Sensors & Transducers Journal, Vol.14-2, March 2012, pp. 18-46

www.sensorsportal.com

Sensors & Transducers

ISSN 1726-5479

http://www.sensorsportal.com

Assessment of Software Modeling Techniques for Wireless Sensor Networks: A Survey

John Khalil Jacoub, Ramiro Liscano, Jeremy S. Bradbury
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
{john.khalil, ramiro.liscano, jeremy.bradbury}@uoit.ca

Received: */Accepted:* */Published:*

Abstract: Wireless Sensor Networks (WSNs) monitor environment phenomena and in some cases react in response to the observed phenomena. The distributed nature of WSNs and the interaction between software and hardware components makes it difficult to correctly design and develop WSN systems. One solution to the WSN design challenges is system modeling. In this paper we present a survey of 9 WSN modeling techniques and show how each technique models different parts of the system such as sensor behavior, sensor data and hardware. Furthermore, we consider how each modeling technique represents the network behavior and network topology. We also consider the available supporting tools for each of the modeling techniques. Based on the survey, we classify the modeling techniques and derive examples of the surveyed modeling techniques by using SensIV system.

Keywords: Wireless Sensor Networks (WSN), Modeling, Node, Software Design, Hardware Design

1. Introduction

A Wireless Sensor Network (WSN) consists of small wireless units called motes, which are attached to specific type of sensors. The sensors measure an environment phenomenon (e.g., humidity, temperature, or soil moisture) and the measured value is expressed as an analog signal generated by the sensors. According to Akyildiz et al. [1], WSN applications can be classified into 2 types; those that have mobile sensors and those that have static sensors. This classification is too broad because within the static WSN category we one can also identify 3 distinct style of static WSNs; Collector

WSNs, aggregation WSNs, and actuation WSNs.

Collector WSNs consist of a number of sensors that gather data and this data is collected at one or more nodes that typically are a gateway to the Internet. These WSNs typically take advantage of tree style of routing. Aggregation WSNs perform some data aggregation and filtering within the network nodes and publish this filtered information for other collector nodes to acquire. Actuator WSNs not only sense the phenomenon, but also react in response to the sensed data. These networks are typically referred to as Wireless Sensor Actuator Networks (WSANs) or at times Wireless Sensor Actor Networks (WSANs) and typically have more stringent real time constraints on the delivery of the data.

WSN systems can be complex and many different challenges can arise during the design of a WSN such as: (i) the distribution of nodes and sensors in a physical environment that may result in lost and delayed data; (ii) the inclusion of real-time behavior within a distributed WSAN; (iii) memory management within the sensor nodes (the small size of the nodes leads to physical limitations that restrict the available memory and therefore memory management is often required;) (iv) operational reliability (WSNs often consist of self-powered nodes in environmentally challenging domains imposing strong reliability requirements;) and (v) improving the network performance, i.e. reduce network delays, packet loss, while increasing throughput (network performance improvements may involve the use of concurrency and event driven communication, which can add additional complexity to the system.)

The design of WSN systems usually occurs at the implementation level and does not involve design at higher levels of abstraction. This leads to a decrease in code portability and to platform-specific implementations [2]. A WSN system produced using this approach is prone to both design and implementation errors and very challenging code debugging (user interfaces to sensor nodes are very limited so even simple text output is challenging). If errors are not detected during the implementation and verification stages of development then they may appear once the system is deployed and is operational. The nodes of an operational WSN application are generally difficult to access once they are deployed in their working locations.

The challenges of developing WSN systems can be mitigated by leveraging higher system level-design and analysis. The use of modeling languages and techniques can drive the design through different abstraction layers and analysis tools can help refine the model. In this paper we survey 9 modeling techniques for WSNs. For each technique we examine how a WSN is modeled at the node and system-level. We also describe a WSN system called Sensor Infrastructure for Viticulture introduce (SensIV) that we use as a case study to assess how each modeling approach would be used to develop the SensIV system. SensIV is considered a collector WSN that was designed and built by our research team to monitor the temperature in a vineyard field. The deployed nodes are static there is no direct action taken in response to the sensed data.

The rest of the paper is organized into 8 sections. Section 2 describes the SensIV case study. Section 3 gives an overview of the modeling techniques reviewed in this survey. Section 4 discusses the modeling of WSN elements including sensors, nodes and hardware. Section 5 presents WSN modeling at the system level. Section 6 discusses the supporting tools and the importance of each tool for WSN design. Section 7 presents related work (i.e other sensor modeling surveys.) Finally section 8 presents the conclusion and future work.

2. Case Study

In this section we present SensIV [3] project as a case study for the modeling techniques surveyed. Using a case study helps in understanding how the modeling approaches presented in this paper could be applied to help develop and analyze the software for a system that we are fairly familiar with.

2.1 System Overview

The aim of the SensIV project is to monitor the temperature of a vineyard field. For this survey understanding the vineyard application is not crucial and therefore SensIV can be considered as simply a collector WSN consisting of a set of sensor nodes that collect temperature data across a field and transmit this data to a collection point using a wireless sensor network. Each sensor node supports 4 temperature sensors that measure the gradient in the vertical direction at one spot in the field. The motes communicate with each other wirelessly until the data reaches the gateway where the temperature values are stored in a database for further analysis.

2.2 Sensor Physical Layer

Each node consists of the following hardware:

- Four temperature sensors: The sensors used are thermal sensors of type LM135 [4]. The sensor accuracy is +/- 1^{o}C and can measure a temperature range in between -55^{o}C to 150^{o}C.
- An MDA300CA acquisition board: The MDA300CA data acquisition board from Crossbow has expansion connectors for 7 single-ended and 4 differential ADC channels.
- A Wireless Iris mode: The Iris wireless module uses the Atmel processor with 128 KB program memory and 8KB RAM. In terms of radio communication, it uses 2.4GHz (IEEE802.15.4) with a range of 500 meters. This type of mote also supports a low-power mode of operation for the micro-processor, radio, and logger.
- A solar cell is used to recharge the 2 AA alkaline batteries that power the sensor node. The system was designed for continuous operation with sunlight conditions encountered in Southern Ontario.

2.3 Routing Protocol

The protocol used for routing a data packet to the collector is the Collector Tree Protocol (CTP) [5]. CTP is an address free protocol that maintains a tree routing topology among the sensor network to the collector. The version of CTP used is the one available in the TinyOS deployment that is fairly reliable. The parent of a node is chosen based on the link estimation value that is continuously updated as data is being sent through the network.

2.4 Sensor Software

The sensor software leverages the TinyOS 2.1 [6] operating system which provides many useful services for sensor networks some of these which are: a task scheduler, boot sequence, timers, memory storage, serial and radio communication, radio management, and multi-hop routing protocols.

There are 2 distinct sensor software packages. One package is for the collector and another package is for the sensor nodes in the field. For the study case we will focus primarily on the software for that resides on the sensor nodes on the field. We also have designed a software component on sensor node that captures “control messages” sent from the collector that are transmitted using a dissemination protocol that is also provided by the TinyOS community. For the purpose of the case study we focus only on the software modules that collect data and transmit this data to the collector and do not present any logic or models that relate to the capture of the “control messages”.

In TinyOS there is a task execution process coordinated by a time scheduler. If there is no task in the task queue, the timer scheduler will put the processor into sleep mode. The processor is woken up by either a message arriving signal from the dissemination protocol or the timer firing event used to

collect data (the system was designed for a 5 minute data acquisition period.)

2.5 Location of the Sensor Nodes

The sensor nodes were deployed in a field at the University Of Ontario Institute Of Technology in Oshawa, Canada. We have captured the coordinates of the nodes and these are shown in a Google map image in Fig. 1. The approximate average distance between the sensor nodes is about 32 m. The terrain slopes downwards from nodes 4, 9, 2, 8, 7, and 3 towards 5, 1, 6, and 10. The collector node (marked as Base) is located about 4 m of the ground in a 2nd floor office of a building. From a system modeling perspective the location of the nodes is important for the performance analysis of the sensor network system.

Fig. 1: Nodes Physical Location

3. Overview of the Software Modeling Techniques for Sensor Networks

In this section, we provide an overview of each of the software modeling techniques for sensor networks included in our survey. Some of those modeling techniques have been used in the application development process, while others take WSNs as a case study for their modeling approach. Some use standard software modeling notations, such as UML, while others use their own custom notation. We also attempt for each approach to describe how the SensIV system would be modeled using that technique.

The techniques use different basic elements, such as channels, processes, modules, and components, to express a WSN as a model. A channel is used to represent the communication between two elements of a WSN. For example, channels can represent the characteristics of sensor-node communication, node-node communication, and node-gateway communication. Processes, modules, and components are used to represent the sensors and nodes of a WSN. We will describe details of the modeling elements later when we discuss the individual modeling techniques.
The modeling techniques surveyed also vary in terms of the scope of modeling. For example, some techniques are intended to model a single node or communication between a pair of nodes while others are intended to model the entire WSN.

