Model-Based Design Framework for Wireless Sensor Networks Using SysML, Simulink and Modelica

Baobing Wang and John S. Baras
Department of Electrical and Computer Engineering
University of Maryland, College Park, MD 20740, USA
{briankw, baras}@umd.edu

ABSTRACT

Existing ad hoc system design methods for Wireless Sensor Networks (WSNs) suffer from lack of reusability, trade-off analysis and design space exploration methods and tools. 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 support system requirements, design, analysis and verification/validation processes. 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-time modules to develop applications with support for trade-off analysis, design space exploration and interactive simulations. Next, the main modules for physical platforms, the Media Access Control (MAC) layer, wireless channels and physical environments are described in detail, and are modeled in 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.

Keywords

Model-Based System Design, Wireless Sensor Networks, SysML, Hybrid Systems Modeling

1. 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 [5]. In addition, since hybrid systems usually are very complex and too costly to be designed from scratch, component reusability can never be overemphasized. Finally, hybrid systems design usually needs to consider multiple objectives that may conflict with each other. For the same application, several design alternatives with different performances may be available. Therefore, the design framework must support trade-off analysis and design space exploration.

Developing such a design framework for WSNs faces three significant challenges. Firstly, 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. Secondly, integrating the numerical solvers for continuous-time models with event-triggered time models is an established, but far-from-trivial problem. Finally, mission-critical applications usually require model checking, to guarantee certain properties (e.g., dead lock freeness, liveness and reachability), which is a very time-consuming process. It is still an open problem on how model libraries can be developed to enhance the scalability of model checking. The involvement of continuous-time models makes this problem more complex.

Several works have tried to improve the code reusability of sensor network protocols. Klues et al. [10] proposed a component-based architecture for the MAC layer in WSNs. Ee et al. [6] 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 [2]. Mozumdar et al. [11] 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 [12], introduced a method to integrate Simulink and ns-2 for hybrid networked control systems. However, only simulations are considered in these works and sensor behaviors are tightly coupled with communication protocols, which makes their components hard to be reused.

Samper et al. [13] 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 [1], 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. Our contributions in this paper are three-fold.

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. Event-triggered-time 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 enhance the component reusability, modules of primitive functionality have been distilled, with clearly defined ports for them to exchange messages and data flows. Consequently, the MBSD framework supports a plug-and-play design fashion. To make our ideas more clear, the model libraries for the MAC layer, physical platforms, wireless channels and physical environments are explained in detail.

Secondly, based on the MBSD framework, a system design flow is proposed, which can integrate both event-triggered-time and continuous-time modules by composition for trade-off analysis, design space exploration and interactive simulations. Using IBM Rational Rhapsody [7] as the development environment, SysML Parametric Diagrams that describe the relationships between the system performance and the design parameters and component properties can be evaluated. Trade-off analysis can be carried out by comparing the evaluated results of system design alternatives. In addition, 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, composability 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 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 2. In Section 3, we describe the main modules in some model libraries. The case study is presented in Section 4. Finally, Section 5 concludes this paper and discusses the future work.

2. 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 trade-off analysis, design space exploration and interactive simulations.

2.1 Hierarchy of System Models

We view a WSN as an application-oriented data-centric service provider. Sensors collaborate to deliver services to accomplish the network missions, fulfilling its requirements and optimizing its performance subject to platform and environment constraints. The hierarchy of system models in the MBSD framework is shown in Figure 1, aligned with their corresponding counterparts in the real world.

