DiscardActuating}\ ]$$

$$\frac{}{(s \odot x \rightarrow P, B) \overset{s\bigodot v:}{\hookrightarrow}}\ [\ \mathit{DiscardSensing}\ ]$$

$$\frac{(P, B_1) \overset{s\odot v}{\hookrightarrow} (P', B_1') \quad (Q, B_2) \overset{s\bigotimes v}{\hookrightarrow} (Q', B_2)}{(P, B_1) \mid\mid\mid (Q, B_2) \overset{s\bigotimes v}{\hookrightarrow} (P', B_1') \mid\mid\mid (Q', B_2)}\ [\ \mathit{SACom}(1)\ ]$$

$$\frac{(P, B_1) \overset{s\bigotimes v}{\hookrightarrow} (P', B_1) \quad (Q, B_2) \overset{s\odot v}{\hookrightarrow} (Q', B_2')}{(P, B_1) \mid\mid\mid (Q, B_2) \overset{s\bigotimes v}{\hookrightarrow} (P', B_1) \mid\mid\mid (Q', B_2')}\ [\ \mathit{SACom}(2)\ ]$$

$$\frac{(P, B_1) \overset{s\odot v}{\hookrightarrow} (P', B_1') \quad (Q, B_2) \overset{s\odot v}{\hookrightarrow} (Q', B_2')}{(P, B_1) \mid\mid\mid (Q, B_2) \overset{s\odot v}{\hookrightarrow} (P', B_1') \mid\mid\mid (Q', B_2')}\ [\ \mathit{SAParallel}\ ]$$

$$\frac{(P, B_1) \overset{\alpha:}{\hookrightarrow} \quad (Q, B_2) \overset{\alpha:}{\hookrightarrow}}{(P, B_1) \mid\mid\mid (Q, B_2) \overset{\alpha:}{\hookrightarrow}}\ [\ \mathit{SAJoinDiscard}\ ]$$

$$\frac{(P, B_1) \overset{\alpha:}{\hookrightarrow} \quad (Q, B_2) \overset{\alpha}{\hookrightarrow} (Q', B_2')}{(P, B_1) \mid\mid\mid (Q, B_2) \overset{\alpha}{\hookrightarrow} (P, B_1) \mid\mid\mid (Q', B_2')}\ [\ \mathit{SADiscard}(1)\ ]$$

$$\frac{(P, B_1) \overset{\alpha}{\hookrightarrow} (P', B_1') \quad (Q, B_2) \overset{\alpha:}{\hookrightarrow}}{(P, B_1) \mid\mid\mid (Q, B_2) \overset{\alpha}{\hookrightarrow} (P', B_1') \mid\mid\mid (Q, B_2)}\ [\ \mathit{SADiscard}(2)\ ]$$

Fig. 3. Sensor-Actuator rules

maintenance in wireless sensor networks. It combines typical sensor network behaviors like broadcasting, higher-order processes, etc. The sensors are designed to be reconfigurable, i.e., the received code will be installed and executed. It uses a "polite gossip" policy. Each node announces its metadata (e.g., a version number) every few

$$\begin{array}{l}
\underline{\;Mote\;} \\
data = 0 \\
version = 0 \\
counter = 0 \\
threshold = 2 \\
timer = 2 \\
tau = 5 \\
Routine = sense \odot x \rightarrow data := x;\ Idle(5);\ Routine \\
Update = c?pro(v, R) \rightarrow ((version := v;\ Routine := R;\ Update) \\
\qquad\qquad \lhd\ version < v \rhd\ Update) \\
Gossip = (Talk \parallel Reply)\ \nabla_{tau}\ counter := 0;\ Gossip \\
Talk = Idle(timer);\ (meta!version \rightarrow Talk \lhd counter < threshold \rhd Talk) \\
Reply = meta?x \rightarrow ([x = version] \bullet counter := counter + 1;\ Reply\ \Box \\
\qquad\qquad [x > version] \bullet meta!version \rightarrow Reply\ \Box \\
\qquad\qquad [x < version] \bullet c!pro(version, Routine) \rightarrow Reply) \\
Main = Routine \parallel Update \parallel Gossip
\end{array}$$

