Integrated Modeling and Simulation Framework for Wireless Sensor Networks

Baobing Wang and John S. Baras
Department of Electrical and Computer Engineering, University of Maryland, College Park
{briankw, baras}@umd.edu

***Abstract*—Existing ad hoc system design methods for Wireless Sensor Networks (WSNs) suffer from lack of reusability. In addition, the interactions between the continuous-time physical environments and WSNs have not been well studied. In this paper, we propose a model-based systems design (MBSD) framework for WSNs, which is a systematic methodology applying systems engineering principles to enhance model reusability and collaborations among multiple modeling domains. Firstly, we describe a hierarchy of model libraries to model various behaviors and structures of WSNs, including physical environments, physical platforms, communication and computation components, system services and applications. Based on the MBSD framework, we introduce a system design flow to compose both continuous-time and event-triggered modules to develop applications with support for performance study by simulations. Next, we briefly describe the main modules for physical platforms, the Media Access Control (MAC) layer, wireless channels and physical environments, which are developed using the Systems Modeling Language (SysML), Simulink and Modelica. Finally, we use a building thermal control system as the case study to demonstrate the composability, reusability and flexibility of the proposed MBSD framework.**

I. Introduction

Wireless Sensor Networks (WSNs) are engineered systems consisting of closely interacting physical environments, physical platforms, communication protocols and computation algorithms. The design of such hybrid systems requires a systems engineering view and an integrated design framework that can support joint event-triggered and continuous-time dynamics [1]. In addition, since hybrid systems usually are very complex and too costly to be designed from scratch, component reusability can never be overemphasized. Developing such a design framework for WSNs faces several challenges. For example, due to the wide variety of WSN applications and the heterogeneity of sensor platforms, it is difficult to figure out the primitive function modules, which are imperative for reusability. In addition, integrating the numerical solvers for continuous-time models with event-triggered models is an established, but far-from-trivial problem.

Several works have tried to improve the code reusability of sensor network protocols. Klues et al. [2] proposed a component-based architecture for the MAC layer in WSNs. Ee et al. [3] introduced a modular network layer to enable co-existing protocols to share and reduce code and resources consumed at run-time. However, both works focus on protocol implementations rather than system designs.

Viptos, a joint modeling and design framework for WSNs, was proposed in [4]. Mozumdar et al. [5] presented a similar work modeled in Simulink. They can analyze the performance of system designs by simulations and generate the TinyOS application codes. Another work [6], introduced a method to integrate Simulink and ns-2 for hybrid networked control systems. However, sensor behaviors in these works are tightly coupled with communication protocols, which makes their components hard to be reused.

Samper et al. [7] presented an approach for the formal modeling and analysis of WSNs at various abstraction levels. Formal model checking tools can be applied to verify their models of hybrid systems. However, trade-off analysis is not considered and the component reusability is not clearly supported in their approach. Moppet, a model-driven performance engineering framework for WSNs, was proposed in [8], which is the closest work to our framework. Moppet enables users to design WSN applications using the model libraries and estimate their performances using event calculus and network calculus without simulations. However, the continuous dynamic behaviors of physical environments are not considered.

In this paper, we propose a model-based system design framework for WSNs, which applies system engineering principles to model both event-triggered and continuous-time components. Firstly, the MBSD framework is proposed, which provides a hierarchy of system model libraries for applications, system services, computation and communication algorithms, physical platforms and physical environments to support a plug-and-play design fashion. Event-triggered components are modeled in SysML and statechart diagrams are exploited to model their behaviors. Continuous-time components are modeled in Simulink or Modelica and their behaviors are described by differential equations. To make our ideas more clear, the model libraries for the MAC layer, physical platforms, wireless channels and physical environments are briefly explained.

Secondly, based on the MBSD framework, a system design flow is proposed, which can integrate both event-triggered and continuous-time modules. With the help of IBM Rational Rhapsody [9], Simulink and C/C++ source files can be generated automatically from the MBSD framework, which can be used for performance study and interactive simulations.

Finally, a building thermal control system is used as the case study to demonstrate the reusability and flexibility of the proposed MBSD framework. In this example, we illustrate how hybrid systems can be easily developed using the modules in the model libraries, and how their performance can be studied. Using the MBSD framework, developers can focus on the system design strategies, rather than implementation

1524-4547/12 $26.00 © 2012 IEEE
DOI 10.1109/WETICE.2012.28

details that are usually not familiar to system experts.

