Softw Syst Model (2017) 16:949–977
DOI 10.1007/s10270-015-0498-5

REGULAR PAPER

An approach based on the domain perspective to develop WSAN applications

Taniro Rodrigues[1] · **Flávia C. Delicato**[2] · **Thais Batista**[1] · **Paulo F. Pires**[2] · **Luci Pirmez**[2]

Received: 3 November 2014 / Revised: 30 July 2015 / Accepted: 26 August 2015 / Published online: 25 September 2015

Abstract As wireless sensor and actuator networks (WSANs) can be used in many different domains, WSAN applications have to be built from two viewpoints: domain and network. These different viewpoints create a gap between the abstractions handled by the application developers, namely the domain and network experts. Furthermore, there is a coupling between the application logic and the underlying sensor platform, which results in platform-dependent projects and source codes difficult to maintain, modify, and reuse. Consequently, the process of developing an application becomes cumbersome. In this paper, we propose a model-driven architecture (MDA) approach for WSAN application development. Our approach aims to facilitate the task of the developers by: (1) enabling application design through high abstraction level models; (2) providing a specific methodology for developing WSAN applications; and (3) offering an MDA infrastructure composed of PIM, PSM, and transformation programs to support this process. Our approach allows the direct contribution of domain experts in the development of WSAN applications, without requiring specific knowledge of programming WSAN platforms. In addition, it allows network experts to focus on the specific characteristics of their area of expertise without the need of knowing each specific application domain.

Keywords WSAN applications · Model-driven architecture · Domain-specific language · UML profile · Architecture · Code generation · Abstraction

Communicated by Prof. Franck Barbier.

✉ Taniro Rodrigues
tanirocr@gmail.com

Flávia C. Delicato
fdelicato@gmail.com

Thais Batista
thaisbatista@gmail.com

Paulo F. Pires
paulo.f.pires@gmail.com

Luci Pirmez
luci.pirmez@gmail.com

1 Universidade Federal do Rio Grande do Norte, Campus Universitário, Natal, RN 59078-970, Brazil

2 Universidade Federal do Rio de Janeiro, Cidade Universitária, Rio de Janeiro, RJ 21941-901, Brazil

1 Introduction

Wireless sensor and actuator network (WSAN) is a promising research field with potential application in different domains, such as military, environmental, industrial, security, and health [43]. A WSAN consists of small, wireless, spatially distributed, and often battery-powered devices equipped with a radio transceiver, sensors, actuators, and a micro-controller. The WSAN systems are capable of observing the physical world to obtain useful information from it. Moreover, these systems can process the obtained data, make decisions, and perform specific operations in the monitored environment, such as turn on devices (as lamps or fans), through actuators installed on the nodes. WSAN systems can be considered as a specialization of *distributed, real-time, and embedded* (DRE) systems. As such, WSANs are characterized by a high heterogeneity regarding application requirements, radio technology, node capabilities, and network protocols.

There are two main sources of complexity of WSAN application: (1) a lack of adequate abstractions in application development and (2) a lack of tool chains for application development [42]. Programming for this type of network is very hard because it requires developers to know the available sensor platforms specificities, increasing the application's development learning curve. Moreover, the WSAN domain has no optimal or recommended development methodology, thus leading developers to use an extreme programming (code-and-fix) approach where, sometimes, the written code has the domain knowledge tied with the platform implementation. Nowadays, there are several different platforms that support the WSAN application development and execution, each one having its own requirements, execution and programming environments, and software tools. Currently, application developers need to know several WSAN-specific characteristics to build applications through the use of low-level abstractions provided by the sensor node's operating system (OS). Thus, WSAN applications are generally developed from two viewpoints: The first is the *domain expert's* (biologists, engineers, geologists, among others) *viewpoint* and the second is the *network expert's viewpoint*. These different viewpoints create a gap between the abstractions handled by such types of developers. Thus, the process of developing a WSAN application becomes harder than it should be. Furthermore, the high coupling between the domain rules and the underlying sensor platform, combined with the lack of a development methodology to support the application lifecycle, result in platform-dependent projects and source code difficult to maintain, modify, and reuse.