Fig. 4. The Trickle Algorithm

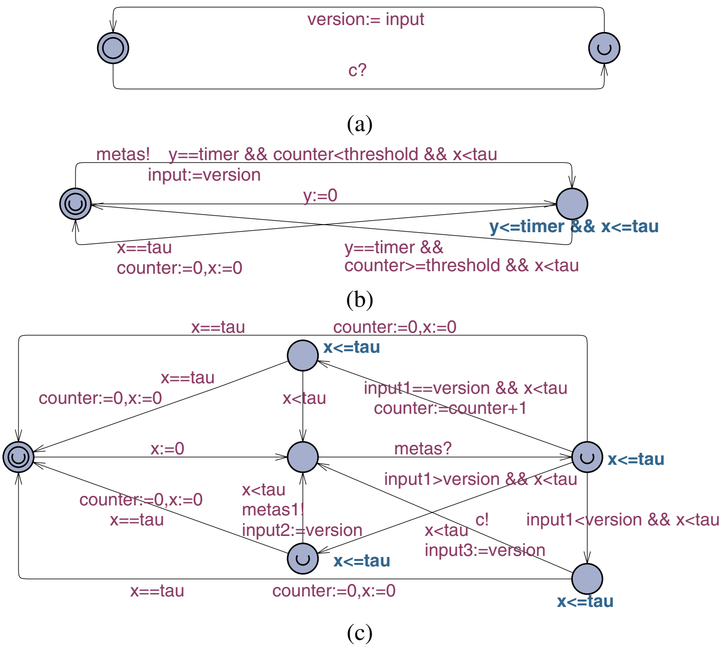
time units. If a node hears an old metadata, it broadcasts the code necessary to update the node sending the old metadata. If it hears a new metadata (e.g., a larger version number), it broadcasts its own metadata, which triggers the receiver to send the updated program. To reduce the number of communication, each node announces only a limited number of times during each gossip. Through time, a network of thousands of sensor nodes shall be updated with the new code. The algorithm has been evaluated with extensive simulation in [21].

The modeling is presented in Figure 4. The main process of each sensor node (called *mote* in [20]) consists of three components running in parallel. Process $Routine$, which is reconfigurable, performs the routine task. Process $Update$ updates the node if an updated version of $Routine$ has been received. Process $Gossip$ models the "polite gossip" policy which is used to discover whether the node is outdated or not. In order to keep the mote up-to-date, a mote records a number of parameters, i.e., $version$ records the version of $Routine$, a counter, a threshold, a timer and a constant tau. Process $Routine$ periodically senses data from the external environment. Process $Update$ keeps waiting for an input on channel c (with message tag pro). Once such a gradient is received, both $version$ and $Routine$ are updated if the received code is newer. Process $Gossip$ captures the essence of the algorithm. The mote announces its current version on channel $meta$ once a while (specified by $Talk$) or it listens to other motes and reacts (specified $Reply$). In the process $Talk$, after t time units, the mote checks if it has talked too much ($counter \geq threshold$). If it has, the mote will keep silent until the next gossip. Otherwise, it will broadcast its version on channel $meta$ and wait for the next turn to talk. Every tau time units, the process restarts (and resets the counter). In the process $Reply$, when a mote hears a $version$ identical to its own ($x = version$), it increments $counter$. If the mote hears a version greater than its own ($x > version$), it broadcasts its own version, which will trigger a receiver to send the updated program. If the mote hears an old version, it broadcasts the code necessary to update the sending mote.

We believe that formal specification is a starting point for a number of formal analysis tasks. It is natural to ask whether this algorithm satisfies important safety and liveness properties. In this experiment, we reuse existing state-of-the-art verification support for real-time systems (e.g., UPPAAL [19]) to reason about the algorithm. In order to model check the algorithm, we must first close the system by explicitly specifying the environment. The following components are part of the environment, i.e., the network topology, the different versions of $Routine$ and the external data resource from which the sensor nodes collect data. As discussed above, the network topology may be specified using pre-defined variables (e.g., $location$ and $range$). There is a link between two nodes if and only if the nodes are within each other's range. In UPPAAL, however, processes communicate only through pre-defined channels. Thus, we need to pre-process the network topology and define channels for each link between the nodes, which is statically done given the values of the variables are not changing. One feature which is missing from timed automata is higher order processes. To the best of our knowledge, there are few tools which support verification of higher-order processes. Thus, higher-order processes are reduced to ordinary processes whenever the system is closed. This may not be always desirable or possible. For this experiment, because the details of $Routine$ are irrelevant, we simply abstract it away. In UPPAAL, timed automata are extended with broadcasting channels and committed states, which can be used to mimic broadcasting in ASP. Because the sensed data is irrelevant, it is simply ignored.