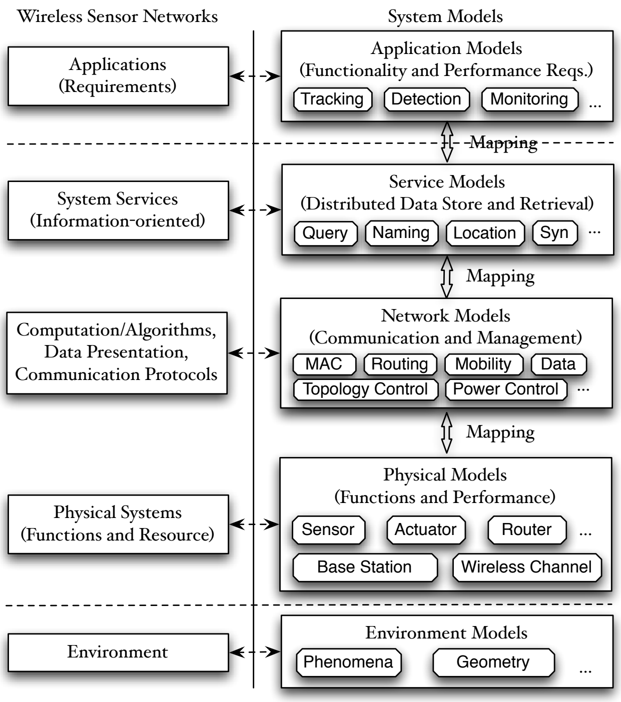

Wireless Sensor Networks
System Models
Applications (Requirements)
Application Models (Functionality and Performance Reqs.)
Tracking
Detection
Monitoring
...
Mapping
System Services (Information-oriented)
Service Models (Distributed Data Store and Retrieval)
Query
Naming
Location
Syn
...
Mapping
Computation/Algorithms, Data Presentation, Communication Protocols
Network Models (Communication and Management)
MAC
Routing
Mobility
Data
Topology Control
Power Control
...
Mapping
Physical Systems (Functions and Resource)
Physical Models (Functions and Performance)
Sensor
Actuator
Router
...
Base Station
Wireless Channel
Environment
Environment Models
Phenomena
Geometry
...

Figure 1: 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 (e.g., tracking, detection and monitoring), performance requirements (e.g., sampling rate, maximum delay, reliability and lifetime), physical system (e.g., sensors, routers, base stations and mobility) 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, the synchronization service, etc. The Service Model Library provides modules for these common services with interfaces to customize and compose components to fulfill the application requirements.

Network Model Library. This library is of paramount importance and 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. In order to enhance the reusability, components are modeled independently, and they can interact with each other only by exchanging events and data through ports. By well defining these ports, 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.

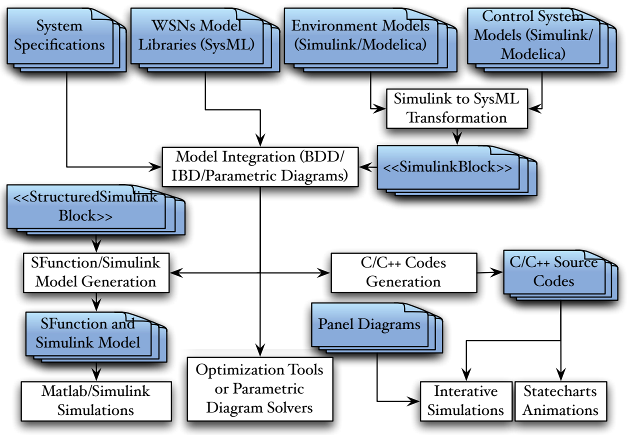
Physical System Model Library. This library is composed of modules for various physical platforms in heterogeneous WSNs. Despite their different capacities and computation powers, these platforms can be viewed to be composed of at most four parts: CPU, sensor, transceiver and battery (power source). 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 (e.g., indoor attenuations and urban attenuations), channel fading models (e.g., Gaussian fading and Rayleigh fading) and bit error rates under different modulation schemes (e.g., BPSK and QPSK).

Environment Model Library. This library serves as the bridge between the continuous-time domain and the event-triggered time domain. On one hand, physical environments usually exhibit continuous dynamic behaviors. For example, temperatures, speeds and air pressures are usually described by differential equations according to their corresponding physical laws. 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.

The proposed MBSD framework is developed using IBM Rational Rhapsody 7.6 [7]. All event-triggered-time components are modeled in SysML using the Rational Harmony for Systems Engineering profile (`HarmonySE.sbs`) and the SysML profile (`SysML.sbs`). Their behaviors are modeled by statecharts and primitive operations that are implemented in C/C++. All continuous-time components are modeld in Simulink or Modelica, which are then integrated with SysML using the Simulink profile (`Simulink.sbs`). Continuous data are passed through flow ports, while events and discrete data are exchanged via rapid ports.

2.2 System Design Flow

Based on the MBSD framework, we propose a system design flow (Figure 2) to compose both event-driven-time and continuous-time modules, with support for trade-off analysis, design space exploration and interactive simulations.



System Specifications
WSNs Model Libraries (SysML)
Environment Models (Simulink/Modelica)
Control System Models (Simulink/ Modelica)
Simulink to SysML Transformation
Model Integration (BDD/ IBD/Parametric Diagrams)
<<SimulinkBlock>>
<<StructuredSimulink Block>>
SFunction/Simulink Model Generation
C/C++ Codes Generation
C/C++ Source Codes
SFunction and Simulink Model
Panel Diagrams
Optimization Tools or Parametric Diagram Solvers
Matlab/Simulink Simulations
Interative Simulations
Statecharts Animations

Figure 2: 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.

