A Compositional Approach for Modeling and Timing Analysis of Wireless Sensor and Actuator Networks

Marjan Sirjani
Malardalen University,
Västeras, Sweden
Reykjavik University,
Reykjavik, Iceland
marjan.sirjani@mdh.se

Ehsan Khamespanah
University of Tehran, Tehran,
Iran
Reykjavik University,
Reykjavik, Iceland
ehsan13@ru.is

Kirill Mechitov
University of Illinois at
Urbana-Champaign
Champaign, United States
mechitov@illinois.edu

Gul Agha
University of Illinois at
Urbana-Champaign
Champaign, United States
agha@illinois.edu

ABSTRACT

Wireless sensor and actuator networks (WSAN) are created through the integration of multiple nodes which acquire data and perform reaction based on them. In a general overview, sensor nodes of WSANs are responsible for data acquisition and sending them to a central node. The central node stores all the received data and performs reactions. Timing verification of WSAN applications to ensure schedulability of tasks is a challenge, and is generally performed by worst-case analysis. This process is error-prone and inherently conservative. On the other hand, using model checking for analyzing WSAN applications results in state space explosion even for middle-sized configurations. The reason is the necessity of considering the interleaving of the large number of sensors in WSANs. In this paper, we show how to build an actor-based model of WSAN applications, starting from sensor node-level and moving towards the full system, and we show how this compositional modeling improves analysability and modifiability. Realtime extension of actor model is appropriate for modeling WSAN applications where we have many concurrent and asynchronous processes, and interdependent realtime deadlines. We demonstrate the approach using a case study of a distributed realtime data acquisition system for high-frequency sensing, where Timed Rebeca is used for modeling. We use model checking to check the intra/inter-sensor node schedulability.

Keywords

Actor Model; Compositional Approach; Realtime Systems; Sensor Network; Timed Rebeca

1. INTRODUCTION

Our primary goal in this paper is to show a compositional real-time scheduling frame-work where global timing properties at the system level are established by analyzing and proving local timing properties at the component level design. This approach has been applied for the analysis of wireless sensor and actuator networks[1].

Wireless sensor and actuator networks (WSANs) provide low-cost continuous monitoring infrastructure. However, as they become more complex because of increased functionalities, it is necessary to develop techniques and methods that facilitate designing and verification of large complex WSAN applications. The complexity of WSAN applications originated from the complexity of concurrent and distributed programming, networking, real-time requirements, and power constraints. Component-based design, as a widely accepted methodology for designing large complex systems, is one of the approaches that can be used to overcome these complexities. Using component-based design, components are assembled into a system without violating the principle of compositionality such that the properties that have been established at the component level also hold at the system level. This way, a model can be incrementally extended and refined during the design process, by adding new interactions and constraints. In this paper, we propose an actor-based modeling approach [2] that allows WSAN application programmers to assess the functional behavior of their code throughout the design and implementation phases (Section 2). To this end, we combine and abstract the collective requirements of a WSAN component as a single requirement (intra-sensor node model in Section 3), and then show how these components are composed into a global system without violating the proven property (inter-sensor node model in Section 4).

For WSAN applications, as real-time systems, it is important to analyze the *timing* properties of components compositionally. Analytical approaches are based on conservative worst-case assumptions and empirical measurements, and may lead into schedules that do not utilize resources in the most efficient way. A second approach is trial and error. In [18], an empirical test-and-measure approach based on binary search is used to find configuration parameters, including worst-case task runtimes, and time-slot length of the communication protocols. Trial and error is a laborious process, which nevertheless fails to provide any safety

[1]This is an invited paper based on [14].

guarantees for the resulting configuration. A third possibility is to extend scheduling techniques that have been developed for real-time systems [19] such that they can be used in WSAN environments. Unlike a real-time operating system (RTOS), the event-driven operating systems of WSAN applications (e.g. TinyOS [11]) generally do not provide real-time scheduling guarantees, priority-based scheduling, or resource reservation functionality. Without such support, many schedulability analysis techniques cannot be effectively employed.

In this paper, in addition to representing a WSAN application as a collection of actors, we show how the modeling language Timed Rebeca [25], and its model checking tool, Afra [1, 15, 16], can be used for timing analysis of WSAN applications. Timed Rebeca is a high-level actor-based language capable of representing functionality and timing behavior at an abstract level.

We present a case study involving real-time continuous data acquisition for structural health monitoring and control (SHMC) of a civil infrastructure [18]. This system has been implemented on the Imote2 wireless sensor platform, and is used in several long-term development of several highway and railroad bridges [29]. SHMC application development has proven to be particularly challenging: it has the complexity of a large-scale distributed system with real-time requirements, while having the resource limitations of low-power embedded WSAN platforms. Ensuring safe execution requires modeling the interactions of components of each node, as well as interactions among the nodes. Moreover, the application tasks are not isolated from the other aspects of the system: they execute alongside with the tasks of other applications, middleware services, and operating system components. In our case study, all periodic tasks (sample acquisition, data processing, and radio packet transmission) are required to complete before the start of their next iteration. Our model checking results show that a guaranteed-safe application configuration can be found. Moreover, this configuration improves resource utilization compared to the previous informal schedulability analysis used in [18], supporting a higher sampling rate or a larger number of nodes without violating schedulability constraints.

