Modeling and Analyzing Wireless Sensor Networks with VeriSensor

Yann Ben Maissa, Fabrice Kordon, Salma Mouline, Yann Thierry-Mieg

Modeling and Analyzing Wireless Sensor Networks with VeriSensor

Yann Ben Maissa[1,2], Fabrice Kordon[2], Salma Mouline[1], and Yann Thierry-Mieg[2]

[1] LRIT – CNRST URAC29, Université Mohammed V-Agdal
4, Avenue Ibn Battouta, B.P. 1014 RP, Rabat Maroc,
mouline@fsr.ac.ma,
[2] LIP6 – CNRS UMR7606, Université P. & M. Curie 4, place Jussieu, 75005 Paris, France
Yann.Ben-Maissa@lip6.fr, Fabrice.Kordon@lip6.fr, Yann.Thierry-Mieg@lip6.fr

Abstract. A Wireless Sensor Network (WSN), made of distributed autonomous nodes, is designed to monitor physical or environmental conditions. WSNs have many application domains such as environment or health monitoring. Their design must consider energy constraints, concurrency issues, node heterogeneity, while still meeting the quality requirements of life-critical applications. Formal verification helps to obtain WSN reliability, but usually requires a high expertise, which limits its adoption in industry.
This paper presents VeriSensor, a domain specific modeling language (DSML) for WSNs offering support for formal verification. VeriSensor is designed to be used by WSN experts. It can be automatically translated into a formal specification for model checking. We present the language, its translation, show how they work on a simple case study, and illustrate how several metrics and properties relevant to the domain can be evaluated.

Keywords: wireless sensor networks, domain specific modeling languages, model driven engineering, formal verification

1 Introduction

Context Wireless sensor networks (WSNs) are composed of distributed autonomous nodes, containing programs and sensors to monitor physical or environmental conditions. Each node is a small physical device embedding sensors, a small CPU, a battery, a wireless transceiver and an antenna for communication. WSNs are useful in many contexts, such as environment or health monitoring, thus being a hot topic [14, 8].

The design of WSNs is complex and error-prone due to their numerous constraints:

- *lifetime* is a crucial preoccupation (even more important than quality of service [3]). Overall lifetime of the WSN usually depends on sensor nodes lifetime because nodes have limited battery power.
- *concurrency and asynchrony* lead to important issues such as interleaving of actions and race conditions.
- *heterogeneity*, because WSNs may contain various types of nodes, each having different characteristics (embedded sensors, wireless range, battery capacity, etc.).
- *limited resources*, because nodes have limited CPU and memory capacities.

Problem When WSNs are intended to handle critical functions, verification and validation must be performed to reach a significant confidence in such systems [12, 7]. Several propositions in that direction have emerged in recent years.

Case studies using Formal Verification. Formal methods have been applied on case studies to verify some relevant properties for WSNs. For instance, [12] uses Real-Time Maude, [11] uses the language IF and the model-checker Kronos, [19] uses UPPAAL.

While these studies show the practical and industrial relevance of performing formal analysis on WSNs, they use ad-hoc modeling of the system by an expert in both WSNs and formal verification. This increases the design and verification costs of WSNs. Moreover, complex verification "tricks" must be elaborated to achieve the verification goals, creating a gap between the formal specifications and the real system.

Language-based approaches. Current trends in software engineering show the emergence of model-driven engineering (MDE): a model of the system is expressed using a domain specific modeling language (DSML) providing concepts of the domain. Then, using model transformation technologies, executable code or simulation models can be automatically produced.

Such DSMLs dedicated to WSNs ease their modeling by domain experts. However, they currently do not support formal analysis. VisualSense [4] for instance only allows simulation which is useful during the early design stages, but may not catch rare unexpected events. Baobab [2], Matilda [21], and Medwsa [20] provide code generators that produce executable artifacts to be deployed on the physical system.

Unfortunately, these DSMLs often have a very detailed level of specification (such as wireless signal propagation characteristics), including non-linear parts that can only be simulated in practice. So, if they are adapted for code generation or simulation, they generate a high combinatorial explosion and are thus not suitable for verification.

Contribution This paper presents VeriSensor, a DSML for WSNs and its mapping to a formal language for verification and analysis. VeriSensor has the features of an architectural description language (ADL [10]) adapted to a modular description of WSNs.

VeriSensor offers "natural" modeling of a WSN to domain experts by providing high-level concepts that capture the main use cases of such systems – periodic data collection, query-based processing, etc. [22, 9]. VeriSensor can be transformed into a discrete formal model supporting analysis: Instantiable Transition Systems (ITS) [18]. At this stage, VeriSensor is not intended for code generation.

To illustrate its capabilities, we use VeriSensor on a small example. Analysis is performed using our own tools, based on the ITS library.

Contents Section 2 gives an overview of VeriSensor. Section 3 presents the language concepts together with the biomedical area network (BAN) case study [13] used as a running example. Section 4 explains the mapping of VeriSensor into ITS. Then, section 5 presents the analysis results we compute on the case study.

2 Overview of VeriSensor

A VeriSensor specification is composed of the definition of the nodes themselves, a description of the physical environment in which the nodes evolve, and the deployment of the system (see Fig. 1).

