A Survey of Modeling Techniques for Wireless Sensor Networks

John Khalil Jacoub
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
john.khalil@uoit.ca

Ramiro Liscano
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
ramiro.liscano@uoit.ca

Jeremy S. Bradbury
University of Ontario Institute of Technology
Oshawa, Ontario, Canada
jeremy.bradbury@uoit.ca

***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 provide future directions to enhance the use of modeling in the design of WSNs.**

***Keywords*-Wireless Sensor Networks (WSN), Modeling, Code Generation, Model Checking, Analysis.**

I. INTRODUCTION

A WSN consists of small wireless units called motes, which are attached to a specific type of sensors. The sensors measure an environment phenomenon (e.g., humidity, temperature, or soil moisture) [6] and the measured value is expressed as an analog signal generated by the sensors. The analog signal is converted to a digital signal within the mote and transmitted wirelessly to a mote that has expressed interest in that data [21]. This mote is generally called a collector node. The collector mote could also be attached to a gateway to facilitate the transfer of data between the WSN and other devices on more conventional networks like IP. Additionally, some WSN applications have been extended to not only sense the phenomenon but also to react in response to the sensed data. These networks are typically referred to as Wireless Sensor Actor Networks (WSANs) [9].

WSN systems can be complex and many different challenges can arise during the design of a WSN. One challenge is the distribution of nodes and sensors in a physical environment that may result in lost and delayed data. A second challenge is the inclusion of real-time behavior within a distributed WSAN. Many WSANs require real-time communication and control when a specific phenomenon is observed [9]. A third challenge is memory management within the sensor nodes [6]. The small size of the nodes leads to physical limitations that restrict the available memory and therefore memory management is often required. A fourth challenge is operational reliability. WSNs often consist of self-powered nodes in environmentally challenging domains imposing strong reliability requirements. For example, there is a requirement to maximize the life-time of a network because the nodes have a limited power source [20]. A fifth challenge is 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 [15]. A WSN system produced using this approach is prone to both design and implementation errors and very challenging code debugging [16] (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 [20].

The challenges of developing WSN systems often benefit from higher 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 [15]. In this paper we survey 9 modeling techniques for WSNs. For each technique we examine how it models the WSN at the node and system-level.

The rest of the paper is organized into six sections. Section II gives an overview of the modeling technique reviewed in this survey. Section III discusses the modeling of WSN elements including sensors, nodes and hardware while Section IV discusses modeling at the system level. Section V discusses the supporting tools and the importance of each tool for WSN design. Finally, Section VI provides conclusions and future research directions.

II. Background

In this section, we provide an overview of each of the modeling techniques included in our survey (see Table I). 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 an appropriate notation to support the aim of the modeling, such as UML,while others use their own notation, such as the Insense technique [6]. The survey discusses the details of each modeling techniques, but not the contribution of each notation in the area of modeling.

The techniques use different basic elements (e.g., channels, processes, modules, 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.

A. High-Level SDL Models (HL-SDL)

HL-SDL is a modeling language that uses the Specification and Description Language (SDL), which is normally used to model and simulate communication protocols [12] .SDL has been adapted by Dietterle, et al., to model TinyOS components using SDL processes (i.e., extended finite state machines) [10]. 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 [13]. In their work, Dietterle et al. used manual optimization to enhance the generate code [10].

B. Insense

Dearle et al. use the Insense modeling language to create a component-based model for a WSN [6]. 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 [6]. Insense is built in the Contiki operating system(Contiki is a popular operating system used for WSN) [11]. The Insense model has a translator that produces C source code that can be used to calculate important details such as worst case execution times (WCETs) and worst case space (WCS) within a given WSN [6].

Approach	Notation	Modeling Scope	Modeling Elements
HL-SDL [10]	SDL	Node	processes-channels
Insense [6]	Insense Language	Node	components, channels
MathWorks [16]	State Diagram and C	Node-Network	state-charts, communication medium
MDEA [15]	UML	Node-Network	components (wireless link as a class)
PM [19]	Promela	Network	processes, channel
SensorML [4]	XML - source code (optional)	Node	components-processes model
SystemC-AMS [21]	Block diagram - C++	Node-Node Communication	block diagram, source code
UM-RTCOM [9]	CORBA	Node-Network	components, channels
XRM [8]	eXtended Reactive Modules	Node-Network	modules

Table I
Overview of Modeling Techniques

C. MathWorks Modeling Approach

The framework 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 containing 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 WSN 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 [2] and nesC code for TinyOS [13]. The Mathworks approach has been used successfully to generate the code for Energy Efficient and Reliable In-Network Aggregation (EERINA) [17] algorithm for clustered sensor networks.