We argue that a solution to increase the abstraction level when developing WSAN applications is by adopting an approach that facilitates the communication among developers, while promoting a clear and synergetic separation between the specification of requirements at the domain level and the specification of such requirements in a given wireless sensor platform. In our solution, we propose adopting a software development process based on the creation of *unified modeling language* (UML) [31] models through a *model-driven architecture* (MDA) [27] approach. The UML is OMG's (*Object Management Group*) specification to model an application's structure, behavior, and architecture.

The development of systems using the MDA [30] approach allows specifying systems at a high abstraction level and subsequently applying automatic transformations (also called mappings) from this specification to generate code for a specific computational platform. In MDA, models are divided into three views [26]: (1) CIM (*computation-independent model*) is a view of a system from the computation-independent viewpoint that is responsible for providing system requirements without defining computational aspects, (2) PIM (*platform-independent model*) is a view of a system from the platform-independent viewpoint responsible for describing the application behavior without considering details of a specific platform, and (3) PSM (*platform-specific model*) is a view of a system from the platform-specific viewpoint that is responsible for describing the application/system using concepts and features of a specific target platform. A *transformation* consists of a script written in a transformation language that has as input an instance of a model (and its associated meta-model) and as output a new model. MDA defines two types of transformations: M2M (*model to model*) and M2T (*model to text*). The first one is responsible for transformations between models, for example, a transformation from a PIM to a PSM. The second one is responsible for transformations between model and text, for example, from PSM to source code.

The MDA approach offers different options for the definition of the PIM and PSM meta-models, such as DSLs and UML profiles. The choice of UML as a meta-language for specifying our meta-models leverages the use of models already existing in the UML definition as a basis for the application's model specification, thus decreasing the learning curve of developers to model applications. Features that are not natively supported by UML have to be included in the meta-models through a set of stereotypes present in a UML profile. A UML profile is an extension mechanism that provides a way to customize models for a specific domain. A profile is composed of classes, stereotypes, data types, primitive types, and/or enumerations that modify the UML's meta-classes (for example, Class, Component, and more) so as to add properties that are needed to specify a target domain. The goal of using UML profiles is to allow application developers to include in the models information about the domain knowledge through the properties present in the stereotypes. Thus, a UML profile can be reused numerous times to model applications of a target domain. The use of UML profile as the extension mechanism has two major advantages: (1) Properties specified in the profiles do not affect the properties of standard UML models and (2) decrease the development effort of the MDA approach, as it is possible to inherit the basic modeling components from the UML specification.

In order to fully achieve the benefits of the MDA approach, it is imperative to tailor the different software artifacts involved in such approach to a target domain. Moreover, as the MDA specification [26] does not encompass a software development process, the development team is in charge of defining a customized process to organize the activities needed to correctly handle the MDA software artifacts used to build the applications. In this context, our goal is to provide such a customization of the MDA approach for the WSAN domain. The proposed MDA infrastructure, named *ArchWiSeN* (Architecture for Wireless Sensor and Actuator Networks), encompasses PIM, PSM and transformations, and an associated process to deal with the aforementioned issues to develop WSAN applications. The application domain knowledge is represented at the PIM level using UML's class and activity diagrams enhanced by a UML profile. Such WSAN profile was designed to add specific properties from the WSAN domain. As the PIM describes the semantics of the elements necessary to build WSAN applications regardless the implementation platform, the knowledge representing different sensor and actuator platforms is specified at the PSM level. Consequently, the proposed MDA infrastructure is able to encompass different PSM metamodels for the existing WSAN platforms. As every WSAN platform has its own features, their meta-models have to be represented differently. For example, our PSM for the TinyOS [18] platform is composed of UML's component and state machine diagrams enhanced with a UML profile, while our PSM for the Sun SPOT [32] platform is composed of UML's component and class diagrams enhanced with another UML profile.

In this work, we claim that the adoption of the high-level abstract models provided by an MDA approach applied to the WSAN domain, along with the use of the proposed application development process, offers an enhanced development environment where (1) each developer executes only his/her independent task related to his/her own expertise field; (2) the communication between developers is facilitated, thus improving their productivity; (3) the developers can interact using high-level models that are easier to understand than platform-specific code; and (4) all previously modeled information can be easily reused for modifying an existing application or creating a brand new application.