Trade-off Analysis and Design Space Exploration. Each component can be characterized with quantities that specify its performance (e.g., delay, reliability, accuracy, response time and energy consumption), design parameters (e.g., sleep interval, transmission power and back off period) and component properties (e.g., cost and weight). The performance can be empirical results or statistical expectations based on mathematical models. Its relationship to the component properties and design parameters of itself and its subcomponents can be specified in Parametric Diagrams, which can be solved by Parametric Diagram solvers.

Rational Rhapsody provides the Parametric Constraint Evaluator (PCE) in the PCE profile (`PCE.sbs`), which evaluates Parametric diagrams using a computer algebra system such as MAXIMA or MATLAB Symbolic Math Toolbox to solve the mathematical expressions. The analytic results can be fed back to the design specification by instructing PCE to update the model. Due to the reusability and flexibility of the MBSD framework, design alternatives can be developed in a plug-and-play fashion, and their Parametric Diagrams potentially need only slight modifications. By evaluating these Parametric Diagrams using PCE, trade-off analysis can be carried out between these design alternatives.

In addition, Parametric Diagrams can be imported to multi-criteria optimization tools, such as the IBM ILOG CPLEX Optimizer, to calculate the optimal values for design parameters. These tools can also replace PCE for efficient trade-off analysis for complex hybrid systems.

Simulate in Matlab/Simulink. The Simulink profile offers the «`StructuredSimulinkBlock`» stereotype that can be applied to a SysML block to create a Simulink simulation environment. Other SysML blocks and Simulink blocks can be contained as subcomponents in this Simulink *structure* block, and the compositions can be specified in IBDs. From this Simulink structure block, we can generate a Simulink source file (`xxx.mdl`) in which all SysML blocks in 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. While models are running, we can observe how they react to events and transit their states accordingly in the animated statechart diagrams in quasi-real time. In addition, we can also inject events manually to simulate exceptions. *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. These are done graphically and no programming is needed. This simulation method is useful for system debug and validation.

3. 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 describe the main modules for physical platforms, the MAC layer, wireless channels and physical environments in detail.

3.1 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.

3.1.1 Physical Platforms

A physical platform is composed of four parts: battery, CPU, sensor and transceiver. A typical composition of a physical sensor platform is shown in Figure 3. During the initiation stage, each mote needs to register itself to the wireless channel component through the `pRegMote` port, with its ID, physical position and transmission power/range. Mote mobility is supported by updating its physical position periodically using `updatePhyPosition()`, the frequency of which is given by `sampleRate`. The `pSetTxPower` port enables upper layers to set the transmission power/range of its transceiver, which is required by many power-adaptive or cross-layer protocols. When a packet has been sent by the transceiver (an event received from `pChSendDone`), it will notify its MAC layer through `pMACSendDone`. Other ports are implemented by its internal subcomponents.

Each subcomponent has the `cost` and `weight` attributes which specify its cost and weight for design optimization and trade-off analysis. Furthermore, every component, including the physical sensor platform itself, is modeled as an active object in Rational Rhapsody, which means that it has its own thread of control and event queue, through which it processes its incoming events.

Battery. This component models the power supplies of sensor motes, wireless routers and base stations. Its capacity is specified by the `capacity` attribute. The amounts of energy consumed by the sensor, CPU and transceiver are received through `pConsumption`, which are used to update its capacity. If the remaining capacity falls below the threshold defined by the `threshold` attribute, it means the sensor mote has run out of battery and thus an event `evDead` is broadcasted to other components through `pDead`. The physical sensor platform will forward this event to components in upper layers. The `isInfinite` attribute can specify if a stable external power supply is available. If its value is `true` (e.g., for wireless routers and base stations), the `evDead` event will never be generated.

CPU. This component models the control logic unit of physical platforms. Its power levels in the active mode and sleep mode are specified by `activePower` and `sleepPower`, respectively. Although several other modes with different power levels are possible (e.g., the standby mode and different working frequencies), we only consider these two main modes, because the differences between energy consumptions in these modes and in the active mode are trivial. In addition, it is quite complex (if not impossible) to model the transitions between them, and we do not want to dive into too much details in the hardware level.