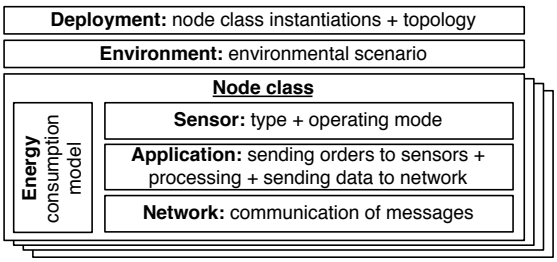


Fig. 1: Structure of a VeriSensor specification

Description of the nodes There can be several classes of nodes in a WSN (e.g. in a heterogeneous network), each one having its own characteristics such as:

- its sensors (which physical quantities to be measured and how they are captured),
- its application operating mode (periodic data collection, query-based processing, etc.) and the way it manipulates data,
- its interface with the network (wireless range, routing, etc.),
- the energy consumption model.

These characteristics are described through four orthogonal *dimensions*: sensor, application, network and energy. Dimensions describe *independent* aspects of the system.

Several node classes can share common dimensions and a node class can be instantiated several times when several nodes have the same characteristics.

Environment Model It defines physical quantities as a function of space (x, y, z) and time (t). Thus, the designer may describe a particular scenario in which the WSN evolves. These scenarios are used to test qualitative properties of models on given problem instances. A given environment represents a particular situation in which a given behavior of the WSN is expected.

Deployment Model It defines how instances of node classes are spread in the physical environment and may change position over time[3]. Engineers use this model to define the topology of the system (number of instances per class and their coordinates) as well as the logical routing of messages among the nodes.

Structuration in VeriSensor The various dimensions are defined separately in VeriSensor to support *modularity* and *reusability* of WSNs components. The deployment model of a system is the entry point of a VeriSensor model. It defines the Environment model and instantiates all nodes from the definition of their classes.

3 Modeling with VeriSensor

This section presents VeriSensor through the specification of a case study.

[3] We do not yet support mobility in our approach but this is a natural extension that is semantically possible in VeriSensor.

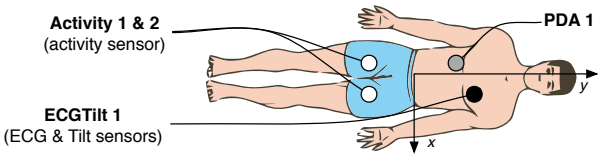


Fig. 2: The Body Area Network (BAN)

3.1 The Body Area Network (BAN)

Our case study takes place in the context of home medical monitoring of patients who need constant care but can stay out of hospitals. Home medical monitoring allows to avoid hospitalization, which is as good for medical staff as for their patients.

The Body Area Network [13] is part of a wireless health monitoring system. It is composed of (see Fig. 2): *i)* a set of sensor nodes capable of sensing, processing and communicating vital signs to a personal server ; *ii)* a Portable Digital Assistant (PDA) that forwards patient data to a medical center through internet (3G or WIFI).

The BAN monitors the vital signs of patients recovering from a heart attack. It checks whether a patient is exercising regularly as recommended by the doctors. WSNs, due to their small size and wireless nature, reduce system intrusiveness in patient's lives.

As shown in Fig. 2, two redundant activity nodes detect periods of physical exercise (when the body activity level is above 8 $Watts.kg^{-1}$) while a third one periodically collects both heartbeat with an electrocardiogram (ECG) sensor and the tilt (i.e. upper body orientation) in terms of the absolute angle relative to a vertical position.

The system designer (i.e. the end-user of VeriSensor) wants to assess some critical aspects of his system. To do so, he needs to evaluate properties such as:

- p_1 evaluate which node limits the system lifetime according to a given scenario,
- p_2 identify scenarios leading to undesirable situations that should be avoided,
- p_3 check that the system behaves as expected by "replaying" existing situations identified by doctors,
- p_4 compare alternative hardware solutions according to their characteristics (energy consumption of sensors, processing duration, etc.),

3.2 Modeling the BAN in VeriSensor

This section illustrates the VeriSensor syntax and structure through the modeling of the BAN case study. Here, we follow a "path" going from the more general aspects of the system (its elements) up to implementation of some nodes and the description of its environment.

The Deployment Model Fig. 3 shows the deployment parameters of the BAN system. Each node instance is parameterized by its position (shown on fig. 3) and next hop. For instance, the only node of class `ECGTilt` is located at position $\langle 0.1, 0.4, 0 \rangle$ (when a position parameter is unspecified, its value is 0) and routes messages to the `pda1` instance. Distances are expressed in meters.

The Node Class Model A node class specifies the physical characteristics of a node to be instantiated. It relates the data dispatched in the four dimensions: sensing, application, network, and energy (Fig. 4, left).

```
System BAN {environment => HumanBody;
            ECGTilt     => ecgtilt1   (x=0.1, y=0.4, nextHop = pda1);
            Activity    => activity1  (x=-0.3, y=0.1, nextHop = pda1),
                           activity2  (x=-0.3 , y=-0.1, nextHop = pda1);
            PDA         => pda1       (x=-0.1, y=0.3, nextHop = null);}
```

Fig. 3: Deployment of the BAN system

In the case study, we only consider static routing based on the **nextHop** parameter defined in the deployment model. The `XNetwork` dimension reflects this choice and is used by all nodes of the BAN as specified in Fig. 3.