The use of an MDA approach, in general, brings classical benefits regarding reuse and maintainability of software, but it is necessary to verify that such benefits are kept in the context of WSAN applications and if so, what are the benefits in terms of development effort. To perform such validation, we evaluated our approach with (1) a proof-of-concept using a number of different scenarios in order to demonstrate the functionality and use of ArchWiSeN in practice, and (2) the execution of a controlled experiment with real users to develop WSAN applications with and without the use of ArchWiSeN in order to test the approach in terms of productivity, understandability, and reuse.

The main goals of this work are: (1) to define a process to create all MDA artifacts encompassed in the ArchWiSeN infrastructure; (2) to develop the MDA artifacts (PIM, PSM, and transformations) that compose ArchWiSeN; (3) to define a process to build WSAN applications with the presented approach using the developed MDA artifacts, where developers benefit from better productivity, comprehension, separation of concerns, and reuse.

The remainder of this paper is organized as follows. Section 2 presents an approach overview, including the development process and all necessary MDA artifacts and the process to define a WSAN application using this approach. Section 3 presents the evaluation performed with a controlled experiment and a proof-of-concept. Finally, Sect. 4 presents the related work and Sect. 5 concludes the paper.

2 Approach overview

This section presents the processes of building and using ArchWiSeN from the domain engineering (ArchWiSeN development) to the application engineering (WSAN application development). Considering the context of building an MDA infrastructure, the domain engineering process concerns the creation of ArchWiSeN's MDA artifacts such as UML profiles and transformations. The application engineering process concerns the definition of a process to guide WSAN application developers (domain and network experts) through the process of creating a concrete application using ArchWiSeN's provided artifacts.

The domain engineering process [8], composed of the design and implementation phases, is responsible for the development of reusable artifacts such as components, transformations, DSLs (*domain-specific languages*), and user documentation needed to build an application. In such process, we emphasize the design and implementation phases focused on building a reusable software infrastructure, instead of developing a single application, as in traditional application development. The domain design phase includes developing a base infrastructure on which WSAN applications can be properly specified. It also allows planning how individual systems can be created from reusable software artifacts. The domain implementation phase involves the development of reusable artifacts, as UML profiles and transformations.

The application engineering is the process responsible for the development of concrete applications from reusable software artifacts created through the domain engineering. The application engineering process begins with requirements elicitation, goes through the analysis and specification phases, and ends with the creation of the source code.

The domain and application engineering processes promote two different levels of reuse. The first level concerns the MDA infrastructure by itself. ArchWiSeN encompasses meta-models and transformations that can be used multiple times during the application engineering processes. Additionally, and regardless the application domain, all the MDA artifacts are able to be extended since they use standardized modeling language (UML) and transformation languages (for both M2M and M2T). The second level of reuse regards the models specified when an application is developed using ArchWiSeN through the application engineering process. All models can be reused to extend an application or to create a new one using the previous models as starting points. Such reuse potentially speeds up the application development in comparison with the building of a brand new application from scratch.

2.1 Domain engineering

This section describes the software artifacts built to support the application engineering process for the development of WSAN applications using ArchWiSeN (Sect. 2.2). Besides supporting the development process of WSAN applications, the presented software artifacts are extensible to accommodate the inclusion of new UML profiles and/or model transformations whenever required, for instance, when a new sensor platform arises.

Figure 1 presents the domain engineering implementation phase. By using this process, developers (for sake of clarity, the developers mentioned in this subsection are the ArchWiSeN's developers) are able to add or to modify any required feature from the available software artifacts. The first activity, "Create Platform Independent UML profile," aims at specifying a UML profile that extends all UML elements needed to describe a WSAN application, at a platform-independent level (PIM). Next, the "Create Platform Specific UML Profile" activity aims at extending UML elements to define specific information for an existing sensor platform (PSM). Each PSM describes the characteristics needed to develop any application in a specific sensor platform. Such PSMs should be designed in a way they cover the majority of details needed to represent any application in the target platform. We have currently built PSM for the TinyOS platform. All UML profiles and models were developed using the Eclipse's Papyrus [14] plug-in (both PIM and PSM).

The "Create M2M Transformation Program" activity aims at transforming, through MDA model to model (M2M) transformations, an application model, developed at the PIM level, into a platform-specific application (PSM) model, having as base a target PSM meta-model. In this work, we built the transformation programs by using the *Atlas Transformation Language* (ATL) [29].