D. Model Driven Engineering Approach (MDEA)

Losilla, et al. use UML and a Model Driven Engineering (MDE) approach that includes three modeling layers:

- WSN Domain Specific Modeling: a meta-model that is created by a domain expert.
- Component-based Platform Independent Models (PIMs): A UML-like language primarily composed of activity diagrams and state-machine diagrams.

- NesC Platform Specific Model (PSM): used with the UML PIMs to generate the NesC source code.

Transformation rules control moving from one modeling layer to another. Moreover, refinement can occur after every transformation to improve the generated model. This MDE 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. The MDE approach has been used to generate the nesC code for the MITRA WSN application which was designed for the precision agriculture applications [15].

E. Promela Model (PM)

Modeling an Ad-Hoc Sensor Network in Promela is an example of using the input language of a model checker to specify a WSN. The model checker used is Spin and the input language is Promela. The Promela model includes the physical location of all the dynamic nodes and supports adding and removing nodes as well as changing their physical location. The Spin model checker is used to perform a network connectivity check. Specifically, the physical location data of the nodes is analyzed in conjunction with the data about the coverage range of the sensors [19].

F. SensorML

SensorML is an XML based language that supports modeling each sensor by specifying the sensor's meta-data (e.g., sensor Id, sensor type). The model includes representations of the physical elements (e.g., the sensor and actuators) and the non-physical element (e.g., mathematical operation within the sensor). All of the elements are modeled as processes that are linked together explicitly through inputs and outputs. Linked sequences of processes form process chains that correspond to the behavior inside a single node [4]. Sensor Web Language (SWL) is a simple version of SensorML and is used in the WSNs deployments to achieve the interfaces between the network elements base station, sensors, and web browser. Additionally, the language can be used to achieve the interface between two or more different WSNs [18].

G. SystemC-AMS

SystemC-AMS combines C++ with block diagrams to model the WSN and simulate the system. For each node, SystemC-AMS models the Analog to Digital Converter (ADC), the microprocessor, and the channel. The goal of SystemC-AMS, is to calculate the Signal to Noise Ratio (SNR) for the ADC and Bit Error Rate (BER) for the communication channel between two nodes [21].

H. UM-RTCOM Model

UM-RTCOM is a real-time component based modeling framework written in CORBA that is composed of sensors, actors and a coordinator (the coordinator in located on a base station). Actors gather data from groups of sensors based on their physical locality and respond to a phenomenon identified by the coordinator [9]. 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: (t1, t2, ..., tn) where each field ti 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 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. [1]. The deployment shows the improvement of the middleware overhead compared to another deployment where the motes send the sensed data periodically to the actors.

I. eXtended Reactive Modules (XRM)

XRM is an extension language of Reactive Modules (RMs). Demaille, et al. used WSNs as a case study to evaluate the XRM modeling language [8]. 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 and APMC [7].

III. 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 II).

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.

A. Node Behaviors

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 could be 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 were later used for code generation or performance analysis. This seperation 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 paper using the Promela Model Checker [19]. 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 bahaviors.

B. 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 [6].

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 [15], [10].

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 model to calculate the BER value [21].

Approach	**Node Behaviors**	**Sensors & Hardware Modeling**
HL-SDL [10]	concurrency, event-driven	-
Insense [6]	concurrency, real-time	sensor types
MathWorks [16]	procedural, state space	-
MDEA [15]	procedural, state space	timers, ports, wireless channel
PM [19]	-	-
SensorML [4]	event-driven	sensor types
SystemC-AMS [21]	procedural	ADC, microprocessor, wireless channel
UM-RTCOM [9]	concurrency, real-time, event-driven	-
XRM [8]	procedural, state space	-

Table II
MODELING OF WSN ELEMENTS

IV. 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 III.

A. 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 the package deliverance 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 [8].

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 [15].

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 [9].

B. Topology Modeling

This section focuses on how the topology is modeled, in the 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. Ad-Hoc SNMC 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 [19]. Additionally, based on the modeling target, the technique models the number of hops in the network design. For instance, SystemC-AMS analyzes the communication channel between two nodes,

Approach	Network Behavior	Topology Modeling
HL-SDL [10]	-	-
Insense [6]	-	-
MathWorks [16]	Node/base station interaction	Single hop, static topology
MDEA [15]	Node/base station interaction	-
PM [19]	Nodes connectivity	Multi hop, dynamic topology
SensorML [4]	-	-
SystemC-AMS [21]	-	Single hop, static topology
UM-RTCOM [9]	Nodes/actors/base station interaction	Single hop, static topology
XRM [8]	Power management-wake up states	Single hop, static topology