Fig. 4 (right) describes the sensors of `ECGTilt`. In our study, this node class samples the upper body orientation (Tilt) and the heartbeat (Heartbeat). Sensors are described through their main technical characteristics: the measured physical quantity startup time (i.e. the time for the sensor to be operational after being turned on), and its capture time (i.e. the time for the sensor to sense the value). For instance, ECGSensor measures the heartbeat and starts-up in 1 time unit. Physical quantities are defined in a dedicated model (see Fig. 6, left) contained in the file `types.def`.

Each physical quantity q is connected to the environment which must provide a function returning the values of q at the coordinates of the node instance and for the current time (at any time). So, in Fig. 4 (right), when `ECGSensor` samples a value, it invokes the corresponding function returning the `Heartbeat` from the Environment dimension. There is one such function per physical quantity of the system.

Fig. 5 (left) shows the application dimension of ECGTilt. In VeriSensor, nodes have several typical behaviors provided as a parameter of the definition. Here, `ECGTilt` behaves in "collect" mode (keyword collectNode in the figure): this periodic data collection is parameterized by a sensing period (i.e. the time between two samples), a processing period (i.e. the time between two processing of the sampled data), and a sending period (i.e. time between two emissions of the processed data). For instance, `Tilt` is sampled every 4 time units, processed every 8 time units, and sent every 16 time units.

Fig. 5 (right) shows the energy dimension that describes the initial power stored in the battery (initial) and defines the consumption of dedicated actions: reception (message reception), emission (message sending), processing (processing of sampled data), and sensing (sample acquisition).

```
NodeClass ECGTilt {
   sensing => ECGTsensing;
   application => ECGTApplication;
   network => XNetwork;
   energy => ECGTEnergy;}
```

```
include types.def;
sensing ECGTsensing {
   sensor ECGSensor (
            physical_quantity = Heartbeat,
            startup_time = 1,
            capture_time = 1);}
   sensor TiltSensor (
            physical_quantity = Tilt,
            startup_time = 0,
            capture_time = 1);
```

Fig. 4: `ECGTilt`, the node class (left) and its sensing dimension (right)

```
application (collectNode) ECGTApplication {
   physical_quantity HeartBeat (sensing_period = 13,
      processing_period = 13, sending_period = 13);
   physical_quantity Tilt (sensing_period = 4,
      processing_period = 8, sending_period = 16);}
```

```
energy ECGTEnergy {
   initial = 1000;
   reception = 4;
   emission = 5;
   processing = 3;
   sensing = 2 ;}
```

Fig. 5: `ECGTilt`, its application (left) and energy dimensions (right)

The Physical quantities Model This model describes physical quantities as discrete ranges of values (see Fig. 6 left). The underlying semantics is the one of discrete event systems, so, continuous values must be mapped to an integer range. This mapping is user-defined; the designer must evaluate the trade-off between precision of quantities units and the analysis complexity.

The Environment Model It defines the evolution of each physical quantity in the model (see Fig. 6, right) in a given scenario. Thus, it provides a function that is bound to each sensor sampling the corresponding physical quantity (e.g `HeartBeatFunc` is bound to the ECGSensor defined in Fig. 4).

In our example of environment model, values of HeartBeat depend on time only. Since there are two Activity sensors, `ActivityFunc` can use the node coordinates to provide different values to each sensor in nodes `activity1` and `activity2`. Here, values in `HeartBeatFunc` are deduced from the thresholds of the application: for instance, any value above 95 is considered a situation where the patient exercises; then, these values are abstracted to the threshold constant. Our simplified example illustrates a common situation where WSNs designers generate such a function from observed traces of the system activity. No parsing of traces is provided since these are too "system dependent". To extend an existing trace, a cyclic behavior may be specified.

For some properties of interest such as worst case scenarios, instead of using the user supplied environment we can use a "free" unconstrained environment, which might return any value at any time. The *clear separation* between the input conditions (environment) and the system specification is important in the analysis phase described below.

```
type HeartBeat is 0..200;
type Tilt is 0..180;
type Activity is 0..15;
```

```
include types.def;
environment cyclic HumanBody {context {}
   body {
      cycle 60; // cyclic behavior (in time units)
      HeartBeat function HeartBeatFunc (x,y,z,t) {
         if (0 <= t and t < 10) then return(95);
         elsif (10 <= t and t < 14) then return(40);
         else return (75);}
      Tilt function TiltFunc (x,y,z,t) {...}
      Activity function ActivityFunc (x,y,z,t) {...}}
```

Fig. 6: Physical quantities definition (left) and an example of Environment Model (right)

4 Formal Analysis of VeriSensor specifications

Formal analysis by model checking of a system is a powerful technique that allows to capture subtle defects as well as to reason about worst case scenarios and occurrence of rare events by exhaustively analyzing all possible behaviors. However it is limited in the scale of the systems it can analyze due to the combinatorial state space explosion characteristic of concurrent asynchronous systems. To partly overcome this problem, techniques and tools have emerged such as SAT solvers [6] or shared decision diagrams [5].

However formal models are usually limited to a low level specification of the system transition relation, that describes the state space generator.