3.1 HL-SDL

HL-SDL [7] is a modeling language that uses the Specification and Description Language (SDL) [8], which is normally used to model and simulate communication protocols. SDL has been adapted by Dietterle, et al., to model TinyOS components using SDL processes (i.e., extended finite state machines). The system is modeled as a collection of channels and processes. The model can be used to

generate nesC source code, which is commonly used in WSNs based on the TinyOS environment. While generating nesC code, each process (which is the smallest unit of the model) represents a component in TinyOS In their work, Dietterle et al. used manual optimization to enhance the generate code

Their approach proposes that a TinyOS component be considered as the minimal entity to encapsulate the behavior of an SDL process by implementing the process state machine. Communication between SDL processes is via asynchronous signals that could be mapped to either a TinyOS command or event handler. They also propose that the process state machine logic can be captured by using one TinyOS task that services the incoming commands and events, performs some state changes in the component, and calls other components in the system. The model can be used to generate nesC source code, which is commonly used in WSNs based on the TinyOS environment. In their work, Dietterle et al. used manual optimization to enhance the generated code.

As related to the SensIV case study we designed the SensIV software using one component named WhiteRabbitC that services 21 events and launches 12 tasks as shown in the class diagram in Fig. 2 (Tasks are shown with the symbol *T* in front of them and the event listeners are marked with the letter *E*.)

This implies that to be able to leverage the SDL modeling language we should be considering the use of several components with very simple behaviors that can be captured with the use of 1 task. This is not something that comes naturally in the design of the software. After reflecting on our design it is possible to consider components that contain only 1 task but this significantly breaks up the code into many small components each having a very specific role. For example, SensIV uses 4 tasks to read the sensory data. It does this because it takes some time to read the data and the read operations are asynchronous. This results in very small and detailed state machines since each component requires

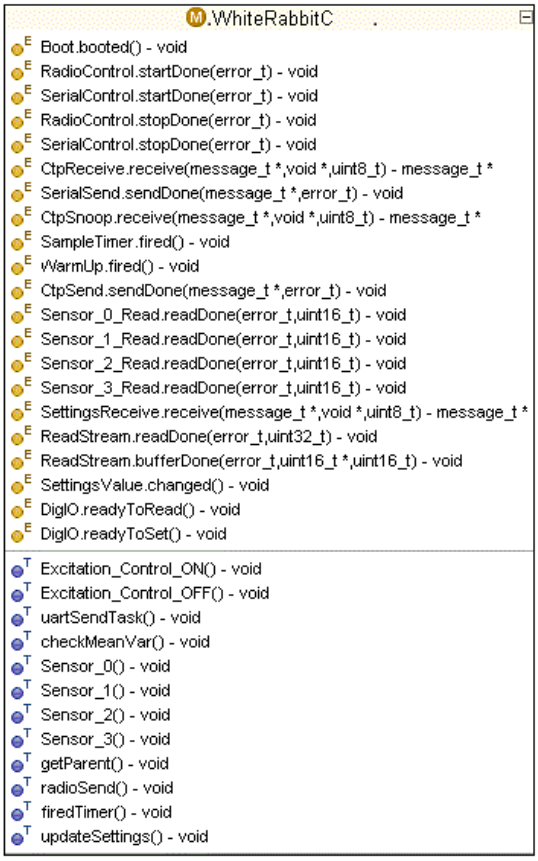

Fig. 2 : SensIV Software Component

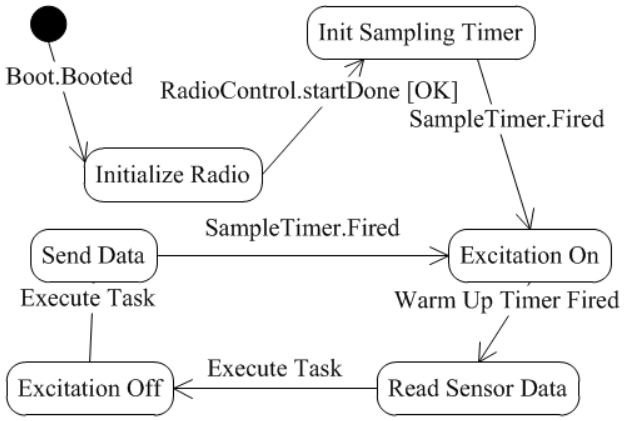

Fig. 3 SensIV State Diagram

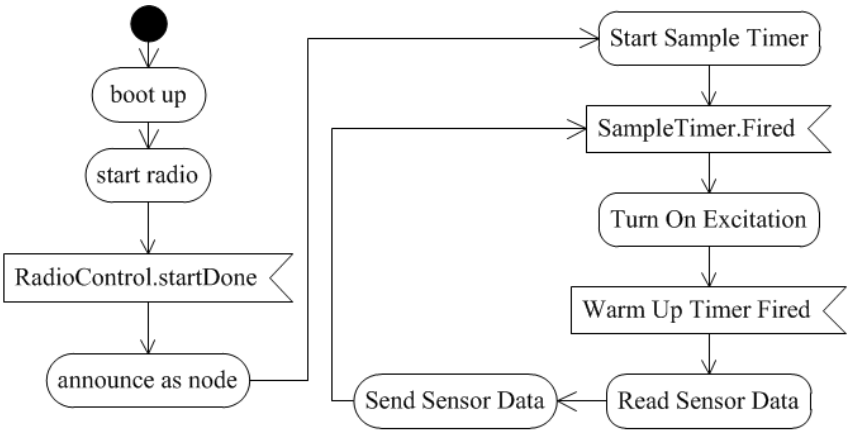

Fig. 4 : SensIV Activity Diagram

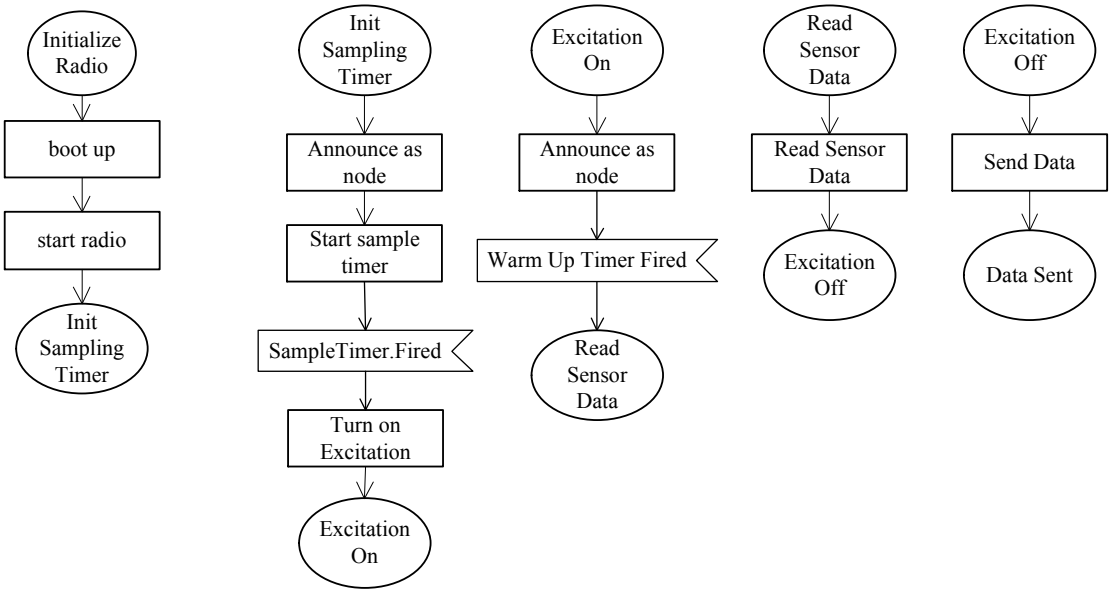

Fig. 5 : SensIV SDL Process Diagram

one as opposed to a single state machine in SensIV that captures the data acquisition cycle and depicted in Fig. 3.

We can follow the process presented in HL-SDL to create an SDL process diagram for the SensIV software but one has to keep in mind that the SDL analysis of this result would not be helpful since several tasks are being used to achieve this. An SDL process is built from the combination of a state and activity diagram. Fig. 3 and Fig. 4show the state and activity diagram of SensIV software related to the data acquisition logic.

An SDL process diagram can be created from the combination of the state and activity diagram as depicted in Fig. 5.

3.2 Insense

Dearle et al. [9] use the Insense modeling language to create a component-based model for a WSN. Components in Insense are concurrent and they communicate synchronously via directional channels that are used to abstract away from low-level synchronization and communication issues. Insense is built in the Contiki operating system, which similar to TinyOS is a popular operating system used for WSNs. The Insense model has a translator that produces C source code that can be used to calculate important details such as worst case execution time.