The rest of this paper is organized as follows. We introduce the proposed MBSD framework and design flow in Section II. In Section III, we describe the main modules in some model libraries. The case study is presented in Section IV. Finally, Section V concludes this paper.

II. Overview of the Framework

In this section, we first introduce the proposed MBSD framework, with a hierarchy of model libraries to model various behaviors and structures of WSNs. Then we describe the design flow to develop applications with support for automatic code generation for simulations.

A. Hierarchy of System Models

The MBSD framework provides the model libraries for applications, services, computation algorithms, communication protocols, physical platforms and physical environments.

Application Model Library. A WSN application can be specified with function requirements, performance requirements, physical platforms and the physical environment where the sensor network will be deployed. The Application Model Library provides modules that can precisely describe common sensor network applications. In addition, special applications can also be modeled by extending proper models in the library.

Service Model Library. Most WSN applications share several common features that are used frequently, such as the query service to retrieve data locally or remotely, the naming service to uniquely identify motes locally or globally, the location service to compute the virtual or physical locations and regions of sensor motes, etc. The Service Model Library provides modules for these common services with interfaces to customize to fulfill the application requirements.

Network Model Library. This library resides in the center of the MBSD framework, consisting of communication modules, computation modules and data management modules that are necessary to implement various algorithms and protocols to accomplish the upper layer services. By well defining their interfaces, different algorithms and protocols can be studied and compared systematically in a plug-and-play way, using components of the same functionalities and ports, but with different implementations.

Physical System Model Library. This library is composed of modules for various physical platforms in heterogeneous WSNs. Despite their different capacities and computation powers, these platforms can be viewed to be composed of at most four parts: CPU, sensor, transceiver and battery. We have distilled the various common primitives and parameters to describe these components and their ports. In addition, this library provides wireless channel models with different radio propagation models, channel fading models and bit error rates under different modulation schemes.

Environment Model Library. This library serves as the bridge between the continuous-time domain and the event-triggered domain. On one hand, physical environments usually exhibit continuous dynamic behaviors. On the other hand, algorithms and protocols in WSNs are usually event-driven and exhibit discrete dynamic behaviors. This library provides modules to exchange information between these two domains.

All event-triggered components are modeled in SysML using statecharts and primitive operations are implemented in C/C++. All continuous-time components are modeled using Simulink or Modelica, which are then compiled to generate S-Funtions and imported to SysML. Continuous data are passed through flow ports, while events and discrete data are exchanged via rapid ports.

B. System Design Flow

Based on the MBSD framework, we propose a system design flow (Fig. 1) to compose both event-triggered and continuous-time modules.

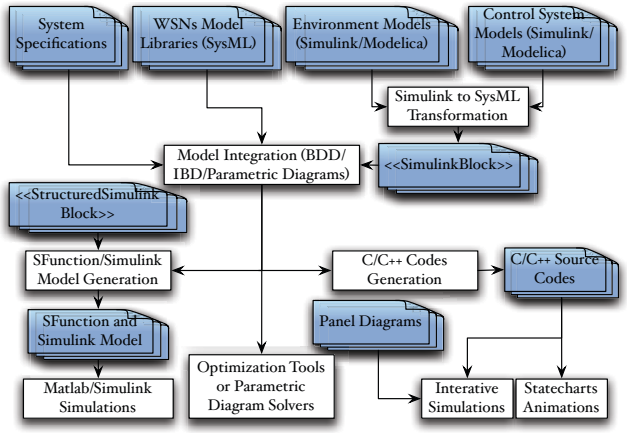

Fig. 1. System Design Flow for Model Integration

WSN applications are developed by composing proper modules from the model libraries according to their system specifications, which specify their functionality, performance requirements, environments, hardware platforms, algorithms and protocols used in the systems, etc.

After all components have been composed using the Block Definition Diagrams (BDDs), Parametric Diagrams and Internal Block Diagrams (IBDs) in SysML, the complete system models can be used in the following three ways.

Simulate in Matlab/Simulink. The Simulink profile offers the `<<StructuredSimulinkBlock>>` stereotype that can be applied to create a Simulink simulation environment. Other SysML blocks and Simulink blocks can be contained as subcomponents, and the compositions can be specified in IBDs. From this Simulink structure block, we can generate a Simulink source file in which all SysML blocks in IBM Rational Rhapsody are transformed into a single S-Function and all Simulink blocks remain the same. After that, we can simulate the whole system in Matlab/Simulink. This simulation method is useful for performance study.