Since WSNs are highly time driven and complex, we need a tool supporting a large amount of concurrency, some notion of time constraints and able to tackle combinatorial explosion. To achieve this, we rely on our own preexisting tool: Instantiable Transition Systems (ITS) [18] and their recent extension that supports discrete time [15]. The ITS model checker is general and efficient: it relies on a powerful decision diagram library to cope with the complexity of large systems. ITS also provide a way to define a structured and hierarchical specification of a system and a notion of behavior instantiation. They were previously experimented to analyze UML activity diagrams through a model transformation approach [17] similar to the one outlined here.

4.1 The Underlying Formal Model

This section first gives an informal overview of the underlying formal notations for VeriSensor. There are two formalisms involved: labeled time Petri nets to describe elementary behavior and ITS to structure the specification. We only provide here an intuitive definition (see [18, 15] for a formal presentation).

Instantiable Transition Systems ITS allow hierarchical and compositional modeling, through a notion of type and instance and an application of the composite design pattern at a behavioral level. A type has an interface, defined as a set of action labels, and some definition of its internal behavior. Similarly to component oriented models, an ITS *composite* is a type that contains *instances* of ITS *types*.

Figure 7 shows a simple example of a composite ITS type. The system offers one interface, begin, that is synchronized with the start interfaces of the nested components (Client and Server). This system contains a local transition (ε) that only has a local effect and is built on the synchronization of send and get interfaces. Client and Server are

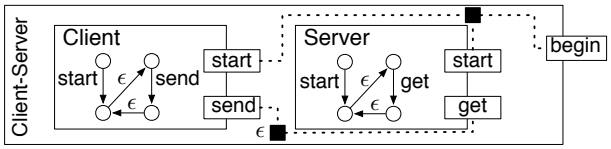


Fig. 7: Small example of composite-ITS

elementary components that contain an automaton where local transitions are labeled by ε too. In practice, we use labeled time Petri nets to define elementary ITS types.

Labeled Time Petri Nets In a Petri net, places (circles) contain tokens representing resources that are consumed by transitions (rectangles) when they fire, producing new tokens. A state of a Petri net assigns to each place of the net an integer representing the number of tokens it contains. In a given state, a transition is enabled if all its input places (connected by an arc from place to transition) contain enough tokens. Each arc may be labeled by an integer that indicates how many tokens are consumed or produced (the value 1 is assumed if there is no annotation). When firing, a transition produces tokens in the places connected by outgoing arcs.

Time Petri nets (TPNs) add a notion of clock to each transition, constrained by an earliest and latest firing time noted $[\alpha, \beta]$. As soon as a transition is enabled, the associated clock starts. This transition cannot fire before α time units have elapsed and must occur if the transition's clock reaches β. Hence a transition with $[0, \infty]$ can occur at any date if it is enabled, like normal Petri nets. This is assumed to be the default values and is not explicitly shown in the figures.

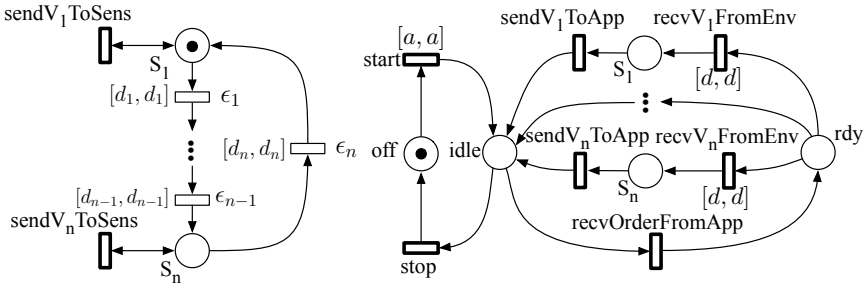
The time model is discrete: a special transition *elapse* represents the evolution of time by one unit. All clocks evolve simultaneously when *elapse* is fired.

Labels add a notion of interface to Petri nets, where some transitions (represented with thick borders) are called *public* and allow communication with the outside world. These transitions define the ITS interface. *Private* transitions can occur locally, independently from any situation outside the net, and typically represent an autonomous control flow.

4.2 Mapping VeriSensor to a Formal Specification

The mapping of VeriSensor into formal specifications relies on patterns associated to its syntactic elements. It is also based on a set of automatically computed abstractions that help containing the combinatorial explosion due to large datatypes.

The Transformation process To automatically transform the specification into a formal model we define a set of "generic ITS", modeling behavioral patterns that correspond to the VeriSensor execution semantics.



(a) *n*-ary cyclic environment (b) Sensor sampling *n* values

Fig. 8: Two generic ITS, interfaces transitions are outlined in bold

Thus, the transformation process takes parameters in a VeriSensor specification to customize such patterns. Each dimension has its own generic pattern that is hierarchically defined, thus taking benefits from the ITS mechanisms. The final model is obtained by assembling and instantiating these patterns according to the deployment model.