2. PRELIMINARIES

2.1 WSAN Applications

A WSAN application is a distributed system with multiple sensor nodes, each comprised of the independent concurrent entities bridged together via a wireless communication device which uses a transmission control protocol. Interactions between components, both within a sensor node and across sensor nodes, are concurrent and asynchronous. Moreover, WSAN applications are sensitive to timing, with soft deadlines at each step of the process needed to ensure correct and efficient operation. Due to performance requirements, and latencies of operations on sensor nodes, sensing, data processing, and communication processes must be coordinated. In particular, once a sample is acquired its corresponding radio transmission activities must be performed. Concurrently, data processing tasks–such as compensating sensor data for the effects of temperature changes–must be executed. Moreover, the timing of radio transmissions from different nodes must be coordinated using a communication protocol.

2.2 The Actor Model of WSAN Applications

The Actor model is a well-established paradigm for modeling distributed and asynchronous component-based systems. The actor model developed as a model of concurrent computation for open distributed systems where *actors* are the concurrently executing entities [2]. One way to think of actors is as components providing services that may be requested via messages from other components. A message is buffered until the provider is ready to execute the message. As a result of processing a message, an actor may send messages to other actors, and to itself. Extensions of the actor model have been used for real-time systems, in particular: RT-synchronizer [24], real-time Creol [6], and Timed Rebeca [25]. The characteristics of real-time variants of the actor model make them useful for modeling WSAN applications: many concurrent processes and interdependent real-time deadlines.

In our compositional approach for modeling WSAN applications, we represent components of each WSAN sensor node capable of independent action as an actor. Specifically, as shown in Figure 1, a sensor node is modeled using four actors: `Sensor` (for the data acquisition) `CPU` (processor), `RCD` (a radio communication device) and `Misc` (carrying out miscellaneous tasks unrelated to sensing or communication). `Sensor` acquires data and sends it to `CPU` for further data processing. Meanwhile, `CPU` may respond to messages from `Misc` by carrying out other computations. The processed data is sent to `RCD` to forward it to a data collector node. We model the communication medium as the actor (`Ether`) and the receiver node also by the actor `RCD`. Using the actor `Ether` facilitates modularity: specifically, implementation of the Media Access Control (MAC) level details of communication protocols is localized, making it is easy to replace component sub-models for modeling different communication protocols without significantly impacting the remainder of the model. During the application design phase, different components, services, and protocols may be considered. For example, TDMA [8] as a MAC-level communication protocol may be replaced by B-MAC [23] with minimal changes.

2.3 Timed Rebeca and the Model Checking Toolset

A Timed Rebeca model consists of reactive classes and a main program which instantiates actors (also called *rebecs*). As usual, actors have an encapsulated state, a local time, and their own thread of control. Each actor contains a set of state variables, methods and a set of actors it knows. An actor may only send messages to actor that it knows. Message passing is implemented by method calls: calling a method of an actor (target) results in sending a message to the target. Each actor has a message bag in which arriving messages may be buffered; the maximum capacity of the bag is defined by the modeler.

Timing behavior in Timed Rebeca is represented using three timing primitives: `delay`, `after`, and `deadline`. A `delay` term models the passing of time for an actor. The primitives `after` and `deadline` can be used in conjunction with a message send: `after` n indicates it takes n time units for the message to be delivered to its receiver; `deadline` n indicates that if the message is not taken in n time units, it should be purged from the receiver's bag. Afra 2.0 supports

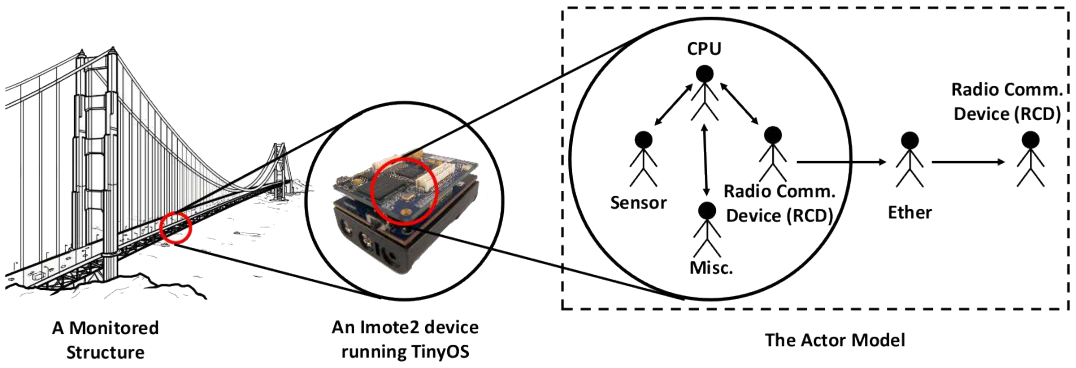

Figure 1: Modeling the behavior of a WSAN application in its real-world installation in the actor model

deadlock detection and schedulability analysis of Timed Rebeca models; we use Afra 2.0 in this work. Timed Rebeca and Afra toolset have previously been used to model and analyze realtime actor based models such as routing algorithms and scheduling policies Network on Chip designs [27, 26].