The CPU component implements the `pMACMoteCtrl` port, which enables upper layers to control the power states of the physical sensor platform. An `evMACMoteCtrl` event sent by upper layers can specify the component to be controlled (i.e., sensor, radio or mote) and its new power state (i.e., on or off). When this event is received, the CPU component transits its state accordingly and forward this event to the sensor and transceiver through `pMoteCtrl`. The CPU component will transit to the sleep mode if and only if the `evMACMoteCtrl` event from upper layers indicates that the *whole* sensor mote should transit to the low power mode. Even if both the sensor and transceiver have transited to the sleep mode, the CPU still need to stay in the the active mode to execute the algorithms and protocols in upper layers.

Periodically the CPU sends an `evConsumption` event via `pConsumption` to the battery component with the energy consumption in this period, the frequency of which is given by `batUpdateRate`. If an `evDead` event is received from `pDead`, it will transit to the termination state.

Sensor. Each sensor has three power states, whose power levels are given by `sleepPower`, `standbyPower` and `activePower`, respectively. Similarly, it sends the energy consumption information to the battery component periodically and transits to the termination state if an `evDead` event is received. If an `evMoteCtrl` message is received from the `pMoteCtrl` port and it is the target, it will transit its power state accordingly.

Each sensor can provide a certain type of service, which is identified by the `serviceType` attribute, to get the information about the phenomenon in the environment. Its quality of service is specified by `maxMeasurableValue`, `minMeasurableValue`, `accuracy`, `responseTime` and `threshold` for component selection and trade-off analysis. When the sensor is in the standby mode, it samples the environment periodically by sending a query `evEnvDataQ` to the environment component through the `pEnvDataQ` port, the frequency of which is specified by the `sampleRate` attribute. When the reply `evEnvDataR` that contains the phenomenon information is received via the `pEnvDataR` port, it will compare the value of the information with its threshold. If this value exceeds its threshold, it will generate a measurement that is sent to upper layers through the the `pSensorOutput` port. In addition, the sensor transits to the active mode, and the energy consumption for processing the measurement is sent to the battery component. This consumption needs to be sent separately because the time for the sensor to stay in the active mode is quite short, compared to the value of `batUpdateRate`.

A typical commercial sensor mote usually contains several different types of sensors. For instance, the ITS400 sensor board for iMote2 [3] provides accelerometer, temperature, humidity and light sensors. This can be modeled by specifying the multiplicity of the sensor component in a physical sensor platform. Each sensor component will process the messages from upper layers and the environment component based on its `serviceType` attribute.

Transceiver. 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. Similarly, 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 through `pMACCCAQ`, which requires to poll the channel to get the CCA information, 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 through `pMACSend`, 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.

3.1.2 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. To fulfill these functions, the ports used to interact with the physical sensor platform are also provided. The basic MAC layer component defines three parameters for trade-off analysis and optimization: `delay`, `energy` and `reliability`. They are statistical or empirical estimations defining the expected delay, energy consumption and reliability to send a single packet with a pre-defined size using this MAC protocol. Many works can be used to model them for different MAC protocols (e.g., [15]), which is out of the scope of this paper.

The main subcomponents in this layer are described as follows, some of which are developed based on [10]. A composition example is shown in Figure 4.

Low Power Listener (LPL). This component is responsible to adjust the transceiver's power state based on channel activity. Two types of LPL listener are provided in the MBSD framework. The *fixed* LPL listener sleeps for a fixed interval and then polls the channel. If channel activity is detected, it activates the transceiver and stop polling the channel. After the transceiver becomes free, it starts a timeout alarm. If no more transceiver activity is detected before the alarm fires, the transceiver is powered down and channel polling is enabled again. The *periodic* LPL listener instead polls the channel all the time, and immediately transits the transceiver into the sleep state when the channel is free. The duty cycle can be defined by `sleepInterval`, `dutyCycle`, `dutyToSleep` and `sleepToDuty`. More details are available in [10].

CSMA/CA Channel Access. This component is responsible to gain the channel access right for a transmission using the CSMA/CA mechanism. If a query is received from `pCSMACAQ`, it starts to poll the channel by sending a request via `pMACCCAQ`, whose reply is received from `pMACCCAR`. If the channel is busy, it waits for a backoff period that is randomly selected from the backoff window and then tries to poll the channel again. This process is repeated until the maximum number of backoff times has been reached, or the channel is polled to be free for certain times that is specified by the contention window size. The result (`false` or `true`, respectively) is returned through `pCSMACAR`. In slotted CSMA/CA mechanism, the backoff period boundaries should be aligned with the slot boundaries.

CSMA/CA Sender. This component is responsible to send a packet using the CSMA/CA mechanism. If a request is received from `pCtrlSend`, it sends a query to to `pCSMACAQ`. If the channel access right is gained successfully (`true` is returned via `pCSMACAR`), it forwards this packet through `pMACSend` for transmission. Otherwise, it replies `false` via `pCtrlSendDone`. In slotted CSMA/CA, a transmission is started only at the slot boundaries.