Insense Model Elements

Insense captures the key system elements of the WSN as components. The components can represent software or hardware entities of a WSN. The components capsulate a specific behavior of an element in the WSN system and has interfaces to interact with other components of the system. Each component has no dependences on the other components. However, a component can initiate the activity of another component. The component stops execution either by an external signal from the other component or by itself. Each component definition contains four main parts:

- The channel part, which contains the input and output channels of each component and other components which interact with the component
- The component variables
- The component constructor
- The behavior part, which captures the component behavior.

The component starts to execute the behavior once the other component creates the instance of it. The flow of the data is declared by using the command send, receive, input, and output as shown in Fig. 8. In order to bind the components with each other, the channels are declared in the design as well. For example, *connect sensor.output to sensor reader.input* where reader and sensor are components and input and output are the ports. Based on that, the communication between the components is synchronized. The language supports deterministic and non-deterministic selection of the nodes by using logic statement, such as If and select statements. Insense provides components in order to model the hardware such as temperature sensor or humidity sensor.

Insense Model for SensIV

Insense models SensIV by using components which represents the hardware and the software components of the one node. In this section, we only explain some components of the SensIV system due to the size limitation of this paper (see Fig. 6).

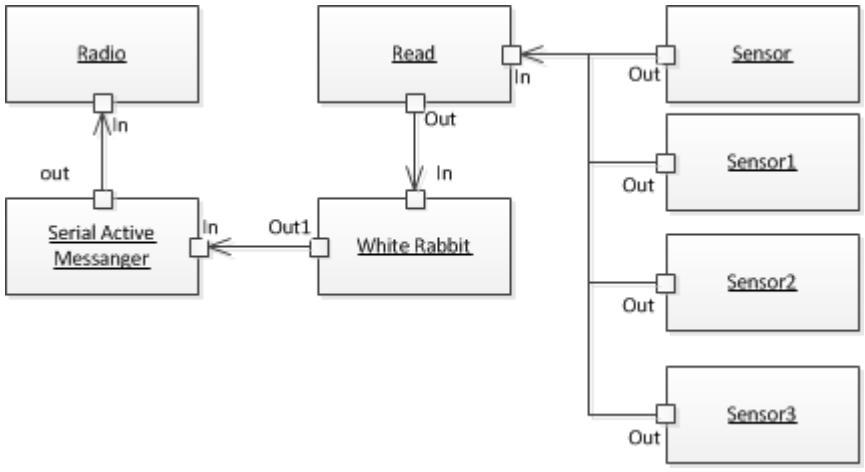

Fig. 6: Captured SensIV Components by Insense

The model captures the interface between the captured nodes (see Fig. **7**) and the behavior of the following components:

- **WhiteRabbit component:** requests the data from the sensors, receives the four values, and generates one data packet (see Fig. 8).
- **Read component**: triggers the sensors to read, receives the values, and send the values to WhiteRabbit (see Fig. 9).
- **Sensor component:** represents a hardware component and captures the behavior of the thermal sensor (see Fig. 10).
- **Serial Active Messenger component:** sends the data packet to the radio and invokes the radio to send them over (see Fig. 11).
- **Radio component**: sends the data through the wireless signal (see Fig. 12).

```
type SensorRD  is interface( out float out )
type SensorRD1  is interface( out float out )
type SensorRD2  is interface( out float out )
type SensorRD3  is interface( out float out )
type Read_Interface is interface( in float  input; out float output)
type White_Interface is interface( in integer input; out vector output )
type Serial_Interface is interface( in vector input; out vector output )
type Radio_Interface is interface( in vector input)
```

Fig. 7: Declaration of Component Interfaces

```
component WhiteRabbit presents White_Interface {
    size=4
    index=0

    constructor() { … }

    behaviour {
    Read= new read()
    receive next from input
    store[index] := next
        index := index + 1
        if( index >= size ) {
            SerialActiveMessanger = new SerialActiveMessanger()
                    connects White_Rabbit.ouput to
SerialActiveMessanger.input
            send store on output
            index := 0
        }
    }
}
```

Fig. 8 : WhiteRabbit Component Model

```
component Read presents Read_Interface {

    constructor() {
        sensor = new sensor ()
        sensor1 = new sensor1 ()
        sensor2 = new sensor2 ()
        sensor3 = new sensor3 ()
    }

    behaviour {
        receive Temp from input
        connects Read.ouput to WhiteRabbit.input
        send Temp on output
        index := 0
    }
}
```

Fig. 9 : Read Component Model

```
component Sensor presents SensorRD {
    constructor() {…}

    behaviour {
        connects sensor0.ouput to Read_Interface.input
        send Temp on output
    }
}
```

Fig. 10 : Sensor Component Model

```
component SerialActiveMessanger presents Serial_Interface {

    constructor() {…}

    behaviour {
        receive store from input
        Radio= new Radio()
        connects SerialActiveMessanger.ouput to Radio.input
        send store on output
    }
}
```

Fig. 11 : SerialActiveMessanger Component Model

```
component Radio presents Radio_Interface {

    constructor() {…}

    behaviour {
        receive store from input
    }
}
```

Fig. 12 : Radio Component Model

We would like to point out that WhiteRabbit develops other activities. However, we include the activity of sensing and sending the data only due to the size limitation of this paper.

3.3 Mathworks

The framework in [10] aims to design, simulate, and generate the code for WSNs. The node behavior is modeled as a parameterized Stateflow block. Nodes in the Mathworks approach also contain timing and random number generators that are used for simulation. Additionally, the communication medium, which is used to define the connectivity between the nodes, is represented at a lower abstraction level and is implemented in the C language. By leveraging Mathworks tools, such as animated state charts, chart displays, scopes, and plots, analysis of the WSNs can be performed. According to the results, the model can be refined. The final stage is to generate the WSN code using the Target Language Complier (TLC) which can generate C code for MANTIS and nesC code for TinyOS. The Mathworks approach has been used successfully to generate the code for Energy Efficient and Reliable In-Network Aggregation (EERINA) algorithm for clustered sensor networks [11].

Design representation

The authors of Mathworks have designed a sensor node block and a communication medium block. The sensor node block contains a timer generator and a parameterized state flow which implements the algorithm deployed inside each node. The sensor node library has been implemented separately and each node contains an instance of the library. Therefore, all nodes run the same algorithm. However, changing the algorithm can be done by creating a new library and then creating an instance of the new library.

The communication medium block was implemented on C language based S-Function [12] which is

used to specify the connectivity of each node and implement the communication medium logic. The data packets are inputs and outputs of the communication block. The data packets are processed first by the communication medium block then it is fed to the appropriate output.

System Analysis

Mathworks takes advantage of the analysis tools, such as the animated state charts, scopes, and displays, to perform functional analysis of the algorithm. The design is enhanced based on the analysis results. However, the authors did not present any results from the analysis or an example of the enhancements that could be done on the model based on the analysis results.

Code Generation

The embedded coder is used in order to generate ANSI C code which represents the state charts of the nodes. The last stage is the Target Language Complier (TLC) which generates the code for the deployment. TLC generates 2 types of code: nesC code for TinyOS and C code for MANTIS [13] (MANTIS is another operating system for WSNs).

TLC contains a group of mapping rules which controls the code transformation process from the ANSI C code to the target code. In addition, TLC adds the information and details to the code which are required by the target platform in order to excute the code. The authors comment that generating the C code for MANTIS was easier than generating nesC for TinyOS because of the similarity between C and ANSI C code. The generated code did not need any kind of modification. On the other hand, the generated code size is bigger than the manually implemented code.

Mathwork Model for SensIV

As mentioned above, Mathworks have used state charts and activity diagram to capture the system behavior. SensIV state chart and activity diagrams are shown in Fig. 3 and Fig. 4 respectively. From communication medium prospective, SensIV nodes are deployed within the range of 500 m of each other. Therefore, potentially all nodes can communicate with each other. Additionally, CTP chooses next hop based on the link estimation. Therefore we have no expectation of the connectivity of the nodes connectivity since all nodes can communicate with each.

3.4 Model Driven Engineering Approach (MDEA)

Losilla et al. [2] use UML and a Model Driven Engineering (MDE) approach that includes three the 3 modeling layers shown in Table 2. The research team was working in parallel on Model-to-Model (M2M) Transformations and Model-to-Text (M2T) Transformations. In M2M transformations the model transformation takes place between the WSN-DSL layer to the PIM-UML layer and then from the PIM-UML layer to nesC. In the M2T transformation the nesC code is generated from the nesC meta-model.

Transformation rules control transforming from one modeling layer to another. Moreover, refinement can occur after every transformation to improve the generated model. The MDEA approach is supported by the Eclipse IDE as well as a number of Eclipse plug-ins (e.g., MOFScript) that are responsible for automating the transformation process.