Figure 8 shows two examples of generic ITS. The first one (Fig. 8a) represents the environment as seen by a given node. To obtain this behavior, the environment function $q(x,y,z,t)$ is projected over the coordinates of the node, yielding a function $q(t)$ of time only that is specific to the considered node sensor. This function is finally discretized, and encoded as a series of plateau values that have a certain duration d_i. Each public transition is labeled by a possible value of the physical quantity. The time bound on local transitions (ε) represents the evolution of $q(t)$ as time progresses. The last transition ε_n can be added to represent a cyclic environment. This ITS is parameterized by n, the number of values sent in the cycle, and by d_i for $i \in [1..n]$, the duration for sending these values. Its ITS interface is the set of possible values sendV$_i$ToSens of the physical quantity.

Fig. 8b represents the behavior of a sensor (as for the BAN system in Fig. 4 right) and is parameterized by n, the number of potential values in the physical quantity, a, the startup time, and d, the capture time. Its ITS interface is composed of the control commands (start, stop, recvV$_i$FromEnv , sendV$_i$ToApp).

Although the model size grows with the number of potential values, we control this combinatorial explosion by reducing the domains of physical quantities to the minimum set of representative values that impact the system control flow (see paragraph *Abstraction* below). Moreover, our ITS tool only encapsulates on P/T nets.

The transmission of a value V_k from the environment to the sensor is represented by a synchronization between sendV$_k$ToSens and recvV$_k$FromEnv. The transition sendV$_i$ToApp transmits sampled values back to the application dimension. Because these definitions of the sensor and the environment are clearly separated we can easily associate the specification to any arbitrary environment instead of a fixed scenario. This is done in the deployment model.

Similarly, each dimension has its own parameterized pattern. Some dimension, such as the application dimension of a node class has one pattern per operating mode (data collection, query processing, etc.).

The Energy dimension is modeled by a one-place Petri net. This place's initial marking depends on the initial energy of the node. Transitions (capture, process, send, receive, etc.) consume the number of tokens corresponding to the energy cost of the associated operation. Since operations in a node are synchronized to the energy dimension, the lack of tokens in the energy dimension stops the corresponding node. When all nodes are out of energy, the system cannot execute anymore and reaches a deadlock.

The full node is then defined as a composite ITS that assembles the projection of the environment with the various ITS corresponding to each dimension (sensors, application, network, energy). This composite ITS has an interface allowing transmission of network messages to other nodes. The nodes are then finally instantiated and connected according to the logical network topology.

Figure 9 illustrates the overall elaboration of the final formal specification for the BAN case study based on the VeriSensor architecture. The assembling of a node class

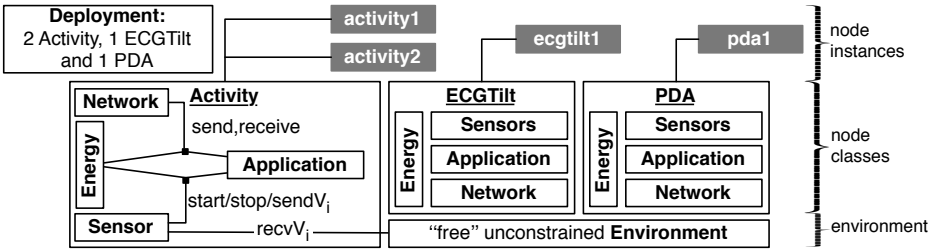


Fig. 9: Structure of the BAN specification according to the deployment of Fig. 3

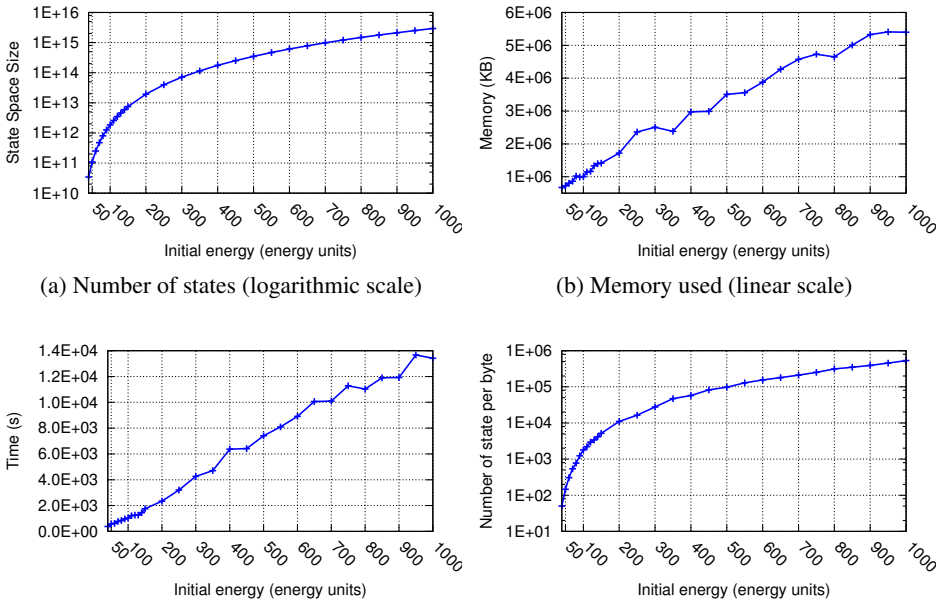
is illustrated for the activity node class. We show how the dimensions interfaces are synchronized one to another. For example, the public transition start is a synchronization between the application, energy and sensor dimensions. This is similar with recvV_i between the sensor and environment dimensions. Let us remind that transitions like recvV_i are instantiated as many times as there are relevant values in the parameters to be exchanged.