Slot Manager. This component manages the slot schedule for TDMA mechanism. The size of each slot is specified by `slotPeriod`. A slot can be marked for transmission, listening or sleep via `pMarkSlot`, whose confirmation is returned via `pMarkSlotDone`. Other components (e.g., the TDMA Sender) can query the slot statuses via `pSlotQ`, whose results are returned via `pSlotR`. Since all nodes share the clock on the host machine in a simulation, synchronization is achieved trivially and slot boundaries can be calculated easily.

TDMA Sender. This component is similar to the CSMA/CA Sender, except that packets are sent using the TDMA mechanism. For each packet, it sends a query to `pSlotQ` to get the slots scheduled for transmission. When the result is returned from `pSlotR`, it starts to send in those slots.

Receiver. This component is responsible to broadcast packets received via `pMACRcv` to the MAC Controller and other protocol-specific components through `pCtrlRcv` for further process.

Queue Manager. This component is responsible to buffer packets in the MAC layer. When a request is received via `pEnMACQueue`, it checks its queue size. If the queue is full, this request is ignored and a reply with `false` is returned via `pEnMACQueueDone`. Otherwise, the packet is added to the queue and a reply with `true` is returned. When a request is received via `pQueueRetrieve`, the first packet in the queue is returned via `pQueueReturn`.

MAC Controller. 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 components have been defined, including components in upper layers, the queue manager, MAC CSMA/CA and TDMA senders, CSMA/CA channel access, slot manager and the receiver components.

3.1.3 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.

The basic features of channel components are specified by `channelFrequency`, `noisePower` and `bandwidth`. If the simple UDG model is used, the `isUDG_Model` attribute should be set to be `true` and the ratio between the interference range and transmission range should be specified by `ratio4UDG`.

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 uses two FIFO queues to buffer the received requests: `CCAQQueue` for channel polling requests that are received from `pCCAQ`, and `sendQueue` for packet sending requests that are received from `pChSend`.

In addition, the channel component maintains two lists. The `moteList` contains the information of all nodes in the network, including their IDs, physical positions and transmission powers/ranges. During the initiation stage of the system, each node in the network will register itself with this information to the channel component via `pRegMote`. This list can be updated using the information contained in the received packets (e.g., beacon packets) that will be checked before entering `sendQueue`. The `channelTXList` contains 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 that are calculated based on the packet sizes and the channel bandwidth. All finished transmissions will be removed from this list by examining it periodically, the frequency of which is given by `updateRate`. This list is essential for the evaluation of CCAs and BERs.

The channel component should model different radio propagation models, which are empirical mathematical formulations for the characterization of radio wave propagation as a function of frequency, distance and other conditions. Every channel model should extend the abstract channel component and implement the `getRSS()` operation that calculate the RSS at one node for a certain signal. For example, the ITU Indoor Propagation Model [9] that estimates the path loss inside a room or a closed area inside a building delimited by walls of any form is formally expressed as follows:

$$Loss_{dB} = 20\log f + N\log d + P_f(n) - 28$$

where f is the signal frequency (unit: MHz), d is the distance (unit: meter), n is the number of floors between the sender and the receiver, and N and $P_f(n)$ are the distance power loss coefficient and floor loss penetration factor that are defined in [9], respectively. In `getRSS()`, $Loss_{dB}$ is computed and subtracted from the transmitted signal strength.

Furthermore, in order to estimate the BER, various channel fading models and modulation schemes should be modeled as well. Every channel model should implement the `getBER()` operation inherited from the abstract channel model. In our case study, we consider the Rayleigh fading [14] that is a useful model when there is no dominant propagation along a line of sight between the transmitter and receiver, because it is viewed as a reasonable model for tropospheric and ionospheric signal propagation as well as the effect of heavily built-up urban environments on radio signals. If BPSK modulation scheme is used, the BER can be calculated as:

$$ber = \frac{1}{2}\left(1 - \sqrt{\frac{E_b/N_0}{E_b/N_0 + 1}}\right)$$

where N_b is the transmitted signal strength, and N_0 is the summation of the noise power and interference from other ongoing transmissions in the channel.