Simulate in Rational Rhapsody. C/C++ source codes for the whole system model can be generated directly from the MBSD framework, which can be used to simulate the whole

system in Rhapsody. Two technologies are available for model validation: statechart animation and interactive simulation. *Statechart animation* enables us to validate system models by tracing, and debug system designs at the design level rather than the source code level, by actually executing system models and animating various SysML statechart diagrams. *Interactive simulation* enables us to design GUIs in a drag-and-drop fashion using the components provided in Panel Diagrams. By wiring these GUI components with the states and attributes of SysML blocks, we can observe their values and modify the configurations interactively in the runtime without recompiling the models. This simulation method is useful for system debug and validation.

Trade-off Analysis and Design Space Exploration. The relationships between component design parameters can be specified in Parametric Diagrams, which can be solved by Parametric Diagram solvers for trade-off analysis and design space exploration. This is beyond the scope of this paper.

III. System Component Models

Modeling WSNs is generally a complex task due to the wide variety of WSN applications, the heterogeneity of physical platforms, and the complex interactions with their physical environments. In order to enhance the reusability, it is imperative to identify primitive function components in different layers and the interfaces for them to interact with each other. In this section, we briefly describe the main modules for physical platforms, the MAC layer, wireless channels and physical environments[1].

A. Modeling Wireless Sensor Networks

Sensor motes (wireless routers or base stations) consist of physical platforms and softwares (i.e., algorithms and protocols), communicating through wireless channels. Sensor motes provide ports to get the clear channel assessment (CCA), query environment phenomenon data, register themselves to the wireless channel component and send/receive packets.

1) Physical Platforms: A physical platform is composed of four parts: battery, CPU, sensor and transceiver. Their interactions in a typical composition of a physical sensor platform is shown in Fig. 3. We describe the transceiver module in detail as an example, whose behaviors are modeled as a state machine in Fig. 2.

Each transceiver has four power states, whose power levels are given by `sleepPower`, `standbyPower`, `txPower` and `rcvPower`, respectively. To support the Unit Disk Graph (UDG) model, the `txRange` attribute can specify the transmission range. The transceiver sends the energy consumption information to the battery periodically and transits to the termination state if an `evDead` event is received. In addition, it will transit its power state accordingly if an `evMoteCtrl` message is received.

If an `evMACCCAQ` query is received from the MAC layer, which requires to poll the channel to get the CCA information,

[1]For more details, please refer to http://www.ece.umd.edu/~briankw/resources/Wang_WETICE_2012_Full.pdf

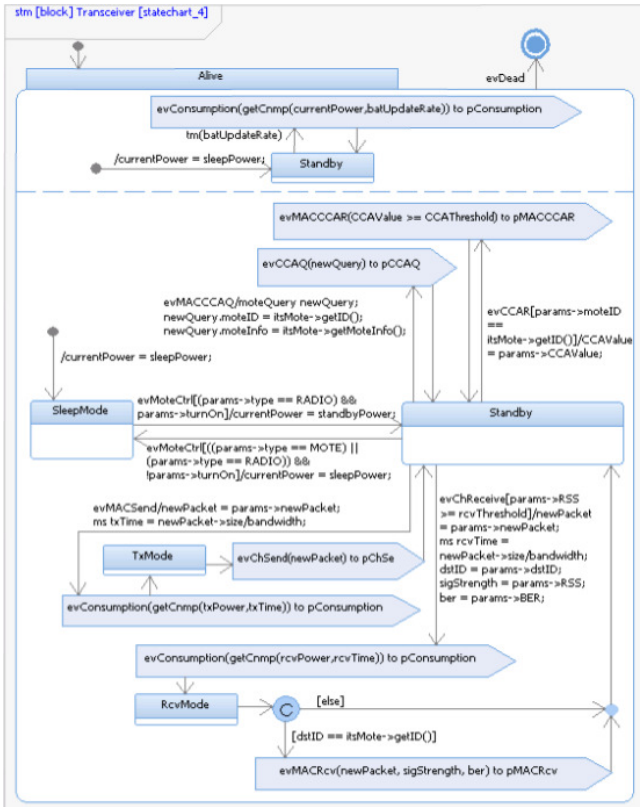

Fig. 2. Behavior Model of a Transceiver Using Statechart

the transceiver will forward this query with the mote ID to the wireless channel component via `pCCAQ`. The reply `evCCAR` that contains the strength of the strongest signal in the channel near this mote is returned by the wireless channel component through `pCCAR`. The transceiver compares this signal strength with its carrier sense threshold (specified by the `CCAThreshold` attribute) and sends the result (`true` or `false`) to the MAC layer through `pMACCCAR`.