3. SCHEDULABILITY ANALYSIS OF INTRA-SENSOR NODE MODEL

We now illustrate our approach using an intra-sensor node Timed Rebeca model of a WSAN application to check for possible deadline violations. Specifically, by changing the timing parameters of our model, we find the maximum safe sampling rate in the presence of other (miscellaneous) tasks in the node. Then, we show how the specification of a intra-sensor node level model can be naturally extended to inter-sensor node specifications. Following the proposed decomposition of WSAN applications to actors in Listing 1, we prepared a Timed Rebeca model which contains four different *reactive classes* in Listing 1 through Listing 3.

As shown in Listing 1, the maximum capacity of the message bag of `Sensor` is set to 10, the only actor `Sensor` knows about is of type `CPU` (line 4), and `Sensor` does not have any state variables (line 5).

Listing 1: Reactive class of the `Sensor`
```
 1 env int samplingRate = 25; // Hz
 2
 3 reactiveclass Sensor(10) {
 4   knownrebecs { CPU cpu; }
 5   statevars { }
 6
 7   Sensor() { self.sensorFirst(); }
 8
 9   msgsrv sensorFirst() {
10     self.sensorLoop() after(?(10, 20, 30)); // ms
11   }
12   msgsrv sensorLoop() {
13     int period = 1000 / samplingRate;
14     cpu.sensorEvent() deadline(period);
15     self.sensorLoop() after(period);
16   }
17 }
```

The behavior of `Sensor` is to periodically acquire data and send it to `CPU`, which is implemented in the `sensorLoop` (lines 12-15) message server. The sent data must be serviced before the start time of the next period, specified by the value of `period` as the parameter of `deadline`. As there is a nondeterministic initial offset after which the data acquisition becomes a periodic task, we implemented `sensorFirst` to nondeterministically starts data acquisition after 10, 20, or 30 units of time (line 10). Note that in line 1, the sampling rate is given as a constant. A similar approach is used in the implementation of the `Misc` reactive class.

The behavior of `CPU` as the target of `Sensor` and `Misc` events is more complicated (Listing 2). Upon receiving a `miscEvent`, `CPU` waits for `miscTaskDelay` units of time; this represents computation cycles consumed by miscellaneous tasks. Similarly, after receiving a `sensorEvent`, `CPU` waits for `sensorTaskDelay` units of time; this represents cycles required for local data processing. Data is packed in a packet and upon the threshold (line 17), `CPU` asks `senderDevice` to send the packet which contains the collected data (line 18).

Listing 2: Reactive class of the `CPU`
```
 1 env int sensorTaskDelay = 2; // ms
 2 env int miscTaskDelay = 10; // ms
 3 env int bufferSize = 3; // samples
 4
 5 reactiveclass CPU(10) {
 6   knownrebecs { RCD senderDevice, receiverDevice; }
 7   statevars { int collectedSamplesCounter; }
 8
 9   CPU() { collectedSamplesCounter = 0; }
10
11   msgsrv miscEvent() {
12     delay(miscTaskDelay);
13   }
14   msgsrv sensorEvent() {
15     delay(sensorTaskDelay);
16     collectedSamplesCounter += 1;
17     if (collectedSamplesCounter == bufferSize) {
18       senderDevice.send(receiverDevice, 1);
19       collectedSamplesCounter = 0;
20     }
21   }
22 }
```

As this is a intra-sensor node model, communication between sensor nodes is omitted. The behavior of `RCD` is limited to waiting for some amount of time (line 6); this represents the sending time of a packet.

Note that computation times (`delay`'s) depend on the low-level aspects of the system and are application-independent; they can be measured before the application design. For schedulability analysis, we set the `deadline` for messages in a way that any scheduling violations are caught by the model checker.

Listing 3: The intra-sensor node implementation of `RCD`

```
env int OnePacketTransmissionTime = 7; // ms

reactiveclass RCD (2) {
  RCD() { }
  msgsrv send(RCD recDevice, byte numOfPackets) {
    delay(OnePacketTransmissionTime);
  }
}
```

4. SCHEDULABILITY ANALYSIS OF INTER-SENSOR NODE MODEL WITH A DISTRIBUTED COMMUNICATION PROTOCOL

Composing the models of intra-sensor nodes to have the inter-sensor node model requires that the wireless communication medium `Ether` be specified and a communication protocol is implemented in radio communication devices. Note that the process of sending a packet is controlled by a wireless network communication protocol.