Then, each node class is instantiated according to the deployment model. In Fig. 9, there are two activity nodes, one ECGTilt and one PDA. Context variables of each node, describing the node characteristics and its coordinates are initialized according to the deployment model.

Abstractions The final assembling, as described, generates complex models since each potential value of a physical quantity q makes the overall model larger. To avoid this, a structural analysis of the VeriSensor specification allows to automatically abstract the domains of physical quantities to the minimum set of representative values that impact the system control flow. Such techniques are derived from automatic symmetry detection [16] or symbolic trajectory evaluation [1]. The complexity of these techniques is low, since it relies on the size of the specification instead of the size of the state space.

Deriving such abstractions automatically is important because: *i*) they are then correct by construction *ii*) using abstractions does not imply any end-user knowledge of the underlying techniques. For instance, activity nodes only detect whether the patient is exercising (i.e. activity>8, see subsection 3.1) or not, so the domain of the physical quantity activity (i.e. 0 to 15, see Fig. 6 left) is automatically reduced to 2 values: 0 for no physical exercise, 1 for physical exercise.

About the Final Model The resulting model for the BAN case study is composed of 17 ITS-types of which 13 are elementary. The enclosed Petri nets contain 100 places and 81 transitions of which 43 are time constrained. Thus, each state is a vector of 143 variables (places marking + transition clocks). Explicit storage with no optimization of such a state would need 143 integer values, and thus 1.12 Kbyte with a 64 bit representation.



(a) Number of states (logarithmic scale)

(b) Memory used (linear scale)

(c) Computation time (linear scale)

(d) Number of stored states per byte (log scale)

Fig. 10: Evolution of the state space complexity when the initial energy allocated for each node increases (activity sensor period=18 time units for node activity1 and 9 time units for activity2, ECG sensor period=10 time units, Tilt sensor period=29 time units, message emission for every node=5 energy units, message reception for every node=4 energy units, sensing cost for every sensor = 2 energy units, processing sampled data = 3 energy units for every nodes)

5 Analyzing the Case study

This section discusses the analysis we performed on our case study. The ITS representation was generated according to the rules defined in the previous section (a tool is being implemented). From the ITS model this procedure produces, we evaluated the properties identified in section 3.1. All experimentations were done with our own tool based on our ITS library [15].

Prior to this, we discuss the efficiency of this translation with regards to the analysis scalability (based on the parameters values). All experiments were run on a Xeon 64 bits at 2.6 GHz processor.

Analysis Scalability The model of the BAN case study is associated with a "free" unconstrained environment providing all possibles situations from the environment point of view. Thus leading to the analysis of possible situations in the system. Figure 10 shows the evolution of the corresponding state space, its computation time and the memory required to build it, according to the initial energy allowed to the system. In these scenarios, a time unit lasts 1 minute and an energy unit is 50 microjoules. Such interpretation is decided by the designer of the WSN.

As seen in Fig. 10a, the state space grows exponentially, the end of the curve tending to a line in a logarithmic scale. Its representation in memory, as well as the computation time, evolves in a much more favorable way, thus validating the choice of ITS, based on decision diagrams, that already proved its efficiency for such systems [18].

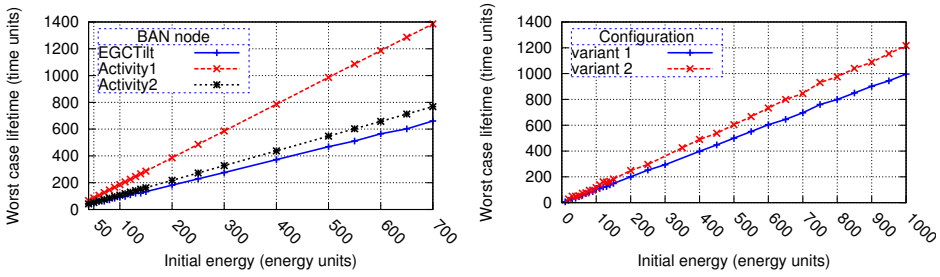
Figures 10b and 10c show the evolution of memory and time required for state space construction according to the initial energy allocated to each node. As shown, we can scale this energy up to 1000 units and still have a reasonable CPU and memory consumption (5.4 GBytes and 3.7 hours). From an industrial point of view, it becomes feasible to process larger values on current high-performance servers.

Considering the memory required to store a state and the size of the state space, our translation into ITS, even if it is yet at a prototype stage, shows encouraging results (up to 5.3×10^5 states per byte as shown in Fig. 10d). Moreover, no particular optimization has been done besides the abstraction automatically computed during the translation, thus avoiding the need for expertise in the underlying formal tools.

This experiment on the BAN shows a good scalability potential for the overall approach. In particular, it shows the verification complexity of reachability properties (e.g. p_2 in section 3.1) that are a reliable way to detect "rare events", difficult to track using classical simulation-based techniques. However, if a Yes/No answer for a reachability property is provided within a reasonable time, we measured that computation of a counterexample takes significantly more time and memory.

Information about the System Lifetime (p_1) Exhibiting the energy consumption of the WSN in the worst case scenario allows the end-user to evaluate a lower bound of the system lifetime. Figure 11a shows the worst case lifetime evolution of the BAN nodes.