In our previous work, we have developed a systematic translation from Timed CSP to timed automata [10]. We defined a rich set of compositional operators for timed automata, which corresponds to the compositional operators in Timed CSP. By following the same approach, we develop a systematic translation from ASP to timed automata. Figure 5 presents the generated automata. Automaton (a) corresponds to the process $Update$. Because passing data through channels is not allowed in UPPAAL, we use shared variables to communicate instead, i.e., message sending writes a shared variable whereas message receiving reads the value. The state after the synchronization is marked as urgent so as to read the input value immediately. The parallel composition of automaton (b) and (c) corresponds to process $Gossip$. Adapting the timed patterns defined in [10], the timed interrupt $\triangledown_{tau}$ has been resolved using extra state invariants and transition guards. For instance, automaton (b) which loosely corresponds to process $Talk$ is modified to guarantee that whenever $x == tau$, the system is restored to the initial state. Automaton (c) loosely corresponds to process $Reply$.

Table 6 shows the experiment results using UPPAAL to verify instances of the algorithm. The network topologies are randomly generated with one constraint, i.e., all nodes are reachable. The results are obtains by executing UPPAAL 4.0.6 on Windows XP platform with Intel Core Duo 2.33GHz CPU and 3.25 GB memory. All models are deadlock-free as expected. A desired property of code propagation algorithms is that if a node is reachable, it will always eventually be updated. A counterexample is produced unexpectedly. Figure 7 elaborates the counterexample with a network containing 3 motes connected circularly. The link between nodes are directed as it is possible that node B hears A but node A cannot hear from B, e.g., A has a longer radio range. Initially, node A's version is 1, meaning that it is updated already. Once node A hears a meta-data from node C, it broadcasts its updated program. Only node B hears from



(a)

(b)

(c)

Fig. 5. UPPAAL Model of the Trickle Algorithm

Model	Property	time taken	Result
2 Motes	deadlock-free	< 1	true
3 Motes	same above	1	true
4 Motes	same above	12	true
5 Motes	same above	1080	true
6 Motes	same above	–	true
2 Motes	always-eventually all motes are updated	< 1	true
3 Motes	same above	1	false
4 Motes	same above	3	false
5 Motes	same above	25	false
6 Motes	same above	–	–

Fig. 6. Verifying Randomized Sensor Networks using UPPAAL

node A and thus node B is updated. Consequently, node C broadcasts its meta-data every time after receiving meta-data from node B and then node A broadcasts the updated program every time after receiving the meta-data from node C. However, because only node B receives from node A and it ignores the message since it has already been updated. As a result, node C is never updated. Notice that the property is true if all links are bi-directional or each node has fixed location and radio range. In order to prove the latter, we can show that if B hears A and A cannot hear from B, then the range of A

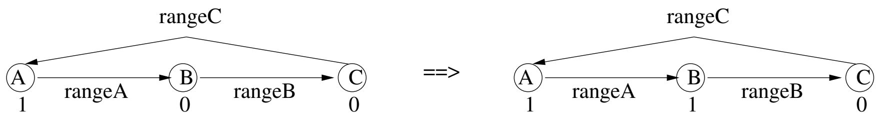


Fig. 7. A Counterexample

must be larger than that of B. Thus, a circular network like the one in Figure 7 is not possible. Nevertheless, such network topology is possible in practice.

For a network with 6 motes, UPPAAL needs significant amount time to verify the property. A typical sensor network application, however, may contain hundreds or thousands of sensor nodes. Observing that some distinguishable features are missing from UPPAAL (e.g., higher-order processes), this suggests that specialized verification mechanism and state space reduction techniques must be developed.

4.2 Face Routing Protocols

In this experiment, we review another example of sensor network application. We argue that current formal verification tools may not be sufficient to provide answers to natural questions on those systems.