Table 1: MDEA Layers

Layer	Information Captured	Example	Annotations
Domain Specific Language	Functional and non-functional requirements	Nodes functionality, data stores locations, physical distributions, communication mechanism, and communication rate	Class diagrams
UML PIM	Data flow	System states and Flow of the algorithm	Activity diagrams and state machine diagrams
nesC Meta Model	Software components	Radio methods, LEDs, and timer methods	Component diagrams

Table 2 : SensIV WSN-DSL Layer

Item	Description	SensIV
Node groups	All nodes with the same behavior	Group 1: The sensors (10 nodes) Group 2: The base station (1 node)
Functional Units	Information about the function of each group	Group 1: Sensing Group 2: Network monitoring and discrimination commands
Functional Units Types	Methods and components of each group	mainC, LedsC ,TimerMiliC, WarmUP WatchDog, DemoSensor0, DemoSensor1 DemoSensor2, DemoSensor3, Radio(ActiveMessageC), DisseminatorC ActiveMessageC, SerialActiveMessage SerialAMSenderC, UARTMessagePoolP(PoolC) UARTQueuePC(QueueC), Batter
Resources	Sensors and ports	Thermal sensors, ethernet ports, radio unit, and solar cell

WSN-DSL Model

WSN-DSL model captures the functional and non-functional requirements of the system. The information captured by this layer along with the corresponding SensIV entities is shown in Table 2. The SensIV class diagram is shown if Fig. 14 as derived by leveraging the DSL meta-model defined in MDEA for sensor networks (reproduced in Fig. 13).

UML-PIM Model

The UML-PIM (Platform) model has been created by using UML 2.0. The aim of developing this stage is to cover the semantic gap between the WSN-DSL model and the nesC meta-model. The UML-PIM model explains the control and the data flow of the system through using the activity diagrams and using state machine diagrams. An example of the transformation rules between WSN-DSL and UML-PIM is each node group is mapped to a component. The state diagram and the activity diagram for SensIV can be used as the UML-PIM model and have been presented in Fig. 3 and Fig. 4 respectively.

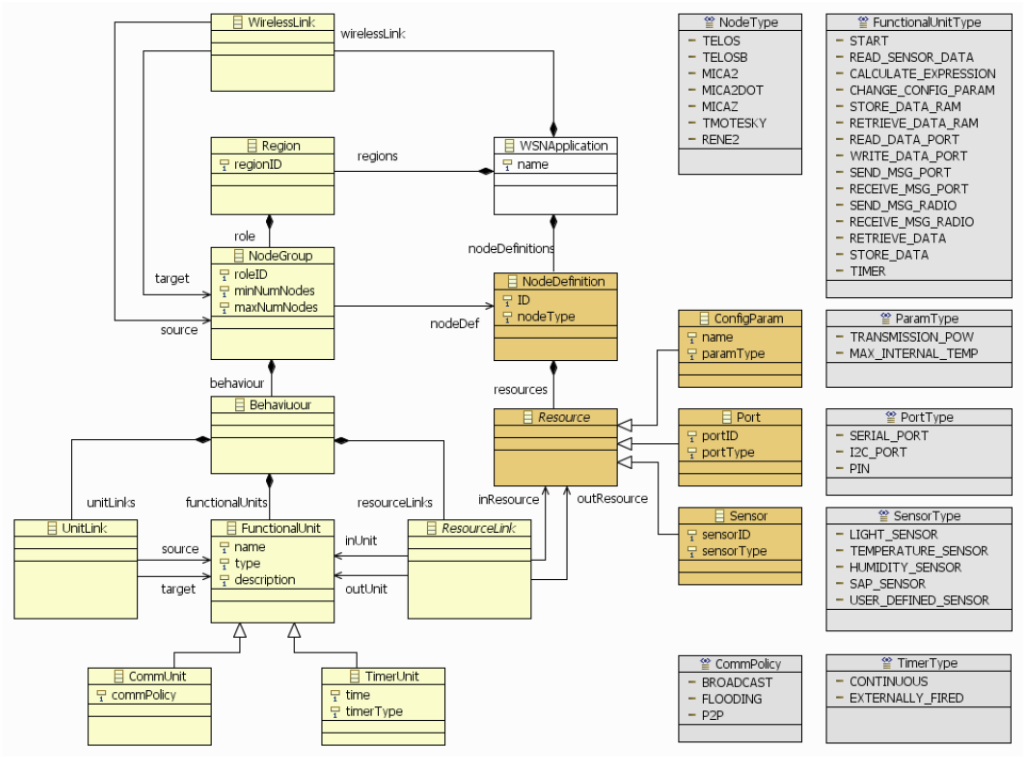

Fig. 13 : WSN-DSL Meta-Model, reproduced from [2]

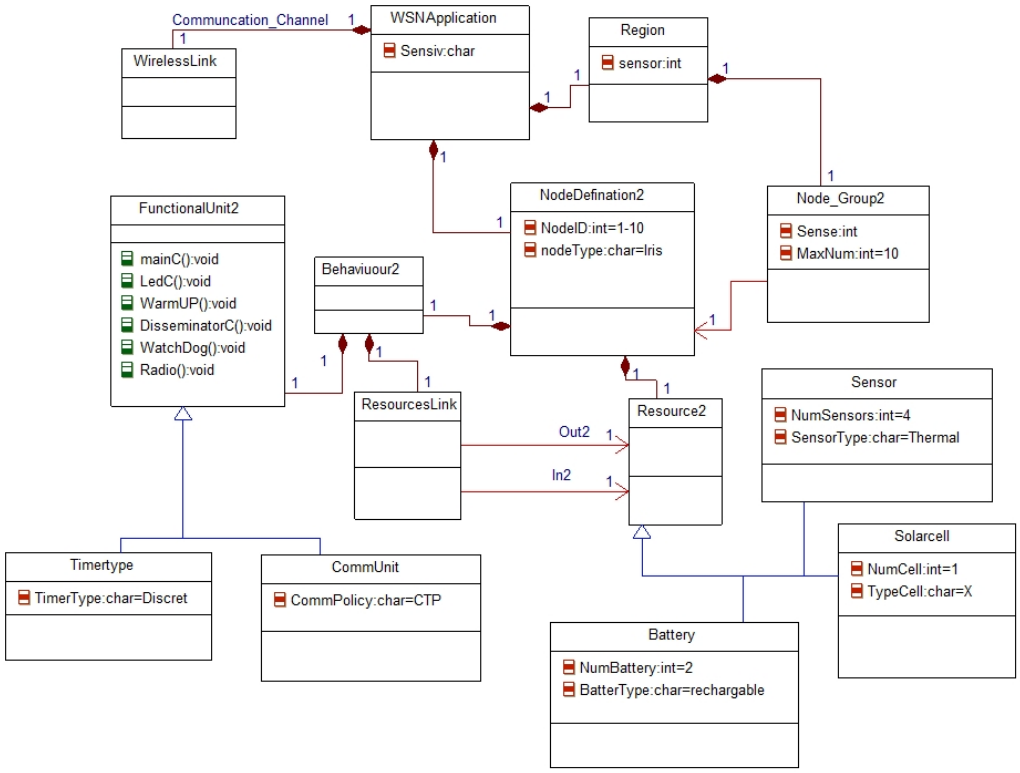

Fig. 14 : SensIV Node Class Diagram based on the WSN-DSL metal Model

nesC Meta-model

The nesC meta-model represents the components of the nesC code and their interconnections. It was used by MDE to generate the nesC code that is used by TinyOS systems. The transformation is done through the M2T Eclipse plug-in MOFScript. The tool offers other facilities such as model checking,