To do so, we associate the BAN model with an *unconstrained environment* allowing any action. We thus compute a superset of all the possible behaviors from which we can obtain a worst case scenario. In this model, we search for S_{end}, the set of states where at least one node cannot communicate anymore (its energy is below a constant Min, the minimum energy to send a message). Then, our tool computes the shortest path (i.e. shortest transition sequence) leading from the initial state to a state in S_{end}. To get the corresponding lifetime, we count the occurrences of the *elapse* transition (that let time elapse for 1 time unit). This is the minimal time from the initial state to a state where an observed node cannot communicate anymore.



(a) BAN nodes worst case lifetime

(b) Compared lifetime of activity1

Fig. 11: Lifetime analysis on the BAN case study

The objective is not to provide quantitative information since the initial number of energy units allocated to nodes is not sufficient (Fig. 11a shows a system duration in hours, while, at least, weeks would be needed). However, a designer can get an idea of the most critical component (i.e. the one that fails first) according to various scenarios. This result is complementary of simulation that can tackle longer duration but not in an exhaustive way.

Let us note that, in Fig. 11a, ECGTilt and Activity1 are the ones to lack energy first. Typically the difference between the lifetime of Activity1 and Activity2 can be analyzed and several parameters can be studied to overcome this situation. Later in this section, we show how two alternative designs for Activity1 can be explored.

Reachability Properties (p_2) A typical and interesting reachability property deals with unexpected deadlocks in the system (expected ones being those where nodes have no more energy). This can reveal real deadlocks in the system, or allow the identification of crucial nodes whose activity is required to keep the system working. Such a situation can be detected using the following reachability formula, computed with no additional cost with respect to state space generation:

$$dead \quad \wedge \bigwedge_{i \in Nodes} (energy(i) > Min_i) \tag{1}$$

Where Min_i corresponds to the minimum energy required by node i to send a message and *dead* is the boolean meaning that the current state of the state space has no successor. On the BAN case study, this property is verified. It was computed with the unconstrained environment and with a configuration providing up to 500 energy units (it took 1 hour 38 minutes and 2.8 Gbytes).

Checking Behavior for Existing Situations (p_3) Such properties usually require causal formulas expressed by means of temporal logic.

For the BAN system, a typical property is to ensure that the system generates neither a false negative (i.e. a heart attack is not detected) nor a false positive (i.e. a heart attack is detected by mistake in the system). To get this equivalence relation, we use the CTL formula 2 to detect the presence of a false negative and the CTL formula 3 to detect the presence of a false positive.

$$AG(occurs_{heart\,attack} \implies AF(detected_{heart\,attack})) \tag{2}$$

$$AG(\neg occurs_{heart\,attack} \implies AF(\neg detected_{heart\,attack})) \tag{3}$$

In this formula the AG and AF operators respectively mean "in all cases" and "in all futures". $occurs_e$ is either true or false for a given environment e. In the BAN case study, a given environment corresponds to a patient behavior which is annotated by the doctors as being sick or healthy. $detected_e$ is a state property. In our case ($e = heart\,attack$) it involves the PDA and corresponds to the detection of low activity (gathered from the activity sensors) and bradycardia detected by ECGTilt.

On the BAN case study, this property is verified (this was computed up to the system with 500 energy units). This was tested for several environments representing different patients. Such a computation is less complex in time and memory than the worst case lifetime analysis since the system is more constrained. Formulas were computed for

Parameter	value in $config_1$	value in $config_2$
sensing frequency	11 TU	20 TU
acquisition time	3 TU	1 TU
acquisition energy	3 EU	4 EU
processing time	1 TU	2 TU
processing energy	4 EU	6 EU
emission time	2 TU	3 TU
emission energy	6 EU	8 EU
reception time	2 TU	3 TU
reception energy	5 EU	7 EU

Fig. 12: Data for the two studied variants in time units (TU) or energy units (EU)

500 initial energy units. Formula 2 was computed in 1 hour 45 minutes and 2.8 Gbyte memory. Formula 3 was computed in 1 hour 28 minutes and 2.7 Gbyte memory.

Comparing Alternative Solutions (p_4) The choice of a given component may have an impact on a WSN lifetime or on some important characteristics of the system. VeriSensor can be useful to compare two possible solutions. To do so, the designer may either change the characteristics of the nodes to be replaced (if only those change) or replace the node by an instance of another node class.

For the BAN case study, we want to evaluate the impact of two configurations on the system lifetime (e.g. when at least one node cannot communicate anymore). These configurations differ with the characteristics of the activity nodes. The first configuration ($config_1$) embeds a node that samples often but performs light computation. The second one ($config_2$) uses a node that performs less samples but more computations.

To evaluate these configurations, we provide two variations of the activity node specification, following the information displayed in Fig. 12. Then, the obtained specification is linked to the "free" unconstrained environment used to evaluate the worst case lifetime of the system. This work leads to the results displayed in Fig. 11b.

6 Conclusion

This paper presented VeriSensor, a domain specific modeling language for wireless sensor networks (WSNs), designed to be used by WSNs experts and offering support for modeling and formal verification. The objective is to evaluate both quantitative results (e.g. estimation of the system's lifetime or average consumption per time unit) as well as qualitative results (e.g. detection of unexpected situations to be avoided).