Geographic routing algorithms are an important family of routing protocols of wireless ad hoc sensor networks. They have been shown to scale better than other alternatives, i.e., they require per node state that depends only on network density and not on network size. Established proposals include GFG [2], GPSR [17], etc. However, due to different forwarding mechanism, reachability between sensor nodes is not always guaranteed. In many of these algorithms (e.g., [17]), nodes forward packets to the neighbor closest to the destination whenever possible. The following process models the relevant behaviors of a node in the simplest greedy routing protocol.

$$\begin{aligned} Main = {} & c?new(src, des, msg) \rightarrow \\ & \quad ([loc = des] \bullet message := msg;\ Skip \ \Box \\ & \quad [src - des > loc - des] \bullet c!new(loc, des, msg) \rightarrow Main \ \Box \\ & \quad [loc - des \geqslant src - des] \bullet Main) \end{aligned}$$

where loc is the location of the node and $message$ is a local data store recording the message received. In what follows, we assume that each node acquires its own position using GPS devices (as assumed in [17]). Without loss of generality, location is abstracted as a natural number. The message is composed of three parts, i.e., src is the location of the sender, des is the location of the intended receiver and msg is the message content itself. Each node tries to forward the message to its neighbors who are geographically closer to the destination than the node itself (i.e., $src - des > loc - des$). If the location of the sensor is the same as the destination, it means that message has reached the destination. If the sensor is not the destination and it is closer to the destination than the the sender, it would go on broadcasting the message with the sender location replaced by its own location. Otherwise it will discard this message. In [17], a simple beaconing algorithm provides all nodes with their neighbors' positions: periodically, each node broadcast a beacon which its own identifier and position. For simplicity, we skip the modeling of the beaconing algorithm and assume that the messages are always broadcasted in the above modeling. Nonetheless, this modeling shares the same pitfall with the one in [17].

A critical requirement for any routing protocol is that, given a static network topology and reliable link layer support, the protocol must guarantee message delivery to a reachable node. It has been shown that message delivery is not always guaranteed for face routing algorithms [13]. For instance, there are topologies in which the only route to a destination requires a packet move temporally further in geometric distance from the destination, due to the presence of routing holes. Refer to [13] for more complicated protocols as well as pitfalls. Given a newly designed protocol, it is thus desirable to answer the question whether there exists a network topology such that the protocol does not function as expected. Different from traditional model checking, a model satisfying different constraints including temporal logic properties (e.g., always eventually the message is delivered) must be constructed and presented as a counterexample. This problem may be categorized as a model satisfiability problem. Because it concerns temporal logic constraints, bounded model checking techniques must be applied.

One solution is to formula the question as a Boolean formula and then apply state-of-the-art SAT-solvers to generate solutions automatically. In our previous work [29], we have developed ways of encoding compositional processes as SAT problems for bounded model checking. By applying a similar approach, our primary experiments show that for face routing protocols, not only we can prove/disprove desirable properties given a network topology but also generate one particular network topology which makes the given algorithm faulty. Given a bound on the number of nodes, and the data range of the pre-defined variable $location$ and $range$ (which identifies the network topology), a fixed number of Boolean variables are used to represent status of each node. The behaviors of each node can be translated to a labeled transition system (by applying the operational semantic) and then encoded as a Boolean formula in the standard way [9]. The encoded sensor nodes are composed using a similar approach proposed in [29]. After composing with the Boolean formula which represents the property (e.g., if there is a path from the source to the destination, then the message must always eventually reach the destination), an SAT-solver is used to assign *true/false* value to all Boolean variables. An assignment to the variables representing $location$ and $range$ identifies a network topology in which the protocol can not guarantee message delivery. In the above example, we have successfully generated connectivity graphes which contains routing holes. We are currently extending our tool [29] to fully automate the process. We have implemented and experimented a number of face routing protocols (like the above one and GFG). However, because of the size of sensor network applications, our prototype implementation must be extended with partial order reduction as well as symbolic techniques before practical usage.