Finally, to enable the source code generation is necessary to perform the "Create M2T Transformation Program" activity. In this activity, templates (source code skeletons to each target platform) are developed. The M2T transformation program is able to generate a source code based on the developed templates from a platform-specific model. In our work, these templates were developed using the *Eclipse Acceleo plugin* [12]. Acceleo is a MOF model to text (M2T) transformation language (MOFM2T) implementation.

The described activities produce the ArchWiSeN's software artifacts. With these artifacts (PIM UML profile, PSMs UML profiles, M2M, and code templates for WSAN platforms), it is possible to apply the proposed MDA approach to build a WSAN application. The following subsections describe each software artifact developed through the domain engineering process.

The remainder of this Section is organized as follows: Sect. 2.1.1 presents the platform-independent meta-model defined for ArchWiSeN, Sect. 2.1.2 presents the platform-specific meta-models for two currently supported sensor network platforms, and Sect. 2.1.3 presents the transformations that allow the automatic generation of source code.

2.1.1 Platform-independent meta-model

We conducted a literature review on domain-specific languages (DSLs) and meta-models of different WSAN domains in order to understand how a PIM could be used to fulfill functional and non-functional requirements of a wide range of WSAN applications, while having a suitable abstraction level without including platform-level decisions. Our goal was to elucidate the properties needed to a meta-model where domain experts could describe their systems using only abstract concepts from the WSAN field. As outcome of such literature review, we highlight the works described in [5,10,17,24,39].

In [5], the authors use a UML profile for the specification of WSAN applications for TinyOS platform. The applications are organized in a structural view and a behavioral view through UML's component and sequence diagrams, respectively. Unfortunately, the numerous TinyOS platform properties present in [5] did no help to construct our domain model, but we found the presented division between the specification of structure and behavior valid for the definition of WSAN applications independently on the target platform. Such a division is important because it separates the physical organization of the network from the node's behavior, two viewpoints that are fundamental for the WSAN development process. Besides enabling the separation of concerns between infrastructure and behavior, separating such view-

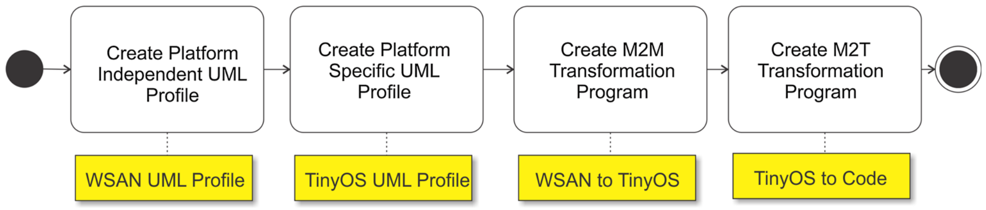
Create Platform Independent UML Profile
Create Platform Specific UML Profile
Create M2M Transformation Program
Create M2T Transformation Program
WSAN UML Profile
TinyOS UML Profile
WSAN to TinyOS
TinyOS to Code

Fig. 1 Domain engineering process

points presents advantages as the possibility to reuse the infrastructure models to define new application for the same infrastructure as well as the reuse of behavior models for implementing new requirements or new applications.

The works presented in [10] and [17] describe a *software product line* for the WSAN context able to represent important features of this domain, such as functional and non-functional requirements, for example, network protocols, network organization (flat, hierarchical), sensor capabilities, and more. Such characteristics are important to define the WSAN domain in a platform-independent way, and their concepts were used during the development of our PIM. From the DSL presented in [24], we were inspired to divide the WSAN organization in *regions* and *node groups*. Moreover, from the different views presented in [39] we adopted the division of the PIM model in a configuration view, allowing the definition of non-functional requirements.

The current platform-independent meta-model uses UML class and activity diagrams as a base to describe all the information needed to define an application. Our PIM uses UML profiles to fill all model elements with WSAN-specific properties. The goal was, through the designed profiles, allowing developers to include WSAN structural and behavioral characteristics in the models.