The reactive class of `Ether` (Listing 4) has three message servers: these are responsible for giving the status of the medium, broadcasting data, and resetting the condition of the medium after a successful transmission. Broadcasting data takes place by sending data to a `RCD` which is addressed by the `receiverDevice` variable. So, we can easily examine the status of the `Ether` using the value of `receiverDevice` (i.e., medium is free if `receiverDevice` is not `null`, line 13). This way, after sending data, the value of `receiverDevice` and `senderDevice` must be set to `null` to show that the transmission is completed (lines 28 and 29). Data broadcasting is the main behavior of `Ether` (lines 15 to 26). Before the start of broadcasting, the `Ether` status is checked (line 16) and data-collision error is raised in case of two simultaneous broadcasts (line 24). With a successful data broadcast, `Ether` sends an acknowledgment to itself (line 19) and the sender (line 20), and informs the receiver of the number of packets sent to it (line 21). In addition to the functional requirements of `Ether`, there may be non-functional requirements. For example, the Imote2 radio offers a theoretical maximum transfer speed of 250 kbps. When considering only the useful data payload (goodput), this is reduced to about 125 kbps.

We now extend `RCD` to support communication protocols. The way that a communication protocol allows sensor nodes to access to the communication medium affects the feasibility of compositional verification. For the development of WSAN applications, developers mainly use one of TDMA or B-MAC as the communication protocol. As we will show in the following, WSAN applications can be compositionally model checked while TDMA or B-MAC is used as the communication protocol.

Using TDMA, an execution period is defined and each node in the network has one or more time slots to transmit a packet or a series of packets in an execution period. If a node has data available to transmit during its allotted time slot, it may be sent immediately. Otherwise, packet sending is delayed until its next allotted time slot.

Listing 4: Reactive class of the `Ether`

```
1  env int OnePacketTT = 7; // ms (transmission time)
2
3  reactiveclass Ether(5) {
4    statevars {
5      RCD senderDevice, receiverDevice;
6    }
7
8    Ether() {
9      senderDevice = null;
10     receiverDevice = null;
11   }
12   msgsrv getStatus() {
13     ((RCD)sender).receiveStatus(receiverDevice != null);
14   }
15   msgsrv broadcast(RCD receiver, int packetsNumber) {
16     if(senderDevice == null) {
17       int transTime = packetsNumber * OnePacketTT;
18       senderDevice = (RCD)sender;
19       receiverDevice = receiver;
20       self.broadcastingIsCompleted() after(transTime);
21       ((RCD)sender).receiveResult(true) after(transTime);
22       receiver.receiveData(receiver, packetsNumber);
23     } else {
24       ((RCD)sender).receiveResult(false);
25     }
26   }
27   msgsrv broadcastingIsCompleted() {
28     senderDevice = null;
29     receiverDevice = null;
30   }
31 }
```

This way, the packet transmission of one sensor node does not interfere with the other sensor nodes. Having more sensor nodes only results in having shorter time slots, so the presence of sensor nodes can be abstracted and modeled as making time slots shorter. Using this abstraction, compositional verification of WSAN applications become feasible as only one node which is in communication with the central node has to be considered for networks in any size. Listing 5 shows the model of the TDMA communication protocol. The periodic behavior of TDMA slot is handled by `handleTDMASlot` message server which sets and unsets `inActivePeriod` to show that whether the node is in its allotted time slot. Upon entering into it's slot, a device checks for pending data to send (line 24) and schedules `handleTDMASlot` message to leave the slot (line 23). On the other hand, when `CPU` sends a packet (message) to a `RCD`, the message is added to the other pending packets which are waiting for the next allotted time slot. `tdmaSlotSize` is the predefined size of the tdma slots, and `currentMessageWaitingTime` is the waiting time of this message in the bag of its receiver.

For the sake of simplicity, the details of `RCD` are omitted in Listing 5. The complete source code (which implements the B-MAC protocol) is available on the Rebeca web page [1].

B-MAC [23] is a CSMA-based technique that utilizes low power listening and an extended preamble to achieve low power communication. To this end, an awake and a sleep period is defined for each node of the network, and each node changes its state from awake to sleep (and vice versa) using its local independent schedule. During the awake period, a node samples the medium and if a preamble for sending data is detected it remains awake to receive the data. So, if a node of the network wishes to send a packet, it precedes the data packet with a preamble that is longer than the sleep period of the receiver. Using extended preamble, a sender is assured that at some point during the preamble the receiver will wake up, detect the preamble, and receive the data. In the case of WSAN applications, as the receiver of data is a central node which does not have any power consumption concern, the receiver is assumed always to be awake. Therefore, the only way that sensor nodes may interfere each other is when a collision occurs in sending data from more than one sensor node. Collisions are avoided by requiring nodes to wait for a time called Backoff time before trying to access the channel after transmission failures. Therefore, having more sensor nodes only results in having more collisions.