If `sendQueue` is not empty, the channel component will process the first packet in the queue by calculating the RSS and packet loss probability (PER) for the destination. Then it will forward/broadcast this packet with the RSS and PER via `pChReceive`, or discard this packet according to its PER. Meanwhile, a new entry is inserted to `channelTXList`, and a notification is sent to the sender via `pChSendDone`. The CCA is estimated by iterating `channelTXList` to get the strength of the strongest signal at the node sending this CCA query, and the value is returned via `pCCAR`.

3.2 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. Meanwhile, different sensors can have different measurements of the same phenomenon based on their positions and the phenomenon propagation model. The main components in this library are described as follows.

Environment. This component models the propagation of the information that are received from the phenomenon component via `pPhnmData`. It accepts `evEnvDataQ` queries from sensors via `pEnvDataQ`, computes the phenomenon values based on the propagation model and the distances between sensors and the phenomenon, and returns the results to sensors via `pEnvDataR`. The 3D uniform propagation model and UDG model are supported. Several environment components can be composed in a network instance for different environment regions. For example, two such components are included in our case study with one for each room.

Phenomenon. This component serves as the interface between the continuous-time domain and the event-driven-time domain. The basic phenomenon component periodically calls the `updatePhnmData()` operation and sends the new information to its environment component via `pPhnmData`, the frequency of which is specified by `sampleRate`. Every phenomenon component should extend this abstract component by implementing `updatePhnmData()` that prepares the phenomenon information in the format required by the environment component. In addition, they also need to define the flow ports through which the outputs of the numerical solvers for differential equations can be received.

Modeling continuous behaviors. The continuous-time dynamic behaviors are actually modeled in equation-based modeling languages/tools. The outputs of their solvers are forwarded to SysML models through flow ports. The MBSD framework currently supports Matlab/Simulink and Modelica. Simulink models should be first built using the Embedded Coder in Matlab to generate C/C++ source codes, and then imported as SysML blocks to Rational Rhapsody by applying the «`SimulinkBlock`» stereotype provided in the Simulink profile. The input and output flow ports of the imported SysML blocks are generated automatically. Modelica models need to be transformed to Simulink models first. This function is available in many Modelica development tools, such as Dymola [4].

4. CASE STUDY

The MBSD framework proposed in this work is intended to provide a reusable and extensible mechanism for system design optimization, trade-off analysis, validation/verification 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. Due to the limited space, only the simulations in Matlab/Simulink are provided here.

4.1 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 safe condition inside the building. 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 for people and the data center room for computer systems. 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 remove the heat and 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, depending on the current room temperatures. One temperature sensor mote 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 [8] 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.

4.1.1 Physical Platforms

The temperature sensor motes, base station and PAN coordinator can be composed using components from the physical system model library. The internal composition of a temperature sensor mote is shown in Figure 3. The base station and PAN coordinator can be composed in the similar way but without the sensor component. The attributes of each component must be set to reflect its properties.

4.1.2 IEEE 802.15.4 MAC Protocol

The IEEE 802.15.4 standard defines a non-beacon contention-based channel access mode. When a reduced-function device (RFD) wants to send a packet, it simply transmits this packet using the unslotted CSMA/CA mechanism to the PAN coordinator. When the coordinator wants to send packets to a RFD, it stores the packets first and waits for that RFD to contact and request data. A RFD may make contact by transmitting a MAC command requesting the data to the coordinator using the unslotted CSMA/CA mechanism as well, at *an application-defined* rate. When the coordinator receives this request, it will send back an acknowledgement first. If packets are pending for this RFD, the coordinator starts to transmit these packets. Otherwise, a data packet with zero-length payload is sent back to this RFD to indicate that no packet is pending.

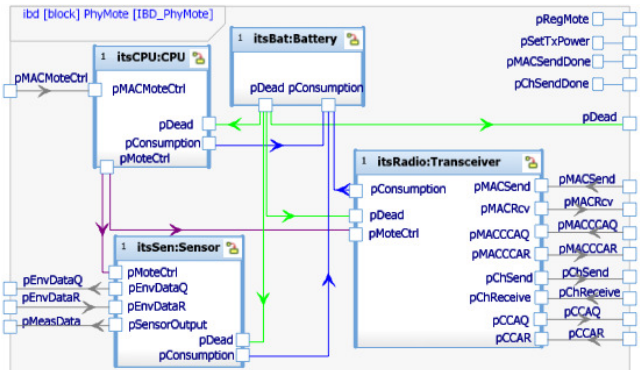


Figure 3: Internal Composition of Physical Sensor Motes

The 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 Figure 4. Both the sensor motes 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.