If an `evMACSend` request that contains the packet to be sent is received from the MAC layer, the transceiver will forward this packet to the wireless channel component through `pChSend`. Furthermore, the amount of energy consumed to send this packet is sent to the battery component. If an `evChReceive` event that contains the packet forwarded by the wireless channel component is received from `pChReceive`, the transceiver will check the received signal strength (RSS) and the destination of the packet. If it is the destination and the RSS exceeds its receive sensitivity (specified by `rcvThreshold`), it will forward this packet to the MAC layer through `pMACRcv`. Otherwise, this packet will be dropped. Similarly, the energy consumption for packet processing is sent to the battery component.

2) MAC Layer Components: A MAC layer component provides the ports for upper layers to send and receive packets, set the transmission power of the transceiver and control the power states of the physical sensor platform. The main

subcomponents in this layer are described as follows, some of which are developed based on [2]. A composition example is shown in Fig. 4.

The *Low Power Listener (LPL)* component is responsible to adjust the transceiver's power state based on channel activity. Both the fixed LPL listener and the periodic LPL listener are provided in the MBSD framework. The *CSMA/CA Channel Access* component is responsible to gain the channel access right for a transmission using the CSMA/CA mechanism. The *CSMA/CA Sender* component is responsible to send a packet using the CSMA/CA mechanism. The *Slot Manager* component manages the slot schedule for TDMA mechanism. The *TDMA Sender* component is similar to the CSMA/CA Sender, except that packets are sent using the TDMA mechanism. The *Receiver* component is responsible to broadcast received packets to the MAC Controller and other protocol-specific components for further process. The *Queue Manager* component is responsible to buffer packets in the MAC layer.

The *MAC Controller* component is the only one that needs to be customized by users. This component specifies the control logic of a MAC protocol. Every MAC protocol should extend this abstract component and implement the protocol-specific behaviors. The ports to interact with other modules have been defined, including components in upper layers and other components in the MAC layer.

3) Wireless Channels: Wireless channel components model various wireless channels with different radio propagation models, channel fading models and bit error rates (BERs) under different modulation schemes. Each network instance usually has only one wireless channel component, to which all sensor motes, actuators, wireless routers and base stations must register themselves with their IDs, physical positions and transmission powers/ranges.

Since the channel component interacts with all nodes in a network instance, it may not be able to process all requests immediately. Therefore, the channel component needs to buffer the received requests in FIFO queues. In addition, the channel component needs to maintain the information of all nodes in the network, including their IDs, physical positions and transmission powers/ranges. Furthermore, this component needs to maintain the information of all ongoing transmissions in the channel, including the physical positions and transmission powers/ranges of the senders, the start time-stamps and their required transmission times. These information are essential for the evaluation of CCAs and BERs.

B. Modeling Physical Environments

Environment phenomena (e.g., temperature and humidity) usually exhibit continuous dynamic behaviors, which are typically described by differential equations according to their physical laws and modeled using Matlab/Simulink or Modelica. Simulink is a generic data-flow simulation tool good for modeling control systems. Simulink models are first built using the Embedded Coder in Matlab to generate C/C++ source codes, and then imported as SysML blocks to Rational Rhapsody. Modelica is a more powerful topological-based modeling language with support for symbolic manipulation of equations and non-causality, which makes it excellent for modeling plants and physical world. Modelica models should be transformed to Simulink S-Functions first [10]. The outputs of their solvers are forwarded to the *Phenomenon* module through flow ports.

The *Phenomenon* component serves as the interface between the continuous-time domain and the event-triggered domain. The basic Phenomenon component periodically sends new phenomenon information to the Environment component. The *Environment* component models the propagation of information that are received from the Phenomenon module. It accepts queries from sensors, computes the phenomenon values based on the propagation model and the distances between sensors and the phenomenon, and returns the results to sensors. Several Environment modules can be composed in a network for different environment regions. For example, two such modules are included in our case study with one for each room.