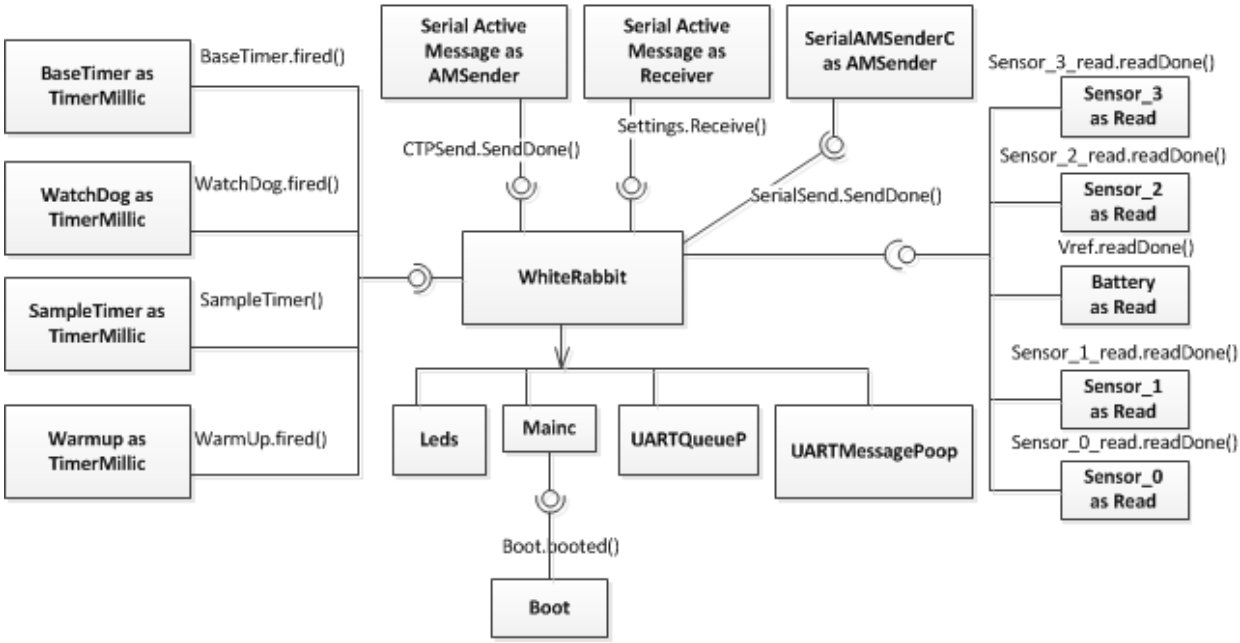

Fig. 15: nesC Component Model for SensIV

parsing, and querying. The transformation process is controlled by a group of rules as well. For example, one of the rules states that for each interface in the nesC meta-model, all of its commands must be implemented. The nesC component model for SensIV is shown in Fig. 15 based on what was presented as the nesC meta-model in MDE.

3.5 Promela Model

The Promela Model [14] was created in order to check a WSN system which has 3 main characteristics: the nodes are in a dynamic movement state, the communication link is bi-directional and is unreliable, and the routing protocol ensures that each node is aware of the neighbor nodes. The model is created in the Promela modeling language, the input language for the model checker Spin. The Spin model checker has been used in the verification of both hardware and software systems. The WSN Promela Model verifies the following correctness properties in Spin:

- Network sensors are **always** connected.
- There is at **least one** communication way for each sensor.
- **Eventually** each sensor will be connected to the rest of the network.

Promela Model Elements

The Promela Model captures the following details of a distributed WSN system:

- The model captures each node behavior as a process. Therefore, there is availability to dynamically create/remove a process from the model based on the dynamic behavior of the nodes. For example, a new node joins the network.

- The nodes update their physical location to a central unit called the Location Manager (LM). The LM keeps data about each node such as the ID, the physical location, the movement location, and the distance between each node.

- The model contains a model for the communication channel, which captures the interaction between each node and the environment. An example for such interaction is when a node receives a notification in order to capture the current temperature. Also, when the node notifies the neighbor node when the temperature exceeds a specific level.

- The internal computation of each sensor is abstracted out since the main focus is the correctness of the nodes interaction with each other.

- The communication channels are dynamically terminated and created based on the nodes physical location and the distance between each node and the neighbor node.

Promela Model and SensIV

The Promela Model approach is intended for a different type of sensor network application than SensIV. There are several reasons why the Promela Model is not appropriate for SensIV:

- The SensIV nodes are in a static state and not dynamic. Therefore, none of the correctness properties are valid for SensIV.
- The nodes do not notify the outer environment when the temperature exceeds a specific value.

Although the Promela Model approach [14] does not suite our case study example, SensIV, we believe that in general model checking techniques, such as Spin, can be used to address the modeling and analysis of SensIV provided that a new model was created.

3.6 SensorML

SensorML [15] is an XML based language that is used to model the sensor specifications such as hardware, physical location, and internal process. The SensorML is designed for the following purposes:

- Support the geo-location of sensors in order to search online for WSNs.
- Capture sensor information, such as the manufacturer, sensor types, and physical locations
- Capturing the flow of data between components in sensor system.

In SensorML, the system components are captured as processes. The components modeled can be physical and non-physical components. Physical components include detectors, actuators, information about the node location, and interfaces while non-physical components include the mathematical equations used to transform raw transducer data to calibrated sensor values.

SensorML Elements

A SensorML specification of a sensor system is composed of the following SensorML elements.

- The **process model** is defined as an operation which takes one or more inputs and according to the data flow generates one or more outputs based on some adjustable parameters.
- The **process chain** block consists of interconnected linked processes. The process chain has its own inputs and outputs and it can participate with other process chains. The process chains are used to model composite models which do not exist in the physical domain. The chain contains the description of the connection between the components it has within the chain.
- A **component** is defined as the basic unit of modeling. A component can be part of the system

or can be part of the process chain.

- A **system** contains several physical and non-physical processes that act together to form the system output. A system provides a list of processes, the links between the processes, the components, and connections properties. In addition, the system can provide relative positions of the components through the positions property.

SensorML Model for SensIV

This section contains a part of the SensorML file for one of SensIV nodes (see Fig. 16 and Fig. 17). The parts shown here are as follows:

- The node manufacture: Crossbow.
- The mote platform: Iris.
- The sensor type: Temperature (thermal) sensors.

```
<sml:SensorML>
    <sml:identification>
        <sml:IdentifierList>
            <sml:identifier name="Mote Type">
                <sml:Term definition="urn:ogc:def:identifier:moteType">
                    <sml:value>Iris</sml:value>
                </sml:Term>
            </sml:identifier>
            …
        </sml:IdentifierList>
    </sml:identification>
    ...
    <sml:inputs>
        <sml:InputList>
            <sml:input name="Temp1">
                <swe:Quantity>
                    <swe:uom code="" xlink:href="degreeC" />
                </swe:Quantity>
            </sml:input>
        </sml:InputList>
    </sml:inputs>
    …
    <sml:member>
        <sml:System>
            <sml:classification>
                <sml:ClassifierList>
                    <sml:classifier name="sensorType">
                        <sml:Term definition="urn:ogc:def:identifier:sensorType">
                            <sml:value>Temperature</sml:value>
                        </sml:Term>
                    </sml:classifier>
                </sml:ClassifierList>
            </sml:classification>
            <sml:components>
                <sml:ComponentList>
                    <sml:component name="Temperature">
                        …
                    </sml:component>
                    …
                </sml:ComponentList>
            </sml:components>
            …
```

Fig. 16 : SensorML Model for SensIV (Part A: Components)

```
            …
            <sml:connections>
                <sml:ConnectionList>
                    <sml:connection>
                        <sml:Link>
                            <sml:source ref="Temperature\Temp1" />
                            <sml:destination ref="Temperature\Temp1" />
                        </sml:Link>
                    </sml:connection>
                    …
                </sml:ConnectionList>
            </sml:connections>
        </sml:System>
    </sml:member>
</sml:SensorML>
```

Fig. 17: SensorML Model for SensIV (Part B: Connections)

3.7 SystemC-AMS

SystemC-AMS [16] is an open source C++ based language. SystemC-AMS captures the system behavior through the use of modules. The system behavior is modeled by using a data flow diagram. The data flow takes place based on a set time steps. SystemC-AMS is suited for communication systems which contain sampling components and communication channels for WSNs. The modeling technique represents each node by using a module. The module contains a model for the sensor, the A/D converter, the microcontroller, the RF, and the transceiver. The analysis developed by SystemC-AMS focuses on the SNR and BER of the ADC process and the communication channel, respectively. Therefore, the model tends to capture the electronic design of the system.

We would like to point out that parameters captured in SensIV model, are taken from the datasheet of the electronic devices in our design. Designing SensIV did not involve the design of the electronic structure of the components. Our designing process involved the following:

- Developing the software deployed in the sensors.
- Assembling the hardware components together, such as the acquisition board, the thermal sensors, and the wireless motes.
- Deployment of the nodes in the field.

We explain the SystemC-AMS electronic model for SensIV in order to help the reader to have a good understanding of using SensIV as a modeling technique for WSNs.

SensIV Thermal Sensors Model

The sensor's basic task is to transfer the analog signal, which represents the temperature, to an analog voltage. The generated voltage of the temperature is loaded on the resistor. The value of resistor R = 10KΩ and value of f = 2.4GHz (see Fig. 18). The output voltage is proportional to the temperature.

A/D Converter

The analog to digital converter converts the signal received from the sensor to a digital form. SensIV A/D converter is captured by second order FIR filter. The sampling rate for T = 0.005sec. (Micaz Datasheet) and number of bits is 16bit.

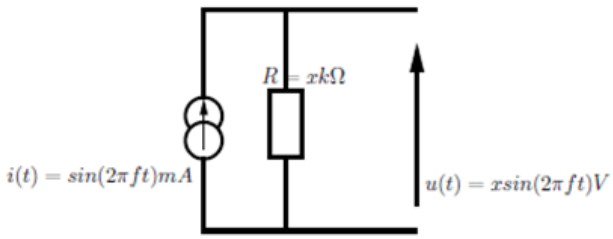

Fig. 18 : Electronic Model of the Thermal Sensors [16]

Microprocessor

The microprocessor used in SensIV nodes is ATMEGA128 [17]. The microprocessor is a 16bit wide instruction processor and has 128Kbyte flash memory. SystemC-AMS uses a C language complied binary file to capture the behavior of the processor.