The platform-independent meta-model was developed considering (1) the different developers' viewpoints (domain and network); (2) the different abstraction levels (platform-independent or platform-dependent); and (3) the need of taking into account functional and non-functional properties during the application engineering process. Given the aforementioned PIM characteristics, and using the UML's ability to provide different views through its diagrams, it becomes possible to separate the modeling application into three different views: (1) structural, (2) behavioral, and (3) configuration. This strategy promotes separation of concerns and reuse of software artifacts.

We will use a scenario to detail each artifact present in MDA infrastructure. This scenario considers the development of a simple *Smart Home* application with the goal of automatic controlling the illumination and the temperature in a house. The application considers the creation of two smart environments inside a house, namely a lounge and a bedroom. Both environments are endowed with automatic light and temperature control. The environment control is done through actuators connected to the air conditioner and lamps. The application will turn the lights on when someone is detected inside a room while the temperature must be kept at 20 °C at all times.

As mentioned before, a UML class diagram is used to model the structural view of WSAN applications. To design such model, the developer must create a new class diagram and apply the WSAN profile (Fig. 4a for further details). Then, the developer should set the main package with the stereotype «system». The next step is to create sub-packages that represent «regions» and a stereotype that characterizes the nodes by physical proximity. Inside each region, it is necessary to create UML classes that represent «nodegroups», containing all the nodes in charge of performing a same task. The «nodegroup» stereotype stores information about the number of nodes that will perform a task and the hardware used by the entire group. A nodegroup has a definition property where the «nodedefinition» is specified. A «nodedefinition» incorporates information that specifies the infrastructure as the sensors, target operational system, actuators, and battery. There are «wirelesslinks» connecting «nodegroups» that represent the wireless communication between two «nodegroups» and the direction of that communication. Each one of the existing «nodegroups» has a behavior that is modeled through the behavioral view.

Figure 2 shows the structural view of our scenario application. In such a diagram, it is possible to see the existence of two regions, each one containing one «nodegroup» and one «nodedefinition». The «nodegroup» of the «region» *ServerRoom* is named *Room* and has a «nodedefinition» named *MyNodes*. In *MyNodes*, it is possible to see the devices that are connected to the nodes, two sensors: temperature and light. Finally, in the «region» *SinkRoom* the *Control* «nodegroup» uses a «nodeDefinition» named *SinkNode*.

For the behavioral view, developers use a UML activity diagram also enhanced by the WSAN profile (Fig. 4b for further details). Figure 3 shows the behavioral view model. In this Figure, it is possible to notice the presence of two swimlanes: *Room* and *Control*. The first swimlane models the behavior of the «nodegroup» *Room* (specified in Fig. 2), while the second swimlane models the behavior of the «nodegroup» *Control* (also specified in Fig. 2). The behavior of the «nodegroup» *Room* is quite simple. The nodes have to initialize its radios («radio»), start a 20 s timer («timer»), perform 10 readings of the temperature sensor («readSensor»), select only the higher value («aggregation»), and send («send») such a value through the radio. Finally, the behavior of the «nodegroup» *Control* is to initialize the radio («radio») and wait to receive messages («receive»).

The configuration view allows including, in both the UML's class and activity diagrams, specific characteristics such as routing protocol and network topology that directly affect non-functional requirements. It is important to notice that the properties present in the configuration view are related either to the application or to the network domain. Thus, such properties are not platform specific. Although such non-functional requirements can represent network features, they are not entangled with a specific platform, preserving the PIM properties. For example, the «system» stereotype from the class diagram can configure WSAN properties as routing protocol and topology, and the «radio» stereotype from the activity diagram can configure the radio