5 Conclusion

In this paper, we proposed a high-level formal specification language specifically for wireless sensor networks. Unique language constructs have been defined to cope with the unique characteristics of such systems, e.g., sensing and actuating, inter-sensor messaging, etc. Next, we developed a formal semantics for Asp. Lastly, we demonstrated how to use Asp to model sensor network applications as well as how to verify those models. From the examples, Asp showed its high expressiveness and conciseness in formally specifying the communications and behaviors of sensor network systems. In summary, Asp not only offers a way of modelling/specifying sensor networks, with the precise semantics defined, but also gives us a starting point for formal sensor network simulation, verification and synthesis.

We are currently developing a series of tools based on Asp. e.g., a high-level simulator, a verifier and a synthesizer (e.g., [28,7,11]). Existing simulators for sensor networks like TOSSIM and ns-2 mimic physical environment rather closely and thus provide rich simulation results. Nevertheless, they may not be as abstract as expected by system designers who are only interested in the high-level functionality of the system. For instance, a protocol designer would be firstly interested in if "in theory" the newly designed protocol is free of deadlocks and livelocks and then analysis the performance of the protocol with realistic settings. We are developing a simulator based on Asp for high-level simulation, based on the operational semantics presented in Section 3. A promising technique to handle large number of sensor nodes is the symbolic simulation proposed in [15]. One in-expensive approach to connect Asp to the current practise of sensor networks programming is by providing a transformation (which has been planned) from Asp to languages like *nesC* which is designed to embody the execution model of TinyOS [14].

Acknowledgement

This work is partially supported by the research project "Sensor Networks Specification and Validation" (R-252-000-320-112) funded by Ministry of Education, Singapore.

References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless Sensor Networks: a Survey. Computer Networks 38(4), 393–422 (2002)
2. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with Guaranteed Delivery in Ad Hoc Wireless Networks. Wireless Networks 7(6), 609–616 (2001)
3. Botts, M.: Sensor Model Language (SensorML) (2006), `http://vast.nsstc.uah.edu/SensorML/`
4. Boulis, A., Han, C.C., Srivastava, M.B.: Design and Implementation of a Framework for Efficient and Programmable Sensor Networks. In: Proceedings of International Conference on Mobile Systems, Applictions, and Services (MobiSys 2003) (2003)
5. Cardelli, L., Gordon, A.D.: Mobile Ambients. Theoretical Computer Science 240(1), 177–213 (2000)
6. Cardelli, L., Gordon, A.D.: Mobile ambients. In: Nivat, M. (ed.) FOSSACS 1998. LNCS, vol. 1378. Springer, Heidelberg (1998)
7. Chen, C., Dong, J.S., Sun, J.: A verification system for timed interval calculus. In: Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), pp. 271–280. ACM Press, New York (2008)