Transceiver

The RF transceiver uses Quadrature Phase Shift Keying transmission with carrier frequency 2.4GHZ and the data frequency 2.4MHZ frequency model. Noises of the communication channel are introduced to the model in order to calculate the BER communication channel.

3.8 UM-RTCOM Model

UM-RTCOM [18] is a real-time component based modeling framework written in CORBA that is better suited for the modeling of Actuator WSNs. It is composed of sensors, actors, and a coordinator (the coordinator is located in the base station). Actors gather data from groups of sensors based on their physical locality and respond to a phenomenon identified by the coordinator. Sensors communicate with actors and actors communicate with the coordinator using channels. Communication via a channel is modeled as a tuple. "A tuple is a sequence of fields with the form: (t_1, t_2, ..., t_n) where each field t_i can be: a TC identifier (or) a value of any established data type of the host language where the model is integrated" [9]. A UM-RTCOM model can also be used for several kinds of analysis, such as Worst Case Execution time (WCET), deadlock freedom, and verification of liveness properties. The communication channel protocol modeled in UM-RTCOM has been tested in an actual sensor network deployment by Barbaran et al. [19]. The deployment shows the improvement of the middleware overhead compared to another deployment where the motes send the sensed data periodically to the actors.

System components

UM-RTCOM uses components to capture the system behavior. The model declares for each component interfaces, the services which are offered by each component, and the connectivity of each component. The components can invoke each other based on event triggers, time triggers, or service requests. The 3 types of components in the UM-RTCOM model are generic components, active components, and passive components. These are defined as follows:

- **Generic components:** A generic component is considered as a container for the other types of components.
- **Active components:** An active component represents the data flow inside the components. Also, it represents components which are responsible for handling the data flow inside the node. Furthermore, active components have the ability to invoke other components.

- **Passive components:** A passive component represents the shared resources. They cannot invoke any action or any other components. However, they offer services to other components.

Virtual Machines

UM-RTCOM models the network element nodes and the base Station by using the virtual machine. The virtual machine communicates with each other by using the tuple channel (TC). All the data exchanged between the nodes and the base station in done through tuples. TC gives the capability of one-to-many and many-to-one communication. All the network elements have an access to the communication channel and the elements are capable to see which tuple has been consumed by which node or base station. The network elements manipulate the channel tuple based on the tuple attributes priority, deadline and remove.

Nodes Locations

UM-RTCOM takes into consideration the physical location of the nodes in order to facilitate the coordination between nodes, avoiding redundant data, and to determine suitable reaction in case the reaction ability is available.

UM-RTCOM is capable of capturing the physical location of the nodes. The location of the node is declared as one of the communication channel parameters. For instance, the communication channel code between the base station and the nodes 2D location is [com_type = one_to_many, location = S_E] where S_E referes to south east.

UM-RTCOM Model for SensIV

UM-RTCOM models SensIV nodes by using one virtual machine and the base station by using another virtual machine. The nodes send the sense information to the base station in the forms of tuples and the communication type will be many-to-one since the data is collected from many nodes to the base station. The base station sends dissemination commands to change the sampling rate. In this case, the communication type will be many-to-one. The tuple which carries the dissemination commands has a higher priority than the tuple which carries the sensed data, such that the TC transfers the tuple to the other nodes. The sensor nodes are represented by using the generic components which contains 2 active components (the CPU and the timers). The radio units and the temperature sensors are represented by using the passive components.

3.9 eXtended Reactive Modules (XRM)

XRM [20] is an extension language of Reactive Modules (RMs). Demaille et al. used WSNs as a case study to evaluate the XRM modeling language. The case study successfully used XRM to model multiple nodes as modules and was able to support several network issues such as communication capability, memory, and energy consumption. Model checking of XRM is possible via a transformation to the original RM language. RM modules can be used with the model checking tools PRISM. PRISM overcomes the problem of state space explosion phenomenon of the traditional model checking techniques. The new language deals with different WSN designing issues such as WSN scalability, nodes locations, power consumption, and package delivery probability. XRM captures each node in the network by using a module. The modules contain variables and methods which represents the node behavior.

WSN Scalability

XRM deals with scalability by generating the modules and using the *gen* function. XRM generates modules according to the number of nodes in the network.

Node Locations

The node physical location is specified using the variables X and Y at each module. Physical locations of the nodes are represented by using a rectangular grid. Each node module contains a declaration of each of the neighbor nodes to which it transmits and receives.

Package Delivery Probability

Each node can be on one of the following four states:

- **Sensing:** The node switches to the state sensing every 5 minutes.
- **Broadcasting:** The node sending a data package to the neighbor's nodes which are declared in the module.
- **Listening:** The node goes into the listening state after it broadcasts the information.
- **Sleep:** The nodes switches to sleep state for the majority of time in order to save energy.

The data packages are lost when a node transfers the data while the neighbor is at sleeping state. Based on this scenario, XRM calculates the package delivery probability for the network.

Power Consumptions

The battery usage of each node is modeled as a local variable of the module. The node activities, such as transmitting, receiving, or sensing, consume some therefore, the value stored in the battery variable is reduced according to the activity. The value of zero represents the death of the node. The modules contain a declaration of the consumed power of each activity. Once the node is down because of a dead battery, the module is not considered in the simulation anymore. Every time the node switches from one state to another, the equivalent part of the energy is reduced.

XRM Model for SensIV

The XRM model contains 10 modules to represent the 10 nodes. Fig. 19 shows an example for node 6. The power consumption values were measured from a real-time test for the nodes and were published in [21]. We have documented the physical location for each of the nodes in the SensIV system. The nodes switch to the sensing mode every 5 minutes.

```
// Grid (even) dimensions.
const int X = 18, Y = 35;
// Initial energy, percentage of lost cells.
const int POWER = 2000
// States.
const int OFF = 0, SLEEP = 1, SENSE = 2,
LISTEN = 3, BROADCAST = 4;
// Energy consumption for each state.
const int COST_SLEEP = 0.3, COST_SENSE = 13,
COST_LISTEN= 8, COST_BROADCAST = 12;
```

Fig. 19 : XRM Model for Node 6

Table 3 : Modeling of WSN Elements

Approach	Node Behaviour	Sensor & Hardware Modeling
HL-SDL [7]	Concurrency, event-driven	-
Insense [9]	Concurrency, real-time	Sensor type
Mathworks [10]	Procedural, state space	-
MDEA [2]	Procedural, state space	Times, ports, wireless channel
PM [14]	-	-
SensorML	Event-driven	Sensor types
SystemC-AMS [16]	procedural	ADC, microprocessor, wireless channel
UM-RTCOM [18]	Concurrency, real-time, event-driven	-
XRM [20]	procedural, state space	-

4. Modeling At the Node and Sensor Level

In this section, we consider how the different modeling techniques represent WSN elements, including nodes, sensors, and hardware. In particular, we consider the modeling of node/sensor behavior, sensor data, and hardware components (see
Table **3**). The node behavior column tries to capture which particular characteristics that an approach focused on modeling. The sensor and hardware modeling column considers if the actual WSN hardware, such as the ADC, microprocessor, and the wireless channel are included in a model. Hardware modeling also considers the types of sensors that a modeling technique can represent.

4.1 Node Behavior

Most of the modeling techniques use a form of component-based modeling to represent a sensor node. The WSN behavior is modeled by specifying the component's internal behavior, component to component interactions, and the communication channel's characteristics. It should be noted that the approaches are divided into two distinct types. Those that focused on the augmentation of the models to capture particular features such as concurrency, event-driven behavior, and real-time behavior and those that leveraged standard models like state space and procedural coding that can be used for code generation or performance analysis. This separation also lets us clearly see those approaches that have included concurrency, event-driven behavior, and real-time behavior, since these three features are crucial for WSN design.

The only technique that we felt did not model node behavior was the work using the Promela Model Checker (Oleshchuk, Sept. 2003). The authors of this work decided to simply focus on the modeling of network connectivity as opposed to including any significant modeling of the node behaviors. Promela though can be used to model and analyze node behaviors.

4.2 Modeling Sensors and Hardware

Most of the modeling techniques surveyed can be used to create a platform independent model. However, even in a platform independent model there is a necessity to include some of the hardware details. One of the reasons for including the hardware details is that the software in the nodes of a WSN is tightly coupled to the hardware elements of the node. Therefore the binding of software and hardware components should be represented in the model. An example of a hardware-software binding is the interaction between the sensor (e.g., humidity, temperature or moisture) and the software component that handles the readings. The sensor type is modeled as a component that uses a communication channel to transfer data to the software components.