VeriSensor enables the modeling of a WSN by providing high-level concepts that support the main use cases of WSNs. Thus, specifying WSNs consists in defining the node characteristics, how nodes are deployed and the physical environment in which the system evolves. The physical environment may model all possible situations (the "free" unconstrained environment), thus leading to the evaluation of the WSN in the worst possible condition. It may also model a dedicated scenario for which the WSN behavior has to be verified.

Instantiable Transition Systems (ITS) and time Petri nets are the underlying formal techniques used for verification. They show encouraging scalability capabilities, thus enabling the analysis of reasonable systems with significant parameters.

The main advantage of the overall approach is to make formal specification and verification more accessible to the end-users (i.e. the designers of WSNs).

Even if we focus on the verification aspects, our approach does not exclude simulation. In fact, since VeriSensor has a formal semantic, it is executable and thus, can be simulated. Then, the environment dimension still allows to select one situation where the system has to be plugged in.

References

1. S. Adams, M. Björk, T. F. Melham, and C.-J. H. Seger. Automatic abstraction in symbolic trajectory evaluation. In *Formal Methods in Computer-Aided Design*, pages 127–135. IEEE Computer Society, 2007.
2. B. Akbal-Delibas, P. Boonma, and J. Suzuki. Extensible and precise modeling for wireless sensor networks. In *UNISCON*, pages 551–562, 2009.
3. I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102–114, Aug. 2002.
4. P. Baldwin, S. Kohli, E. A. Lee, X. Liu, Y. Zhao, C. H. Brooks, N. V. Krishnan, S. Neuendorffer, C. Zhong, and R. Zhou. Visualsense: Visual modeling for wireless and sensor network systems. Technical report, U.C. Berkeley, 2005.
5. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33. IEEE Computer Society Press, 1990.
6. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In E. Brinksma and K. Larsen, editors, *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 241–268. Springer Berlin / Heidelberg, 2002.
7. S. C. Ergen, M. Ergen, and T. J. Koo. Lifetime analysis of a sensor network with hybrid automata modelling. In *WSNA*, pages 98–104, 2002.
8. O. Gnawali and M. Welsh. Sensor networks architectures and protocols. In *Emerging Wireless Technologies and the Future Mobile Internet*, pages 125–153. Cambridge University Press, 2011.
9. A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *1st ACM Int. workshop on Wireless sensor networks and applications (WSNA)*, pages 88–97. ACM, 2002.
10. N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.*, 26:70–93, January 2000.
11. L. Mounier, L. Samper, and W. Znaidi. Worst-case lifetime computation of a wireless sensor network by model-checking. In *4th ACM workshop on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks (PE-WASUN)*, pages 1–8. ACM, 2007.
12. P. C. Ölveczky and S. Thorvaldsen. Formal modeling and analysis of the ogdc wireless sensor network algorithm in real-time maude. In *9th Int. conf. on Formal Methods for Open Object-based Distributed Systems (FMOODS)*, pages 122–140. Springer, 2007.
13. C. Otto, A. Milenković, C. Sanders, and E. Jovanov. System architecture of a wireless body area sensor network for ubiquitous health monitoring. *J. Mob. Multimed.*, 1:307–326, January 2005.
14. K. Sohraby, D. Minoli, and T. Znati. *Wireless sensor networks: technology, protocols and applications*. Wiley Interscience, 2007.
15. Y. Thierry-Mieg, B. Bérard, F. Kordon, D. Lime, and O. H. Roux. Compositional Analysis of Discrete Time Petri nets. In *1st workshop on Petri Nets Compositions (CompoNet 2011)*, volume 726, pages 17–31, Newcastle, UK, June 2011. CEUR.

16. Y. Thierry-Mieg, C. Dutheillet, and I. Mounier. Automatic symmetry detection in well-formed nets. In *Proc. of ICATPN 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 82–101. Springer Verlag, June 2003.
17. Y. Thierry-Mieg and L.-M. Hillah. UML behavioral consistency checking using Instantiable Petri nets. *ISSE*, 4(3):293–300, 2008.
18. Y. Thierry-Mieg, D. Poitrenaud, A. Hamez, and F. Kordon. Hierarchical Set Decision Diagrams and Regular Models. In *15th Int. conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 5505 of *LNCS*, pages 1–15. Springer, March 2009.
19. S. Tschirner, L. Xuedong, and W. Yi. Model-based validation of QoS properties of biomedical sensor networks. In *8th ACM Int. conf. on Embedded software (EMSOFT)*, pages 69–78. ACM, 2008.
20. C. Vicente-Chicote, F. Losilla, B. Álvarez, A. Iborra, and P. Sánchez. Applying mde to the development of flexible and reusable wireless sensor networks. *Int. J. Cooperative Inf. Syst.*, 16(3/4):393–412, 2007.
21. H. Wada, P. Boonma, J. Suzuki, and K. Oba. Modeling and executing adaptive sensor network applications with the Matilda UML virtual machine. In *11th IASTED Int. conf. on Software Engineering and Applications (SEA)*, pages 216–225. ACTA Press, 2007.
22. Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Record*, 31, 2002.