8. Ciancio, A.G., Pattem, S., Ortega, A., Krishnamachari, B.: Energy-efficient Data Representation and Routing for Wireless Sensor Networks based on a Distributed Wavelet Compression Algorithm. In: Proceedings of Information Processing in Sensor Networks (IPSN 2006), pp. 309–316 (2006)
9. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
10. Dong, J.S., Hao, P., Qin, S.C., Sun, J., Wang, Y.: Timed Patterns: TCOZ to Timed Automata. In: Davies, J., Schulte, W., Barnett, M. (eds.) ICFEM 2004. LNCS, vol. 3308, pp. 483–498. Springer, Heidelberg (2004)
11. Dong, J.S., Hao, P., Sun, J., Zhang, X.: A Reasoning Method for Timed CSP Based on Constraint Solving. In: Liu, Z., He, J. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 342–359. Springer, Heidelberg (2006)
12. Estrin, D., Govindan, R., Heidemann, J., Kumar, S.: Next Century Challenges: Scalable Coordination in Sensor Networks. In: Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 1999), pp. 263–270 (1999)
13. Frey, H., Stojmenovic, I.: On Delivery Guarantees of Face and Combined Greedy-face Routing in Ad Hoc and Sensor Networks. In: Proceedings of the 12th Annual International Conference on Mobile Computing and Networking (MOBICOM 2006), pp. 390–401 (2006)
14. Gay, D., Levis, P., von Behren, J.R., Welsh, M., Brewer, E.A., Culler, D.E.: The nesC Language: A Holistic Approach to Networked Embedded Systems. In: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation 2003 (PLDI 2003), pp. 1–11. ACM Press, New York (2003)
15. Goel, A., Meng, S., Roychoudhury, A., Thiagarajan, P.S.: Interacting process classes. In: Proceedings of the 28th International Conference on Software Engineering (ICSE 2006), pp. 302–311. ACM Press, New York (2006)
16. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs (1985)
17. Intanagonwiwat, C., Govindan, R., Estrin, D.: Directed Diffusion: a Scalable and Robust Communication Paradigm for Sensor Networks. In: Proceedings of International Conference on Mobile Computing and Networking (MOBICOM 2000), pp. 56–67 (2000)
18. Lamport, L.: Real-Time Model Checking Is Really Simple. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 162–175. Springer, Heidelberg (2005)
19. Larsen, K.G., Pettersson, P., Wang, Y.: Uppaal in a Nutshell. International Journal on Software Tools for Technology Transfer 1(1-2), 134–152 (1997)
20. Levis, P., Culler, D.E.: Maté: a Tiny Virtual Machine for Sensor Networks. In: Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2002), pp. 85–95 (2002)
21. Levis, P., Patel, N., Culler, D.E., Shenker, S.: Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In: Proceedings of Networked Systems Design and Implementation (NSDI 2004), pp. 15–28 (2004)
22. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: an Acquisitional Query Processing System for Sensor Networks. ACM Transactions on Database Systems 30(1), 122–173 (2005)
23. Mahony, B., Dong, J.S.: Timed Communicating Object Z. IEEE Transactions on Software Engineering 26(2) (February 2000)
24. Mezzetti, N., Sangiorgi, D.: Towards a Calculus For Wireless Systems. Electronic Notes in Theoretical Computer Science 158, 331–353 (2006)
25. Prasad, K.V.S.: A Calculus of Broadcasting Systems. In: Abramsky, S. (ed.) TAPSOFT 1991. LNCS, vol. 493, pp. 338–358. Springer, Heidelberg (1991)

26. Schneider, S.: An Operational Semantics for Timed CSP. Information and Computation 116(2), 193–213 (1995)
27. Schneider, S., Davies, J., Jackson, D.M., Reed, G.M., Reed, J.N., Roscoe, A.W.: Timed CSP: Theory and practice. Real-Time: Theory in Practice 600, 640–675 (1992)
28. Sun, J., Dong, J.S.: Design Synthesis from Interaction and State-Based Specifications. IEEE Transactions on Software Engineering 32(6) (2006)
29. Sun, J., Liu, Y., Dong, J.S., Sun, J.: Bounded Model Checking of Compositional Processes. In: Proceedings of the Second IEEE International Symposium on Theoretical Aspects of Software Engineering, pp. 23–30. IEEE Computer Society Press, Los Alamitos (2008)
30. Thomsen, B.: Calculi for Higher Order Communicating Systems. PhD thesis (1990)
31. Tschirner, S., Xuedong, L., Yi, W.: Model-Based Validation of QoS Properties of Biomedical Sensor Networks. In: Proceedings of the International Conference on Embedded Software (EMSOFT 2008) (accepted, 2008)
32. von Eicken, T., Culler, D.E., Goldstein, S.C., Schauser, K.E.: Active Messages: A Mechanism for Integrated Communication and Computation. In: Proceedings of International Symposium on Computer Architecture 1992 (ISCA 1992), pp. 256–266 (1992)

Appendix A: Basic Operational Semantics

$$\frac{}{(\mathit{Stop}, B) \overset{\tau}{\hookrightarrow} (\mathit{Stop}, B)} \qquad \frac{}{(\mathit{Skip}, B) \overset{\checkmark}{\hookrightarrow} (\mathit{Stop}, B)}$$

$$\frac{}{(e \rightarrow P, B) \overset{e}{\hookrightarrow} (P, B)} \qquad \frac{(P, B) \overset{\lambda}{\hookrightarrow} (P', B') \quad \lambda \neq \checkmark}{(P;\ Q, B) \overset{\lambda}{\hookrightarrow} (P';\ Q, B')}$$

$$\frac{(P, B) \overset{\checkmark}{\hookrightarrow} (P', B)}{(P;\ Q, B) \overset{\checkmark}{\hookrightarrow} (Q, B)} \qquad \frac{(P, B) \overset{\lambda}{\hookrightarrow} (P', B')}{(P \Box Q, B) \overset{\lambda}{\hookrightarrow} (P', B')}$$