Another reason that hardware information may need to be represented is to be able to generate source code from the model. Generated source code is interacting with the node hardware (timers, ports, sensor types) and therefore the model has to be aware of the hardware components in order to generate the correct code (Losilla, Vicente-Chicote, B. lvarez, & Snchez, 2007), (Dietterle, Ryman, & K. Dombrowski, Sept 2004). Finally, hardware representation also helps in the analysis stage. For instance the ADC circuit has to be modeled to calculate the SNR, the communication channel has to be modeled to calculate the BER value.

5. Modeling at the System Level

This part of the paper focuses on modeling contributions to the distributed nature of sensor networks. The modeling techniques deal with various distribution issues, such as network behavior and topology modeling. The modeling techniques that deal with the network system are shown in Table 4.

5.1 Network Behavior

Modeling network behavior in a WSN is crucial because many important performance values are based on the network. For example, the trade-off between packet loss and power. Due to the fact that the node has limited power resources, it is common to use a power management algorithm that controls the wake up state of a node from active to sleeping and vice versa. Packages can be lost if this is not done properly. XRM for example, calculates the package delivery probability. This can be helpful for applications in which package delivery is an important factor.

Also related to power management, modeling the power consumed in the wireless communication process between the nodes is an important factor in increasing the life-time of the WSN. XRM models the power consumed by each wireless communication channel. Every time the node model is provoked to send or receive a signal, a specific amount is subtracted from the energy level. Another example where modeling at the network behavior is important is in capturing the deployment and interaction of the software components across the network. For example, MDEA divides the software elements into two groups, those residing on the nodes and those residing on the gateway. The generated code should guarantee the interaction between the node and the gateway.

In a similar fashion UM-RTCOM models the network elements (sensors, actors, and the gateway) as three virtual machines (VMs), where each VM models a single element. The system behavior is modeled by the interaction between the three VMs.

Table 4 : Modeling in the System Level

Approach	Network Behaviour	Topology Modeling
HL-SDL [7]	-	-
Insense [9]	-	-
Mathworks [10]	Node/base station interaction	Single hop, static topology
MDEA [2]	Node/base station interaction	-
PM [14]	Nodes connectivity	Multi hop-dynamic topology
SensorML [15]	-	-
SystemC-AMS [16]	-	Single hop, static topology
UM-RTCOM [18]	Nodes/actors/base station interaction	Single hop, static topology
XRM [20]	Power management-wake up states	Single hop, static topology

5.2 Topology Modeling

This section focuses on how the topology is modeled, in other words how the physical locations of the nodes have been modeled. The topology of WSN systems can be dynamic or static. The static topology represents the nodes in a fixed location while the dynamic topology represents the nodes while they are in a moving state. PM captures the dynamic topology by recording the physical location of the nodes in a Location Manager (LM). While the nodes change their physical location, they send the updated location to the LM. Through the use of model checking, the nodes connectivity can be checked. Additionally, based on the aim of modeling, the technique models the number of hops in the network design. For instance, SystemC-AMS analyzes the communication channel between two nodes, such that the model deals with single hop communication issues between two nodes.

XRM is an example of modeling for static topologies. The topology is modeled as a grid location. Each node location is captured as a 2D variable. In MathWorks, the framework is able to model the static topology declaration of the nodes connectivity. In the UM-RTCOM model, single hop communication is used between the network nodes because of the application requirements and nature of the problem. The behavior is modeled by the interaction between the three VMs.

6. Supporting Tools

Almost all of the modeling techniques surveyed offers some tools to support the design of WSNs. In this section we discuss support tools that include code generation, execution and analysis, and model checkers (see Table 5).

6.1 Code Generation

Code generation is the process of generating source code from a model or other source code representation. Tools for generating source code from WSN models are beneficial with respect to design for two main reasons:

- Implementing the source code for the nodes is tedious, time consuming and requires a lot of time and effort from the developers.
- Debugging the design at the source code level is also a very challenging and time consuming process.

Modeling can help to solve code implementation problems by designing the system at higher abstraction layers and generating the target code from that layer. The simplicity of the code generation process depends on the degree of similarity between the modeling notation and the generated code notation. The Mathwork technique generates nesC code and C code from ANSI C modeling notation.

Table 5 : Modeling Techniques Supporting Tools

Approach	Code Generation	Model Checking	Execution and Analysis
HL-SDL [7]	NesC	-	WECT
Insense [9]	C	Spin (Channel Protocol)	WCS
Mathworks [10]	NesC, C	-	Functional analysis
MDEA [2]	NesC	-	-
PM [14]		Spin (Connectivity)	
SensorML [15]	JavaBeans	-	-
SystemC-AMS [16]	-	-	BER, SNR
UM-RTCOM [18]	-	-	Deadlock, WCET
XRM [20]	-	Prism, APMC	Execution, debugging

Code is developed through minor changes in the modeling notation versus the generation for nesC needs a lot of the changes for ANSI C to generate the proper code.

One criticism of code generation tools is that the code produced is not as efficient as hand-written source code. Manual optimization by the user is one solution to achieving better performance from generated source code. An example of manual optimization of WSN source code is modifying the communication between the components from asynchronous in the model to synchronous in the target platform. In addition to manual optimization of the generated source code, simulation can be used at the model level to refine the model (with respect to performance) prior to code generation.

Our survey reviewed four modeling techniques which are capable of generating source code: MDEA, HL-SDL, Insense and MathWorks. MDEA and HL-SDL can generate nesC code for WSN. MathWorks generates nesC as well as C code that executes under the MANTIS operating system while Insense generates C code.

6.2 Model Checking

Model checking is a formal methods technique for software engineering [22]. A model checker takes as input a model of a system and a property specification. The model is converted into a finite state model and the model checker uses an exhaustive state space search to verify the specification. The model checker will determine if the model satisfies the specification. If it does not, than a counter example (error trace) may be provided.

Applying model checking to WSN models allows the designer to verify that the design is correctness as well as detect potential errors. In response to errors, the model can be modified and the design improved prior to implementation. Model checking for WSNs can be classified as direct and indirect model checking. Direct model checker occurs when a model checker exists that can take the WSN model as input. An example of direct model checking is in PM where the modeling language, Promela, is also the input language for the model checker Spin [23].

Indirect model checking occurs when no model checker exists for the WSN modeling language and model transformation is required in order to transform the WSN model to a language that can be input to a model checker. For example, XRM models need to be transformed into RM in order to be used with the model checkers PRISM and APMC [22].

A drawback of indirect model checking approaches such as the one used in XRM is that the model checking results are given with respect to the RM model and need to be transformed back into an XRM form. The challenge of transforming between the WSN modeling language and the model checker input language is known as the semantic gap problem. Another example of indirect model checking is in IM where the authors manually created a Promela model of the component communication channel in order to verify the correctness of the communication protocol. Their verification identified an error that lead to a modification to the original Insense model.

6.3 Model Execution and Analysis

In addition to code generation and model checking, we also consider other tool support including tools that execute and analyze the WSN models. Model execution refers to the execution or interpretation of the WSN design at the model level. XRM is the only techniques in our survey supports model execution. The XRM compiler, a domain specific compiler, allows for model execution and debugging as well as model optimizations (e.g., dead code removal).

Model analysis includes a variety of static and dynamic techniques and a number of the approaches in our survey include some kind of analysis tool. The analysis predicts the behavior of the system through calculating some the parameters. The modeling techniques can analysis the system for the system performance such as (Network performance, ADC process, communication process performance). This section contains an explanation of parameters can be calculated by the modeling techniques included in the survey:

- The **deadlock** happens in large scale networks which contains many components. Because the network component are interacting with each other and because every component has a buffer, there is a possibility when all buffers are full and all nodes are waiting for each other to transmit but no node will because of the full buffer. The deadlock analysis shows the probability of that scenario happening so the designer should avoid that in the designing process [24].

- **WCET** is the maximum time length taken to execute the process. WCET is important on the real-time schedulability analysis. While the WSN components are interacting with each other, the commands call and wait are used. For the schedule analysis, it is important to calculate the WCET value. The value basically depends on two factors, the software implementation such as the maximum number of iterations, if conditions, etc and the target hardware platform. Both factors should be annotated in the model in order to calculate the WCET. The hardware factor is represented by timing information of the hardware, such as the time response provided by the manufacture [25].

- **WCS** is the maximum space taken by the components. WCS can be calculated by adding the space requirements of the parameter used by each component. The parameters can be local, which are used by the component only or global parameters, which are used by the components to interact with other components. Signal to Noise Ratio (SNR) is the ratio of the level of the desired signal to the background noise level. SystemC-AMS calculates the SNR ratio of the ADC process [26].

7. Related Work

Akyildiz et al. [1] survey talks about the WSN system design elements such as the routing protocols, the sensors, and application layer. The authors also provide the factors which affect the implementation of each component. In addition, the survey contains information about open research topics for each design element. Our survey talks about the software modeling techniques which are used to capture those system elements.