Listing 5: Reactive class of the `RCD`

```
env int OnePacketTT = 7; // ms (transmission time)

reactiveclass RCD (3) {
  knownrebecs { Ether ether; }
  statevars {
    byte id;
    int slotSize, sendingPacketsNumber;
    boolean inActivePeriod, busyWithSending;
    RCD receiverDevice;
  }
  RCD(byte myId) {
    id = myId;
    inActivePeriod = false;
    busyWithSending = false;
    sendingPacketsNumber = 0;
    receiverDevice = null;
    if (id != 0) { handleTDMASlot(); }
  }
  msgsrv handleTDMASlot() {
    inActivePeriod = !inActivePeriod;
    if(inActivePeriod) {
        int cmwt = currentMessageWaitingTime;
        assertion(tmdaSlotSize - cmwt > 0);
        self.handleTDMASlot() after(tmdaSlotSize-cmwt);
        self.checkPendingData();
    } else {
        self.handleTDMASlot() after((tmdaSlotSize *
             (numberOfNodes - 1))- cmwt);
    }
  }
  msgsrv send(RCD receiver, int packetsNumber) {
    assertion(receiverDevice == null);
    sendingPacketsNumber = packetsNumber;
    receiverDevice = receiver;
    self.checkPendingData();
  }
  msgsrv checkPendingData() { ... }
  msgsrv receiveResult(boolean result) { ... }
}
```

Based on this fact, the presence of sensor nodes can be abstracted and modeled as the possible number of collisions before a data communication is performed successfully. Using this abstraction, compositional verification of WSAN applications become feasible as only one sensor node which is in communication with the central node has to be considered for networks in any size. Any data transmission of this sensor node may encounter a collision. The maximum number of the collisions is the number of sensor nodes in the model. So, in the Rebeca code for `RCD`, for each data transmission we have a non-deterministic choice between a successful transmission or a collision. During model checking, in the case of collision, data transmission with zero, one, ..., up to n collisions are considered where n is the number of sensor nodes. The Timed Rebeca model of this protocol is available on the Rebeca web page [1].

Once a complete model of the distributed application has been created, the Afra model checking tool can verify whether the schedulability properties hold in all reachable states of the system. If there are any deadline violations, a counterexample will be produced, indicating the path—sequence of states from an initial configuration—that results in the violation. This information can be helpful with changing the system parameters, such as increasing the TDMA time slot length, to prevent such situations.

5. EXPERIMENTAL RESULTS AND A REAL-WORLD CASE STUDY

We examined the applicability of our approach using a WSAN model intended for use in structural health monitoring and control (SHMC) applications.[2] Wireless sensors deployed on civil structures for SHMC collect high-fidelity data such as acceleration and strain. Structural health monitoring (SHM) involves identifying and detecting potential damages to the structure by measuring changes in strain and vibration response. SHM can also be employed with *structural control*, where it is fed into algorithms that control centralized or distributed control elements such as active and semi-active dampers. The control algorithms attempt to minimize vibration and maintain stability in response to excitations from rare events such as earthquakes, or more mundane sources such as wind and traffic. The system we examine has been implemented on the Imote2 wireless sensor platform [18], which features a powerful embedded processor, sufficient memory size, and a high-fidelity sensor suite required to collect data of sufficient quality for SHMC purposes. These nodes run the TinyOS operating system, supported by middleware services of the Illinois SHM Services Toolsuite [12].

This flexible data acquisition system can be configured to support real-time collection of high-frequency, multi-channel sensor data from up to 30 wireless smart sensors at frequencies up to 250 Hz. As it is designed for high-throughput sensing tasks that necessitate larger networks sizes with relatively high sampling rates, it falls into the class of *data-intensive sensor network applications*, where efficient resource utilization is critical, since it directly determines the achievable scalability (number of nodes) and fidelity (sampling frequency) of the data acquisition process. Configured on the basis of network size, associated sampling rate, and desired data delivery reliability, it allows for near-real-time acquisition of 108 data channels on up to 30 nodes—where each node may provide multiple sensor channels, such as 3-axis acceleration, temperature, or strain—with minimal data loss. In practice, these limits are determined primarily by the available bandwidth of the IEEE 802.15.4 wireless network and sample acquisition latency of the sensors. The accuracy of estimating safe limits for sampling and data transmission delays directly impacts the system's efficiency.

[2]The Timed Rebeca code of this case study, some complimentary shell scripts, the model checking toolset, and the details of the specifications of the state spaces in different configurations are accessible from the Rebeca homepage [1].

To illustrate the applicability of this work, we considered applications where achieving the highest possible sampling rate that does not result in any missed deadline is desired. This is a very common requirement in WSAN applications in the SHMC domain in particular. We begin by setting the value of `OnePacketTT` to 7ms (i.e., the maximum transmission time of this type of applications) and fixed the value of `sensorTaskDelay`, `miscPeriod`, and `miscTaskDelay` to some predefined values. In addition to the sampling rate, the number of nodes in the network and the packet size remain variable. By assuming different values for the number of nodes and the packet size, different maximum sampling rates are achieved, shown as a 3D surface in Figure 2. As shown in the figure, higher sampling rates are possible when the buffer size is set to a larger number (there is more space for data in each packet). Similarly, increasing the number of nodes decreases the sampling rate: in competition among three different parameters of Figure 2, the cases with the maximum buffer size (i.e., 9 data points) and minimum number of nodes (i.e., 1 node) results in the highest possible maximum sampling rates. Decreasing the buffer size or increasing the number of nodes, non-linearly reduces the maximum possible sampling rate.