IV. Case Study

The MBSD framework proposed in this work is intended to provide a reusable and extensible mechanism for system design and performance simulations. In this section, we present a simple building thermal control system as the case study to demonstrate the composability, reusability and power of the MBSD framework.

A. Building Thermal Control System

A building thermal control system is responsible to control the temperature inside a building so that people can feel comfortable and equipments can work in a optimal condition. In addition, it also needs to reduce the energy consumed by heaters and air conditioners (ACs).

In this case study, we consider a simple building that consists of two large rooms: the living room and the data center. Each room has a desired temperature, which is usually 22 °C and much higher than the environment temperature in the winter. Therefore, a heater is needed in the living room to generate warmth. On the other side, the temperature in the data center will naturally rise because the large amount of electrical power used by the computer systems will heat the air. Consequently, an AC is needed in the data center to keep the temperature at the desired level.

An efficient way to reduce the energy consumption is to use the heat emitted by the computer systems to heat the living room through a pipe. A central control system decides when the heater, AC and pipe should be turned on or off. One temperature sensor is deployed in each room, which sends the room temperature to the control system in the base station through the wireless channel. The commands from the control system are sent to the heater, AC and pipe directly.

In this case study, we assumed the IEEE 802.15.4 unslotted CSMA/CA mode [11] is used as the MAC protocol, and both the two sensors and base station can communicate with the personal area network (PAN) coordinator directly. The compositions of the main components are introduced as follows.

1) Physical Platforms: The temperature sensors, base station and PAN coordinator can be composed using components from the physical system model library. The internal composition of a temperature sensor is shown in Fig. 3. The base station and PAN coordinator can be composed in the similar way but without the sensor component.

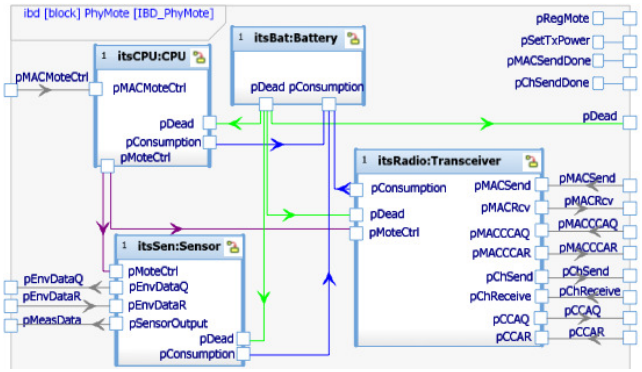

Fig. 3. SysML Internal Block Diagram for Physical Sensor Motes

2) IEEE 802.15.4 MAC Protocol: The unslotted CSMA/CA MAC protocol for each node can be composed using components from the network model library. The internal composition of the MAC protocol for the PAN coordinator is shown in Fig. 4. Both the sensors and base station act as RFDs, whose MAC protocols can be composed in the similar way but without the queue manager component, and the PAN controller component is replaced with a RFD controller component.

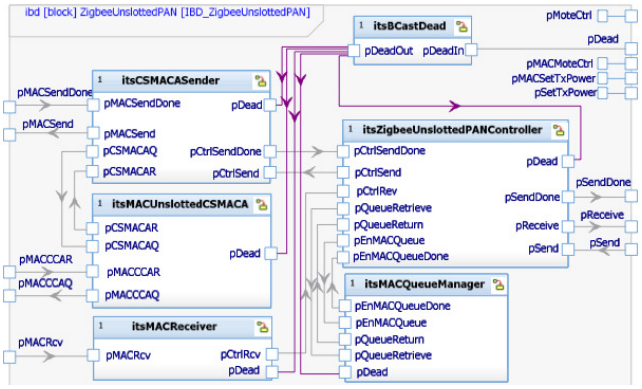

Fig. 4. SysML Internal Block Diagram for the IEEE 802.15.4 Unslotted CSMA/CA Mode for the PAN Coordinator

3) Building Thermal Model in Simulink: The control system and building thermal dynamics are modeled in Simulink. The thermal dynamics model of the Data Center is shown in Fig. 5 as an example.