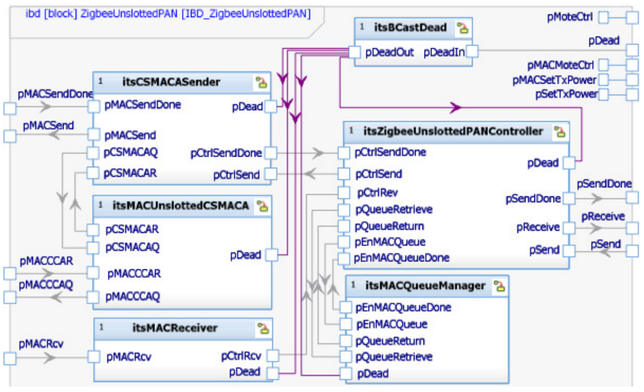


Figure 4: Internal Composition of the IEEE 802.15.4 Unslotted CSMA/CA Mode for the PAN Coordinator

4.1.3 Building Thermal Model in Simulink

The control system and building thermal dynamics are modeled in Simulink as shown in Figure 5.

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} <= 20 and turn off if T_{LR} >= 24
- AC: turn on if T_{DC} >= 24 and turn off if T_{DC} <= 20

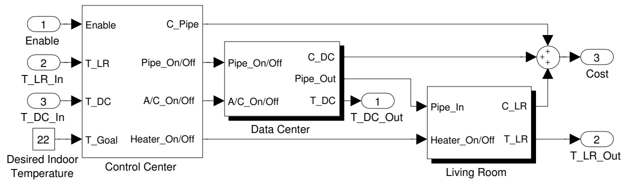


Figure 5: Simulink Model for the Thermal Control Block

- Pipe: turn on if $T_{LR} <= 22$ and turn off if $T_{LR} >= 24$

where T_{LR} and T_{DC} are the measured temperatures of the living room and data center, respectively. Here, a variance of 2 °C around the desired temperature is allowed to avoid oscillations. The pipe is turned on when $T_{LR} <= 22$ to reduce the usage of the heater.

The thermal models for the living room and data center are developed based on the Thermal Model of a House provided as a demo in Simulink. It is modified by considering the effects of the pipe, AC and heat transfer efficiency. The details are omitted due to the limited space.

4.1.4 Overall System Composition

After all required components have been composed using the model libraries, they can be connected together to model the whole application. The overall composition of the building thermal control system is shown in Figure 6.

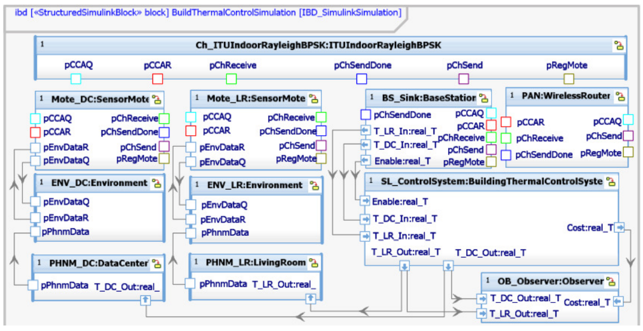


Figure 6: Internal Composition of the Building Thermal Control System

The channel component with the ITU indoor propagation model, Rayleigh fading and BPSK modulation scheme is selected for this system, because the two sensors are deployed in different rooms, which means they are blocked by walls and thus with no line-of-sight between them. 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 building thermal dynamics model in Simulink by applying the «`SimulinkBlock`» stereotype. The `Observer` component stores the outputs of the Simulink solver, which are used for debug and validation in interactive simulations.

4.2 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 the simulations:

Wireless + No Pipe. The room temperatures are collected to the base station using the WSN, but the pipe is never turned on. Each sensor wakes up to measure once every 5 seconds.

Wired + Pipe. The room temperatures are fed back to the control center directly, and the pipe feature is enabled. This scenario is totally modeled in Simulink and used as the reference to study the impacts of the delays in the WSN.

Wireless + Pipe ($5s$). The room temperatures are collected to the base station using the WSN, and the pipe feature is enabled. Each sensor wakes up to measure the temperature once every 5 seconds.

Wireless + Pipe ($60s$). This scenario is quite similar to the above one, except the sensor sleep interval is 60 seconds.

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. Other parameters used in the Simulink model are the same as those used in the demo of Thermal Model of a House. The MAC protocol uses the default parameter values specified in [8].

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.