A server with Intel Xeon E5645 @ 2.40GHz CPUs and 50GB of RAM, running Red Hat 4.4.6-4 as the operating system was used as the model checking host. We varied the size of the state space from less than 500 states to more than 140K states, resulting in model checking times ranging from 0 to 6 seconds. In comparison to the time consumption of the other approaches, the amount of time needed for model checking of the model is fairly limited. It is because of the fact that using the compositional approach results in having a limited number of actors which avoids generating huge state spaces. Analyzing the specifications of the state spaces, some relations between the size of the state spaces and the configurations of the models are observed. For example, the largest state spaces correspond to configurations where `sensorTaskDelay`, `bufferSize`, and `numberOfNodes` are set to large values. The parameters used in our analysis of configurations were determined through a real-world installation of an SHMC application. Our results show that the current manually-optimized installation can be tuned to an even more optimized one: by changing the configuration, the performance of the system can be safely improved by another 7% percent.

6. LIMITATIONS AND FUTURE WORKS

In this paper, we only addressed the schedulability analysis of WSAN components and did not consider the interference on the wireless channel issues (the details of communication protocols). Here, we assumed that there is a reliable wireless infrastructure in the application which provides guaranteed delivery of messages, which is a reasonable assumption for a wide range of deployments of structural health monitoring and control systems. However, this work can be extended by taking the details of communication protocols into account together with noises and unreliability of wireless communication which results in errors. This way, only Ether and RCD actors have to be modified to contain the details of the protocols. Note that the implementation of the chosen MAC protocol as well as the interaction of the processing hardware with the transmitter has to be added to RCD to take hardware and software into account and provide combined analysis of the underlying hardware infrastructure as well as the application software. Other different assumptions, including fairness in access to B-MAC, time drift of actors, and uncertainties, can be added. Note that extending the number of modeled MAC layer protocols also can be performed as a future work of this paper. Comparing the efficiency of MAC protocols in different cases to study their characteristics will be one of the outcomes of this extension.

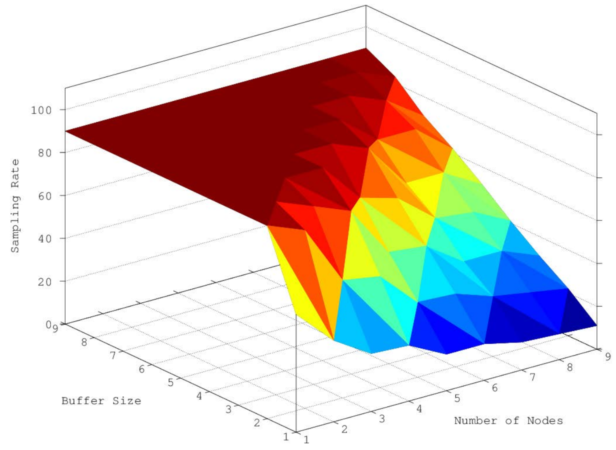

Figure 2: The maximum sampling rate in case of using TDMA protocol and setting the value of `sensorTaskDelay` to 2ms

On the other hand, some WSAN applications also exhibit probabilistic behaviors which are not discussed in this paper. Also, in many soft real-time systems it is desirable to know whether the application does not violate any deadlines with at least a given probability. This is particularly important when deadline violation probability is very small, but requires significant extra resource allocation to be avoided completely. In resource-constrained WSAN environments, the price of such safety guarantee may be too high. To address these cases, we are going to extend this work by using Probabilistic Timed Rebeca [13] for modeling WSAN application and benefiting from combining performance evaluation with functional verification of models. This way, we develop one model for the purposes of model checking, performance evaluation, and probabilistic model checking.

We also planned to extend this work in two different dimensions which requires extending the support of Timed Rebeca and the Afra toolset. The first direction is extending the time model of Timed Rebeca from discrete time to dense time. This way, instead of putting one integer number or a deterministic expression with finite values as the parameter of timing primitives, an integer interval can be used. So, a nondeterministically chosen value (i.e. a real number) from the given interval would be the concrete value of its associated primitive. Although analysis of dense time systems is more complex than discrete time systems and the current features of Timed Rebeca is sufficient for modeling of WSAN applications, modeling the details of communication protocol (mentioned at the beginning of this section) requires dense time in some cases. This extension can go further to support probability distributions as parameters of the timing primitives and provides stochastic analysis of

WSAN applications together with their correctness analysis.

The second direction is realized by allowing modelers to express operation costs in their models. In case of WSAN applications, associating costs with operations can be interpreted as the power consumption of operations. So, developing a WSAN model which is enriched with costs makes its modelers able to perform both performance evaluation and optimization analysis over the model.