The control system decides when the heater, AC and pipe should be turned on or off based on the following rules:

- Heater: turn on if $T_{LR} \leq 20$ and turn off if $T_{LR} \geq 24$
- AC: turn on if $T_{DC} \geq 24$ and turn off if $T_{DC} \leq 20$
- Pipe: turn on if $T_{LR} \leq 22$ and turn off if $T_{LR} \geq 24$

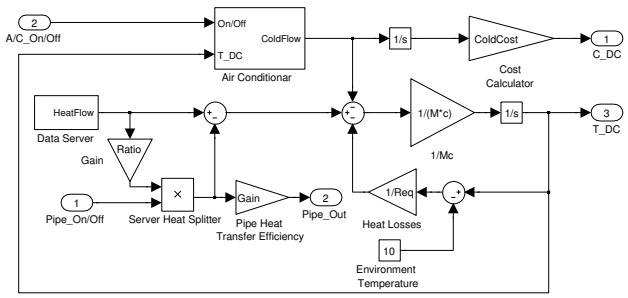

Fig. 5. Simulink Thermal Dynamics Model of the Data Center

where T_{LR} and T_{DC} are temperatures of the living room and data center, respectively. Here, a variance of 2 °C around the desired temperature is allowed to avoid oscillations.

4) Overall System Composition: After all required components have been composed using the model libraries, they can be connected together to model the whole system. The overall system composition of our case study is shown in Fig. 6. The channel component with the ITU indoor propagation model [12], Rayleigh fading and BPSK modulation scheme is selected for this system. The connections between the channel component and the node components are not shown here for clarity. Ports with the same name (indicated by the same color) should be connected. The `SL_ControlSystem` component is used to import the Simulink models by applying the `<<SimulinkBlock>>` stereotype.

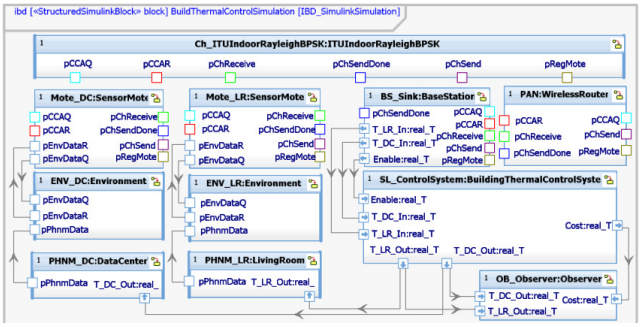

Fig. 6. SysML Internal Block Diagram for the Whole System

B. Evaluation

The Simulink source file for the overall system is generated from the Simulink structure block in Figure 6 automatically, which is then simulated in Matlab/Simulink. The following four scenarios are considered in our simulations:

- **Wireless + No Pipe**. The temperatures are collected to the base station using the WSN, but the pipe is never turned on. Each sensor measures once every 5 seconds.
- **Wireless + Pipe** ($5s$). Similar to the above scenario, but the pipe feature is enabled.
- **Wireless + Pipe** ($60s$). Similar to the above scenario, but the sensor sleep interval is 60 seconds.

- **Wired + Pipe**. The room temperatures are fed back to the control center directly. This scenario is used as the reference to study the impacts of the delays in the WSN.

The temperatures of the environment, the heater, the AC and the air flow from the computer systems are 10 °C, 50 °C, 4 °C and 50 °C, respectively. The air flow rate of the AC, the heater and the computer systems are 2 kg/s, 2 kg/s and 0.5 kg/s, respectively. We assume 50% of the air flow from the computer systems can be piped out and 30% of their heat can be delivered to the living room. The MAC protocol uses the default parameter values specified in [11].

The initial temperature of both rooms is 20 °C. The simulation results for the first 2 hours of the room temperatures and cost are shown in Figure 7, and the working statuses of the AC, heater and pipe are shown in Figure 8. The results indicate that the pipe is working efficiently, which can decrease the working time of the heater and AC, and thus reduce the total electricity cost by 24%. When the sensors wake up to measure the temperature once every 5 seconds, the impacts of the delays on the system performance are negligible. However, if the interval between two successive measurements is increased to 60 seconds, the room temperatures may cross the desired boundaries, which should be avoided. This can be used to study the trade-off between the system performance and energy efficiency of the sensor motes.