The survey by Akkaya et al. [27] talks about the routing protocols for wireless sensor networks. According to their classification there are 3 routing categories; data-centric, hierarchical, and location based. The survey discusses each protocol and classifies the protocol based on the 3 mentioned routing categories. The modeling techniques which have been included in our survey have captured the sensors connectivity only. However, none of them have a representation for the routing behavior.

8. Conclusion and Future Direction

Modeling helps to resolve some of the WSN software implementation challenges before deployment. As depicted in Table I, several approaches have focused only on the modeling of software elements in a sensor node while others also model the sensor network. Both of these are important to be modeled from a sensor network perspective. Also many of the modeling languages support components as one of its basic modeling elements.

Component based modeling is a fundamental way to partition software entities because it can be used to support multi-threading design and analysis. Most of the approaches reviewed can also model the communication channel. A few like PM and SensorML can model a process. Process modeling is important in capturing a systems behavior. As depicted in Table II, the modeling techniques are targeted to analyze specific software challenges like concurrency, real-time, and event modeling. We also see that they may be focused on simply modeling the sensor information or hardware to facilitate code design as is the case with MDE and SystemC-AMS.

Modeling at the system level is also another feature that some modelers support. As seen in Table 4, several modeling techniques like UM-RTCOM, XRM, PM, MDEA, and MathWorks can all model the sensor network but there is a focus for each on what behavior they model. They all model node activity and take into account node to node communication but not all can explicitly model the network topology as is the case with MDEA.

Support for analysis and code generation tools is vital for a modeling technique. Table IV reflects, the fact that certain modeling technique can do code generation while others cannot. This depends primarily on the focus of the developers of the modeling technique and the maturity of the approach. It should be noted that very few of the techniques support model checking. We speculate that this is largely due to the gap between the designing process and the model checking. This gap exists because the design takes place in domains such as CORBA and UML. In order to check the model with model checking methods, the design has to be re-modeled again in the model checking domain.

As a future direction for WSN modeling, enhancements for code generation tool are required. Such enhancements can improve the quality of the generated code in terms of the code size and avoidance of manual optimization for the generated code. Additionally, the modeling domain should be selected such that model checking can be done without redoing the model in the model checking domain.

Moreover, the modeling domain should support analysis at the design stage, which helps the software system developer detect and correct software system problems at an earlier stage of the sensor system design. Some of the reviewed papers have performed analysis for WCET, WCS, deadlock, SNR for sensor interfaces, and BER for the communication channel. To the best of our knowledge, package delay and data losses have not been considered in the analysis but these factors are important for sensor networks.

The modeling techniques also could not capture the behaviour of the routing protocol used in the SensIV system. MDEA approach has the facility to mention the routing protocol in the WSN-DSL protocol. However, none of the modeling techniques can capture the behaviour of the routing algorithm. The modeling techniques which are capable of capturing the distributed nature of the network have focused on the interaction between 2 neighbour nodes. Therefore, there is a necessity to state the nodes connectivity since there is no possibility to capture the CTP protocol.

9. Reference

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393-422, March 2002.

[2] F. Losilla, C. Vicente-Chicote, B B. lvarez, and A. Iborra and P. Snchez, "Wireless sensor network application development: An architecture-centric mde approach," in *In Software Architecture volume 4758 of Lecture Notes in Computer Science.*, 2007, pp. 179-194.

[3] R. Liscano et al., "Network Performance of a Wireless Sensor Network for Temperature Monitoring in Vineyards," in *Welcome to the 8th ACM PE-WASUN 2011*, Miami, Fl, USA,

November, 2011.

[4] LM135 Datasheet. [Online]. http://www.national.com/ds/LM/LM135.pdf

[5] *Tinyos 2 tep 123. The collection tree protocol.*: http://www.tinyos.net/tinyos-2.x/doc/txt/tep123.txt.

[6] P. Levis, "TinyOS Programming. Cambridge University Press," 2009.

[7] D. Dietterle, J. Ryman, and and R. Kraemer. K. Dombrowski, "Mapping of high-level SDL models to efficient implementations for TinyOS.," in *In Proc. of Euromicro Symp. on Digital System Design (DSD 2004)*, Sept 2004, pp. 402-406.

[8] ITU-T, "Specification and description language (SDL) z.100(11/99)," 1999.

[9] A. Dearle, D. Balasubramaniam, J. Lewis, and and R. Morrison, "A component-based model and language for wireless sensor network applications," in *Proc. of 32nd Annual IEEE Int. Conf. on Computer Software and Applications (COMPSAC)*, Aug. 2008, pp. 1303-1308.

[10] M. Mozumdar, F. Gregorett, L. Lavagno, L. Vanzago, and S. Olivieri, "A framework for modeling, simulation and automatic code generation of sensor network application," in *Proc. of 5th IEEE Comm. Soc. Conf. on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'08).*, June-2008, pp. 515-522.

[11] A. Bonivento, L. Lavagno, A. Sangiovanni- Vincentelli, and L. Vanzago. L. Necchi, "EERINA: an energy efficient and reliable in-network aggregation for clustered wireless sensor networks Conference," in *In Wireless Communications and Networking*, 2007, pp. 3364-3369.

[12] Simulak. Simulation and Model-Based Design. [Online]. http://www.mathworks.com/products/simulink/

[13] S. Bhatti et al., "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," in *Mobile Networks and Applications.*, 2005, pp. 563–579.

[14] V. Oleshchuk, "Ad-hoc sensor networks: modeling, specification and verification.," in *In Proc. of 2nd IEEE Int. Work on Intelligent Data Acquisition and Advanced Computing Systems Technology and Applications*, Sept. 2003, pp. 76-79.

[15] M. Botts, "OpenGIS sensor model language (SensorML) implementation specification.," Jul. 2007.

[16] M. Vasilevski, N. Beilleau, H. Aboushady, and and F. Pecheux, "Efficient and refined modeling of wireless sensor network nodes using System-AMS," in *In Conf. on Ph.D. Research in Microelectronics and Electronics (PRIME 2008)*, 2008, pp. 81-84.

[17] Crossbow. [Online]. http://www.openautomation.net/uploadsproductos/micaz_datasheet.pdf

[18] M. Diaz, D. Garrido, L. Llopis, B. Rubio, and J. Troya, "A component framework for wireless sensor and actor networks.," in *In Proc. of IEEE Conf. on Emerging Technologies and Factory Automation (ETFA '06)*, Sept. 2006, pp. 300-307.

[19] J. Barbaran, M. Diaz, I. Esteve, D. Garrido, and L. LlopisB Rubio and J. Troya, "Tc-wsans: A tuple channel based coordination model for wireless sensor and actor networks.," in *In Computers and Communications, 2007. ISCC 2007. 12th IEEE Symposium on,* 2007, pp. 173-178.

[20] A. Demaille, S. Peyronnet, and and B. Sigoure., "Modeling of sensor networks using XRM," in *In Proc. of 2nd Int. Symp. on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2006)*, Nov. 2006, pp. 271-276.

[21] J. Zheng, C. Elliott, A. Dersingh, R. Liscano, and M. Eklund, "Design of a Wireless Sensor Network from an Energy Management Perspective," in *Communication Networks and Services Research Conference (CNSR), 2010 Eighth Annual}*, 2010, pp. 80-86.

[22] E. M. Clarke Jr., D O. Grumberg, and A. Peled, "Model Checking," The MIT Press, 1999.

[23] O. Sharma et al., "Towards verifying correctness of wireless sensor network applications using Insense and Spin.," in *In Model Checking Software volume 5578 of Lecture Notes in Computer Science*, 2009, pp. 223-240.

[24] F. L. Lewis, "Wireless Sensor Networks. Smart Environments: Technologies, Protocols, and Applications ed. D.J. Cook and S.K. Das, John Wiley, New York.," in *Smart Environments: Technologies, Protocols, and Applications*. New York, 2004.

[25] M. Diaz et al., "Integrating real-time analysis in a component model for embedded systems.," in *Euromicro Conference*, Sept. 2004, pp. 14-21.

[26] J. Blieberger and R. Lieger, "Worst-case space and time complexity of recursive procedures.," *Real-time system Springer Netherland*, no. 0922-6443, pp. 115-144, 1996.

[27] K. Akkayal and M. Younis, "A survey on routing protocols for wireless sensor networks," *Ad Hoc Networks*, vol. 3, no. 3, pp. 325-249, May 2005.

[28] Manuel Díaz, Daniel Garrido, Luis Llopis, Bartolomé Rubio, and José M. Troya, "A Component Framework for Wireless Sensor and Actor Networks," in *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*., 2006, pp. 300 -307.

[29] B. Lu. and J. Nickerson, "A language for wireless sensor webs," in *In Communication Networks and Services Research, Proceedings. Second Annual Conference on*, 2004, pp. 293-300.