$$\frac{(P, B) \overset{\lambda}{\hookrightarrow} (P', B')}{(P \triangledown Q, B) \overset{\lambda}{\hookrightarrow} (P' \triangledown Q, B')} \qquad \frac{(Q, B) \overset{\lambda}{\hookrightarrow} (Q', B')}{(P \triangledown Q, B) \overset{\lambda}{\hookrightarrow} (Q', B')}$$

$$\frac{(P, B) \overset{\lambda}{\hookrightarrow} (\mathit{Skip}, B)}{(P \triangledown Q, B) \overset{\lambda}{\hookrightarrow} (\mathit{Skip}, B)} \qquad \frac{\lambda \notin X \quad (P, B) \overset{\lambda}{\hookrightarrow} (P', B')}{(P \,|[\, X \,]|\, Q, B) \overset{\lambda}{\hookrightarrow} (P' \,|[\, X \,]|\, Q, B')}$$

$$\frac{e \in X \quad (P, B) \overset{e}{\hookrightarrow} (P', B) \quad (Q, B) \overset{e}{\hookrightarrow} (Q', B)}{(P \,|[\, X \,]|\, Q, B) \overset{e}{\hookrightarrow} (P' \parallel Q', B)}$$

Appendix B: Operational Rules for Inter-sensor Messaging

$$\frac{}{(c?X \to P, B) \overset{c?v}{\hookrightarrow} (P \oplus \{X \mapsto v\}, B)}\ [\,BI\,] \qquad \frac{}{(c?v \to P, B) \overset{c?v:}{\hookrightarrow}}\ [\,\mathit{DisI}\,]$$

$$\frac{}{(c!v \to P, B) \overset{c!v}{\hookrightarrow} (P, B)}\ [\,BO\,] \qquad \frac{}{(c!v \to P, B) \overset{c!v:}{\hookrightarrow}}\ [\,\mathit{DisO}\,]$$

$$\frac{(P_1, B_1) \overset{c!v}{\hookrightarrow} (P_1', B_1) \quad (P_2, B_2) \overset{c?v}{\hookrightarrow} (P_2', B_2')}{(P_1, B_1) \mathbin{|||} (P_2, B_2) \overset{c!v}{\hookrightarrow} (P_1', B_1) \mathbin{|||} (P_2', B_2')}\ [\,\mathit{Broadcast}\,]$$

$$\frac{(P_1, B_1) \overset{c?v}{\hookrightarrow} (P_1', B_1') \quad (P_2, B_2) \overset{c?v}{\hookrightarrow} (P_2', B_2')}{(P_1, B_1) \mathbin{|||} (P_2, B_2) \overset{c?v}{\hookrightarrow} (P_1', B_1') \mathbin{|||} (P_2', B_2')}\ [\,\mathit{IParallel}\,]$$

$$\frac{(P_1, B_1) \overset{\beta:}{\hookrightarrow} \quad (P_2, B_2) \overset{\beta:}{\hookrightarrow}}{(P_1, B_1) \mathbin{|||} (P_2, B_2) \overset{\beta:}{\hookrightarrow} (P_1, B_1) \mathbin{|||} (P_2, B_2)}\ [\,\mathit{BJointDiscard}\,]$$

$$\frac{(P_1, B_1) \overset{\beta:}{\hookrightarrow} \quad (P_2, B_2) \overset{\beta}{\hookrightarrow} (P_2', B_2')}{(P_1, B_1) \mathbin{|||} (P_2, B_2) \overset{\beta}{\hookrightarrow} (P_1, B_1) \mathbin{|||} (P_2', B_2')}\ [\,\mathit{BDiscard}(1)\,]$$

$$\frac{(P_1, B_1) \overset{\beta}{\hookrightarrow} (P_1', B_1') \quad (P_2, B_2) \overset{\beta:}{\hookrightarrow}}{(P_1, B_1) \mathbin{|||} (P_2, B_2) \overset{\beta}{\hookrightarrow} (P_1', B_1') \mathbin{|||} (P_2, B_2)}\ [\,\mathit{BDiscard}(2)\,]$$