Table III
MODELLING AT THE SYSTEM LEVEL

Approach	Code Generation	Model Checking	Execution & Analysis
HL-SDL [10]	NesC	-	WCET
Insense [6]	C	Spin (Channel protocol)	WCS
MathWorks [16]	NesC, C	-	functional analysis
MDEA [15]	NesC	-	-
PM [19]	-	Spin (Connectivity)	-
SensorML [4]	JavaBeans [14]	-	-
SystemC-AMS [21]	-	-	BER, SNR
UM-RTCOM [9]	-	-	Deadlock, WCET
XRM [8]	-	Prism, APMC	execution, debugging

Table IV
SUPPORTING TOOLS

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 modeled as an X-Y variable [8].

In MathWorks, the framework is able to model the static topology by modeling the nodes with state chart and the communication medium which is implemented in C, models the connectivity between the network node [16]. In the UM-TRCOM model, single hop communication is used between the network nodes because of the application requirements and nature of the problem. behavior is modeled by the interaction between the three VMs [9].

V. 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 IV).

A. Code Generation

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

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

Modeling can help to solve these problems by designing the system at higher abstraction layers and generating the target code from that layer [15]. 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. Generation of C 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 [16].

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 [16]. 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 [10].

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 WSNs [15]. MathWorks generates nesC as well as C code that executes under the MANTIS operating system [16] while Insense generates C code [6].

B. Model Checking

Model checking is a formal methods technique for software engineering [5]. 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 [19].

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 [8]. 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 know 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 [20].

Model checking has also been used to check the network topology of WSNs [3]. However, we have excluded this work from our survey since our primary focus is on software design.

C. 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) [8]. 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. We will not discuss some of the analysis provided by these tools.

Real-time behavior is an important system requirement for WSANs and real-time design often includes schedulability analysis. Schedulability analysis includes WCET (the maximum time length taken to execute a process), WCS (the sum of the space requirements of each component's parameters), and deadlock analysis [6] [9]. HL-SDL, Insense and UM-RTCOM provide some form of schedulability analysis (see Table IV).

Other modeling techniques to include analysis tools are MathWorks and SystemC-AMS. MathWorks includes functional analysis of the WSN algorithms. SystemC-AMS calcules factors to judge the electronic system of the nodes by simulation. The first factor is SNR for the ADC process inside the node. The second factor is BER for the wireless communication channel [21].

VI. Conclusions and Future Directions

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 sensory 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 III, 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.

References

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

[2] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Networks and Applications*, 10:563–579, 2005.

[3] S. Bhatti, J. Xu, and M. Memon. Model checking of a target tracking protocol for wireless sensor networks. In *Proc. of IEEE 10th Int. Conf. on Computer and Information Technology (CIT'10)*, pages 2867–2872, Jul. 2010.

[4] M. Botts and A. Robin. OpenGIS sensor model language (SensorML) implementation specification. Technical report, OGC, Jul. 2007.

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

[6] A. Dearle, D. Balasubramaniam, J. Lewis, 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 '08)*, pages 1303–1308, Aug. 2008.

[7] A. Demaille. Probabilistic verification of sensor networks. In *Proc. of 4th IEEE Int. Conf. on Comp. Sci., Research, Innovation and Vision for the Future (RIVF'06)*, pages 45–54, 2006.

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

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

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

[11] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proc. of 29th Annual IEEE Int. Conf. on Local Computer Networks*, pages 455–462, Nov. 2004.

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

[13] P. Levis. *TinyOS Programming*. Cambridge University Press, 2009.

[14] R. Liscano and K. Kazemi. Integration of component-based frameworks with sensor modelling languages for the sensor web. In *Proc. of IEEE GLOBECOM Workshops (GC Wkshps)*, pages 235–240, Dec. 2010.

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

[16] M. Mozumdar, F. Gregoretti, 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)*, pages 515–522, Jun. 2008.

[17] L. Necchi, A. Bonivento, L. Lavagno, A. Sangiovanni-Vincentelli, and L. Vanzago. E2rina: an energy efficient and reliable in-network aggregation for clustered wireless sensor networks. In *Wireless Communications and Networking Conference*, pages 3364 –3369, 2007.

[18] B. Nickerson and J. Lu. A language for wireless sensor webs. In *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, pages 293 – 300, May 2004.

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

[20] O. Sharma, J. Lewis, A. Miller, A. Dearle, D. Balasubramaniam, R. Morrison, and J. Sventek. Towards verifying correctness of wireless sensor network applications using Insense and Spin. In *Model Checking Software*, volume 5578 of *Lecture Notes in Computer Science*, pages 223–240. 2009.

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