7. RELATED WORK

Three different approaches have been used for analysis of WSANs: system simulation, analytical approach, and formal verification.

System Simulation and Analytical Approach.

Simulation of WSAN applications is useful for their early design exploration. Simulation toolsets for WSANs have enabled modeling of networks [17], power consumption [28], and deployment environment [31]. Simulators can adequately estimate performance of systems and sometimes detect conditions which lead to deadline violations. But even extensive simulation does not guarantee that deadline misses will never occur in the future [5]. For WSAN applications with hard real-time requirements this is not satisfactory. Moreover, none of available simulators is suitable for the analysis WSAN application software.

On the other hand, a number of algorithms and heuristics have been suggested for schedulability analysis of real-time systems with periodic tasks and sporadic tasks with constraints, e.g. [20]. But, although these classic techniques are efficient in analyzing schedulability of a wide range of real-time systems, their lack of ability to model random tasks make them inappropriate for WSAN applications. To the best of our knowledge, there is no work on using analytical approach for analysis of WSAN applications.

Formal Verification.

Real-time model checking is an attractive approach for schedulability analysis with guarantees [5]. Model checking tools systematically check whether a model satisfies a given property [4]. The strength of model checking is not only in providing a rigorous correctness proof, but also in the ability to generate counter-examples, as diagnostic feedback in case a property is not satisfied. This information can be helpful to find flaws in the system. Norström et al. suggest an extension of timed automata to support schedulability analysis of real-time systems with random tasks [21]. Fersman et al. studied an extension of timed automata which its main idea is to associate each location of timed automata with tasks, called task automata [10].

TIMES [3] is a toolset which is implemented based on the approach of Fersman et al. [9] for analysis of task automata using UPPAAL as back-end model checker. TIMES assumes that tasks are executed on a single processor. This assumption is the main obstacle against using TIMES for schedulability analysis of WSAN applications, which are real-time distributed applications. De Boer et al. in [7] presented a framework for schedulability analysis of real-time concurrent objects. This approach supports both multi-processor systems and random task definition, which are required for schedulability analysis of WSAN applications. However, asynchronous communication among concurrent elements of WSAN application results in generation of complex behavioral interfaces which lead to a state space explosion even for small size examples.

Real-Time Maude is used in [22] for performance estimation and model checking of WSAN algorithms. The approach supports modeling of many details such as communication range and energy use. The approach requires some knowledge of rewrite logic. Our tool may be easier to use by engineers unfamiliar with rewriting logic: our language extends straight-forward C-like syntax with actor concurrency constructs and primitives for sensing and radio communication. This requires no formal methods experience from the WSAN application programmer, as the language and structure of the model closely mirror those of the real application.

8. CONCLUSION

We have shown a compositional approach for modeling and schedulability analysis of WSAN applications. WSAN applications are very sensitive to their configurations: the effects of even minor modifications to configurations must be analyzed. However, analyzing the real installation of a WSAN application with many concurrently executing sensors nodes, with a high level of confidence, is impossible. In this paper, we showed that our approach provides an accurate view of the behavior of a WSAN application and its interaction with the operating system and distributed middle-ware services.

Our realistic—but admittedly limited—experimental results support the idea that the use of compositional modeling and analysis approach may result in more robust WSAN applications. This would greatly reduce development time as many potential problems with scheduling and resource utilization may be identified early.

An important direction for future research is the addition of probabilistic behavior analysis support to the tool. In many non-critical applications, infrequent scheduling violations may be considered a reasonable trade-off for increased efficiency in the more common cases. Development of a probabilistic extension is currently underway.

Acknowledgments

The work on this paper has been supported in part by the project "Self-Adaptive Actors: SEADA" (nr. 163205-051) of the Icelandic Research Fund.

9. REFERENCES

[1] Rebeca Formal Modeling Language. http://www.rebeca-lang.org/.

[2] G. A. Agha. *ACTORS - a model of concurrent computation in distributed systems.* MIT Press series in artificial intelligence. MIT Press, 1990.

[3] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: A tool for schedulability analysis and code generation of real-time systems. In *FORMATS*, volume 2791 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2003.

[4] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, 1999.

[5] A. David, J. Illum, K. G. Larsen, and A. Skou. *Model-Based Design for Embedded Systems*, chapter Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1, pages 93–119. CRC Press, 2010.

[6] F. S. de Boer, T. Chothia, and M. M. Jaghoori. Modular Schedulability Analysis of Concurrent Objects in Creol. In *Fundamentals of Software Engineering, Third IPM International Conference, FSEN 2009, Kish Island, Iran, April 15-17, 2009, Revised Selected Papers*, volume 5961 of *Lecture Notes in Computer Science*, pages 212–227. Springer, 2009.

[7] F. S. de Boer, M. M. Jaghoori, and E. B. Johnsen. Dating concurrent objects: Real-time modeling and schedulability analysis. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.