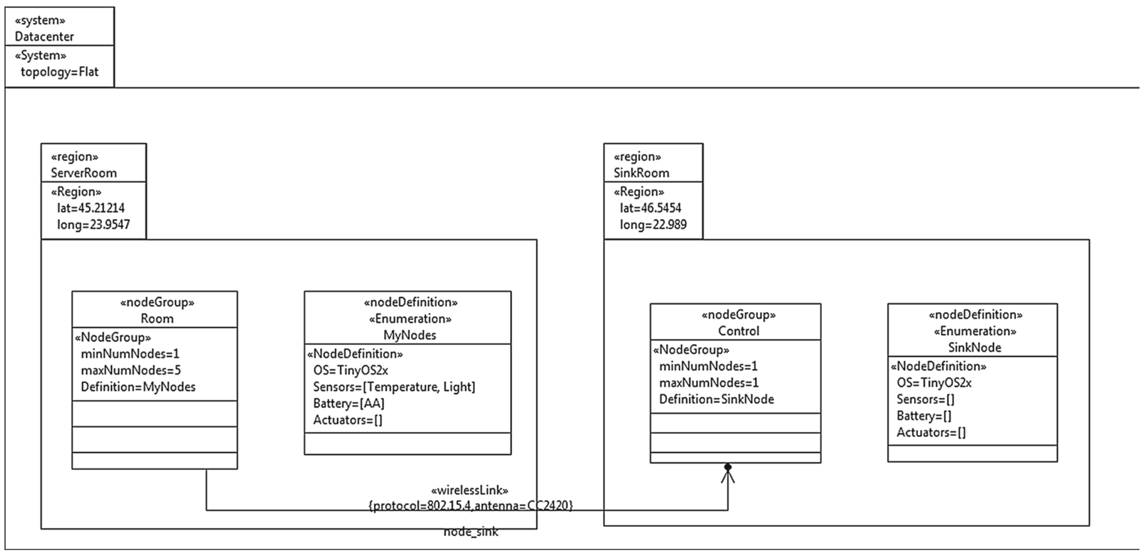
«system» Datacenter
«System» topology=Flat
«region» ServerRoom
«Region» lat=45.21214 long=23.9547
«nodeGroup» Room
«NodeGroup» minNumNodes=1 maxNumNodes=5 Definition=MyNodes
«nodeDefinition» «Enumeration» MyNodes
«NodeDefinition» OS=TinyOS2x Sensors=[Temperature, Light] Battery=[AA] Actuators=[]
«region» SinkRoom
«Region» lat=46.5454 long=22.989
«nodeGroup» Control
«NodeGroup» minNumNodes=1 maxNumNodes=1 Definition=SinkNode
«nodeDefinition» «Enumeration» SinkNode
«NodeDefinition» OS=TinyOS2x Sensors=[] Battery=[] Actuators=[]
«wirelessLink» {protocol=802.15.4,antenna=CC2420}
node_sink

Fig. 2 Structural model of a simple application

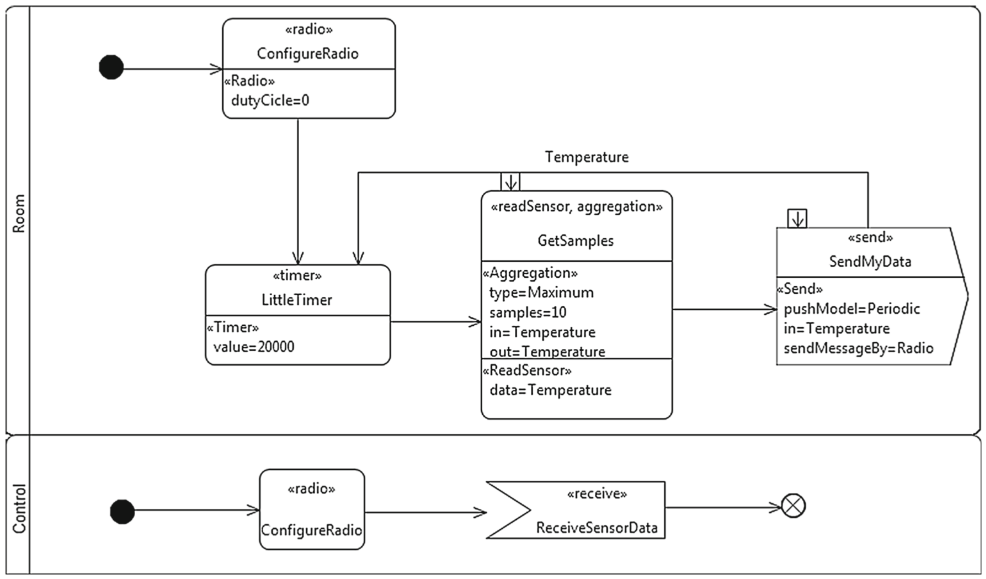
Room
«radio» ConfigureRadio
«Radio» dutyCycle=0
Temperature
«readSensor, aggregation» GetSamples
«Aggregation» type=Maximum samples=10 in=Temperature out=Temperature
«ReadSensor» data=Temperature
«timer» LittleTimer
«Timer» value=20000
«send» SendMyData
«Send» pushModel=Periodic in=Temperature sendMessageBy=Radio
Control
«radio» ConfigureRadio
«receive» ReceiveSensorData

Fig. 3 Behavior model of a simple WSN application

duty cycle; such a configuration is independent on the specific platform the application will run at.