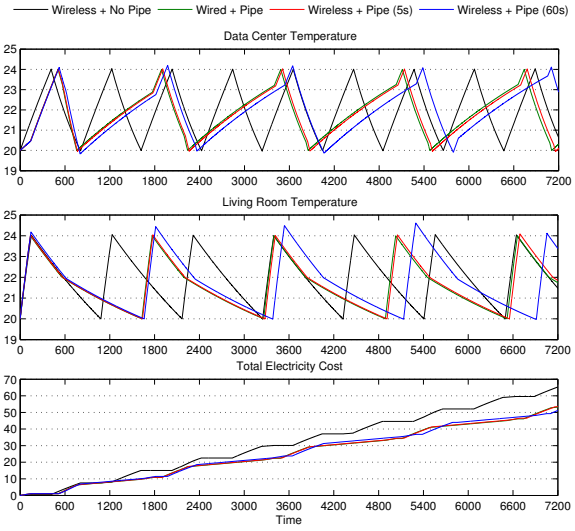

Fig. 7. Room Temperatures and Total Electricity Cost

V. Conclusions

In this work, we have proposed a model-based system design framework for WSNs, which can model both continuous-time and event-driven components, and integrate them by composition for performance study by simulations. SysML, Simulink and Modelica that are standard modeling languages in the industry are used to develop the model libraries. The main component models for physical platforms, the MAC layer, wireless channels and physical environments are described briefly. A hybrid system is used as the case study to demonstrate the composability, reusability and flexibility of the MBSD framework.

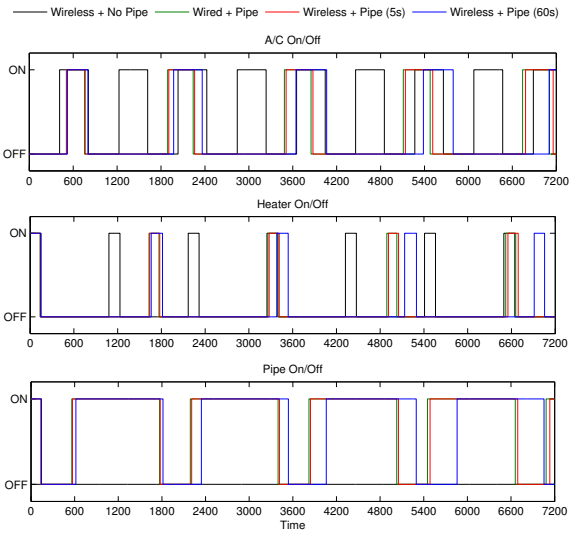

Fig. 8. Working Statuses of the A/C, Heater and Pipe

Acknowledgment

Research partially supported by DARPA and SRC through grant award 013641-001 of the FCRP, and by the National Science Foundation under grant CNS-1035655.

References

[1] P. Derler, E. A. Lee, and A. Sangiovanni-Vincentelli, "Modeling cyber-physical systems," *Proceedings of the IEEE Special Issue on CPS*, December 2011.

[2] K. Klues, G. Hackmann, O. Chipara, and C. Lu, "A component-based architecture for power-efficient media access control in wireless sensor networks," *SenSys*, pp. 59–72, November 2007.

[3] C. T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica, "A modular network layer for sensornets," *USENIX OSDI*, 2006.

[4] E. Cheong, E. A. Lee, and Y. Zhao, "Joint modeling and design of wireless networks and sensor node software," *Intl. Sympo. on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, 2006.

[5] M. M. R. Mozumdar, F. Gregoretti, L. Lavagno, L. Vanzago, and S. Olivieri, "A framework for modeling, simulation and automatic code generation of sensor network applications," *SECON*, 2008.

[6] D. Riley, E. Eyisi, J. Bai, X. Koutsoukos, Y. Xue, and J. Sztipanovits, "Networked control system wind tunnel (ncswt)- an evaluation tool for networked multi-agent systems," *SIMUTools*, 2011.

[7] L. Samper, F. Maraninchi, L. Mounier, and L. Mandel, "Glonemo: global and accurate formal models for the analysis of ad-hoc sensor networks," *1st Intl. Conf. on Integrated Internet Ad Hoc and Sensor Networks (InterSense)*, 2006.

[8] P. Boonma and J. Suzuki, "Moppet: A model-driven performance engineering framework for wireless sensor networks," *The Computer Journal*, vol. 53, no. 10, pp. 1674–1690, 2010.

[9] IBM, "IBM Rational Rhapsody Help," *www.ibm.com/software/awdtools/rhapsody/*.

[10] Dassault Systemes, "Dymola," *www.dynasim.se/*.

[11] IEEE 802.15 TG4, "IEEE 802.15.4 Standard," *http://www.ieee802.org/15/pub/TG4.html*.

[12] ITU-R, "Propagation data and prediction models for indoor radio communication systems," *ITU-R Recommendations*, 2001.