5. CONCLUSIONS AND FUTURE WORK

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 trade-off analysis, design space exploration and interactive simulations. SysML, Simulink and Modelica that are standard modeling languages in the industry are used to develop the model libraries, by taking advantage of the powers of each language. The main component mod-

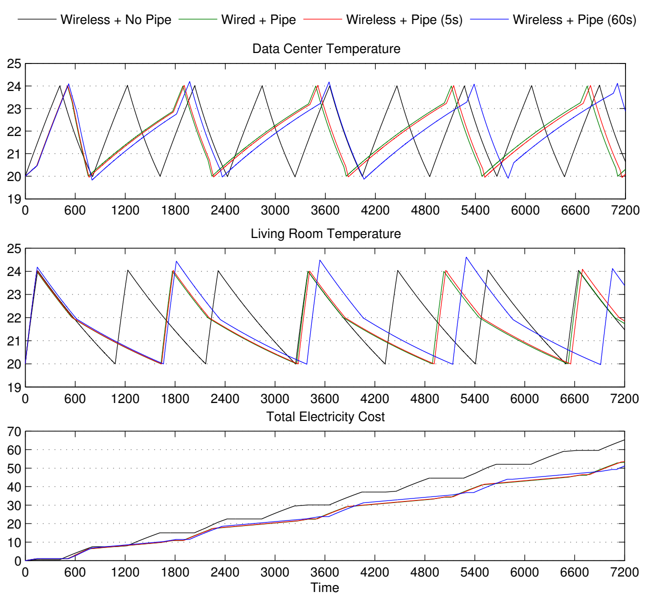


Figure 7: Room Temperatures and Total Electricity Cost

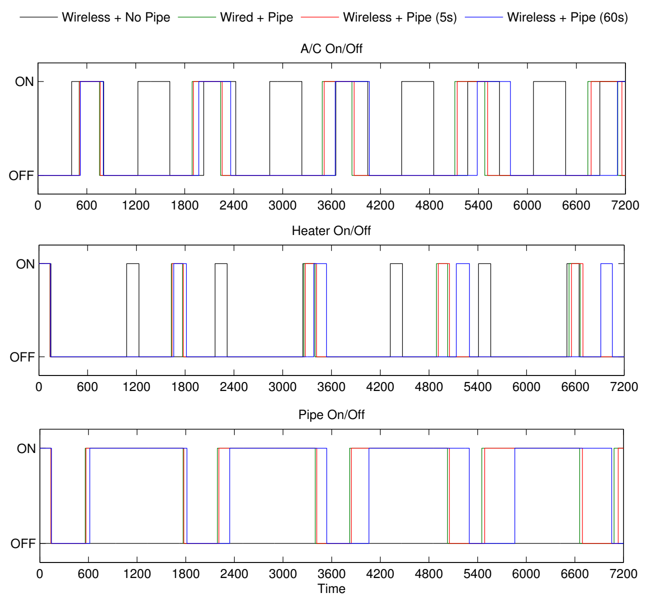


Figure 8: Working Statuses of the A/C, Heater and Pipe

els for physical platforms, the MAC layer, wireless channels and physical environments are described in detail. A hybrid system is used as the case study to demonstrate the composability, reusability and flexibility of the MBSD framework.

The MBSD framework can be used as a modeling hub to integrate other modeling tools and languages. The future work can follow these two directions. Firstly, multi-criteria optimization tools can be integrated into the MBSD framework, because the current Parametric Diagram solver can not solve complex optimization problems efficiently, and the trade-off analysis should be carried out in a more automatic fashion. The IBM ILOG CPLEX Optimizer is good candidate for this purpose. Secondly, the model libraries for the application and service layers should be revised to improve their reusability and flexibility. The accuracy of the models described in Section 3 should also be verified by comparing with a standard simulator for WSNs.

6. REFERENCES

[1] P. Boonma and J. Suzuki. Moppet: A model-driven performance engineering framework for wireless sensor networks. *The Computer Journal*, 53(10):1674–1690, 2010.

[2] 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.

[3] Crossbow. ITS400 Data Sheet. *www.xbow.com:81/Products/productdetails.aspx?sid=261*.

[4] Dassault Systemes. Dymola. *www.dynasim.se/*.

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

[6] 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.

[7] IBM. IBM Rational Rhapsody Help. *www.ibm.com/software/awdtools/rhapsody/*.

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

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

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

[11] 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.

[12] 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.

[13] 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.

[14] B. Sklar. Rayleigh fading channels in mobile digital communication systems part i: Characterization. *IEEE Communications Magazine*, 35(7):90–100, July 1997.

[15] B. Wang and J. S. Baras. Performance analysis of time-critical peer-to-peer communications in IEEE 802.15.4 networks. *ICC*, pages 1–6, 2011.