The non-functional requirements currently supported in ArchWiSeN's implementation are network lifetime, data accuracy and asynchronous communication. The design decisions related to the choice of these non-functional requirements are strictly related to their importance in the WSAN domain. We surely did not add all the non-functional requirements that are relevant for all WSAN application, nor the missing non-functional requirements are less important, our goal was to consider some non-functional requirements that have an impact during early stages of the application design phase. For example, the network lifetime property is a non-functional requirement that is critical in the WSAN domain, since if the nodes have their battery depleted before the required period of collection, the acquired data can be considered irrelevant to the application. Therefore, in one of our previous works we managed to add to ArchWiSeN a "Network Energy Consumption Analyzer" [35] that estimates the usage of power of a node and provides such information to the developers where they can compare the obtained information with the application requirements in early stages of development. Data accuracy is a property that corresponds to the precision of the reported phenomena and is very important in domains where data precision is a required quality of service attribute [3]. The precision of the data obtained through a sensor can be determined by the sensor chip inherent characteristics, and thus, it is

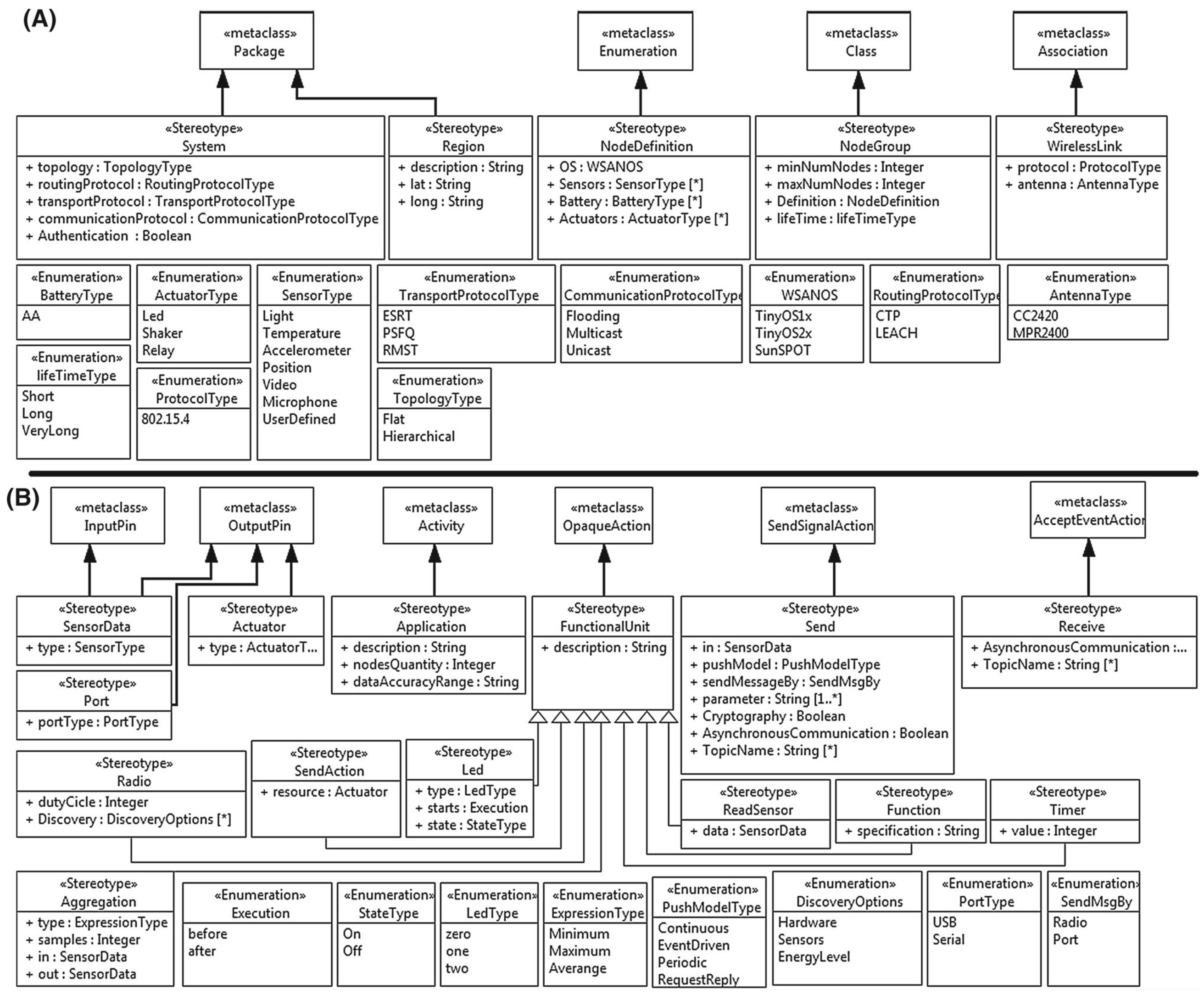

Fig. 4 All stereotypes available in the WSAN profile. **a** Structural view stereotypes, **b** behavioral view stereotypes

a non-functional requirement that limits the number of target hardware and is defined at the structural view. Finally, as the communication is critical for the WSAN domain, it is important to properly design the application according to the chosen communication policy. WSAN applications can implement different data delivery models based on the trigger of events or in a predefined frequency of data collection. Event-based applications are often implemented with asynchronous communications, and the decoupling in both space and time provided by such communication paradigm is well suited for WSANs in general.

Figure 4 represents all the available stereotypes from the WSAN profile for the current implementation of ArchWiSeN. The arrows in Fig. 4 represent extensions according to the UML specification [41]. Figure 4a presents the stereotypes responsible for enhancing the UML class diagram used in the structural view. Figure 4b presents the stereotypes responsible for enhancing the UML activity diagram used in the behavioral view. The stereotypes of the configuration view are represented in both parts of Fig. 4.

2.1.2 Platform-specific meta-model