[8] A. El-Hoiydi. Spatial TDMA and CSMA with preamble sampling for low power ad hoc wireless sensor networks. In *Proceedings of the Seventh IEEE Symposium on Computers and Communications (ISCC 2002), 1-4 July 2002, Taormina, Italy*, pages 685–692. IEEE Computer Society, 2002.

[9] E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Schedulability Analysis of Fixed-Priority Systems Using Timed Automata. *Theor. Comput. Sci.*, 354(2):301–317, 2006.

[10] E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2002.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Notices*, 35:93–104, November 2000.

[12] Illinois SHM Services Toolsuite. http://shm.cs.illinois.edu/software.html.

[13] A. Jafari, E. Khamespanah, M. Sirjani, H. Hermanns, and M. Cimini. Ptrebeca: Modeling and analysis of distributed and asynchronous systems. *Sci. Comput. Program.*, 128:22–50, 2016.

[14] E. Khamespanah, K. Mechitov, M. Sirjani, and G. A. Agha. Schedulability analysis of distributed real-time sensor network applications using actor-based model checking. In *Model Checking Software - 23rd International Symposium, SPIN 2016, Co-located with ETAPS 2016, Eindhoven, The Netherlands, April 7-8, 2016, Proceedings*, volume 9641 of *Lecture Notes in Computer Science*, pages 165–181. Springer, 2016.

[15] E. Khamespanah, M. Sirjani, Z. Sabahi-Kaviani, R. Khosravi, and M. Izadi. Timed rebeca schedulability and deadlock freedom analysis using bounded floating time transition system. *Sci. Comput. Program.*, 98:184–204, 2015.

[16] E. Khamespanah, M. Sirjani, M. Viswanathan, and R. Khosravi. Floating Time Transition System: More Efficient Analysis of Timed Actors. In editors, *Formal Aspects of Component Software - 12th International Symposium, FACS 2015, Rio de Janeiro, Brazil, October 14-16, 2015*, Lecture Notes in Computer Science. Springer, 2016.

[17] P. Levis, N. Lee, M. Welsh, and D. E. Culler. TOSSIM: accurate and scalable simulation of entire tinyos applications. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, Los Angeles, California, USA, November 5-7, 2003*, pages 126–137. ACM, 2003.

[18] L. Linderman, K. Mechitov, and B. F. Spencer. TinyOS-Based Real-Time Wireless Data Acquisition Framework for Structural Health Monitoring and Control. *Structural Control and Health Monitoring*, 2012.

[19] G. Lipari and G. Buttazzo. Schedulability analysis of periodic and aperiodic tasks with resource constraints. *Journal of Systems Architecture*, 46(4):327–338, 2000.

[20] J. W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

[21] C. Norström, A. Wall, and W. Yi. Timed automata as task models for event-driven systems. In *RTCSA*, pages 182–189. IEEE Computer Society, 1999.

[22] P. C. Ölveczky and S. Thorvaldsen. Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in real-time maude. *Theor. Comput. Sci.*, 410(2-3):254–280, Feb. 2009.

[23] J. Polastre, J. L. Hill, and D. E. Culler. Versatile low power media access for wireless sensor networks. In Stankovic et al. [30], pages 95–107.

[24] S. Ren and G. Agha. RTsynchronizer: Language Support for Real-Time Specifications in Distributed Systems. In *Workshop on Languages, Compilers, & Tools for Real-Time Systems*, pages 50–59. ACM, 1995.

[25] A. H. Reynisson, M. Sirjani, L. Aceto, M. Cimini, A. Jafari, A. Ingólfsdóttir, and S. H. Sigurdarson. Modelling and Simulation of Asynchronous Real-Time Systems using Timed Rebeca. *Sci. Comput. Program.*, 89:41–68, 2014.

[26] Z. Sharifi, S. Mohammadi, and M. Sirjani. Comparison of NoC Routing Algorithms Using Formal Methods. To be published in proceedings of PDPTA'13, 2013.

[27] Z. Sharifi, M. Mosaffa, S. Mohammadi, and M. Sirjani. Functional and performance analysis of network-on-chips using actor-based modeling and formal verification. *ECEASST*, 66, 2013.

[28] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In Stankovic et al. [30], pages 188–200.

[29] B. F. Spencer Jr., H. Jo, K. Mechitov, J. Li, S.-H. Sim, R. Kim, S. Cho, L. Linderman, P. Moinzadeh, R. Giles, and G. Agha. Recent advances in wireless smart sensors for multi-scale monitoring and control of civil infrastructure. *Journal of Civil Structural Health Monitoring*, pages 1–25, 2015.

[30] J. A. Stankovic, A. Arora, and R. Govindan, editors. *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys 2004, Baltimore, MD, USA, November 3-5, 2004*. ACM, 2004.

[31] S. Sundresh, W. Kim, and G. Agha. Sens: A sensor, environment and network simulator. In *Proceedings 37th Annual Simulation Symposium (ANSS-37 2004), 18-22 April 2004, Arlington, VA, USA*, pages 221–228. IEEE Computer Society, 2004.