A platform-specific model (PSM) must be able to describe an application implemented in a specific target platform. A PSM includes all the information required by the chosen platform to perform its tasks when executing the application. In this work, we focus only in the TinyOS platform to be used at the PSM level, but it is important to emphasize that out approach is open to the different platforms existing for WSAN. In fact, our previous works [35,36] have already assessed the use of the proposed MDA approach for different platforms, as the Sun SPOT, a widely used WSAN platform for academic experimentation. In [36], we tested the switching between

WSAN platforms using the same application model and concluded that our approach was able to generate applications for both platforms from a single platform-independent model, thus fully exploiting one of the advantages preconized by MDA.

TinyOS is an open-source operating system tailored for wireless sensor networks. The NesC [18] language is an extension of the C programming language used to implement TinyOS applications. TinyOS is event-oriented and has a component-based architecture, which enables the implementation of applications using preexisting components. TinyOS libraries provide interfaces and components for common abstractions and functions such as packet communication, routing, sensing, actuation, and storage for different types of sensors. Several sensor hardware platforms, such as Mica [25] and Telos [40] family, use TinyOS.

TinyOS components are classified as *modules* or *configurations* [18]. *Modules* implement all the functionalities supported by the application, while *configurations* describe how TinyOS components are connected (*wired* on NesC notation) to each other. Every component can provide or use an *interface*. There are two sources of concurrency in TinyOS: *tasks* and *events*. *Tasks* are a computation mechanism that runs to completion and do not preempt each other. *Events* mean either a completion of a *split-phase operation* or an event from the environment, and they also run to completion, but may preempt the execution of a task or another event. Because tasks are executed non-preemptively, TinyOS has no blocking operations. Thus, all long-latency operations are split-phase; the operation request and completion are separated functions. *Commands* are typically requests to execute an operation. If the operation is split-phase, the command immediately returns and the completion will be signaled with an event. For the aforementioned reason, a *module* using an *interface* must implement its *events*. The use of an *interface* also allows a *module* to call *commands*, while a *module* providing an *interface* must implement the *commands* and its completion will be signaled with an event.

The semantics of the PSM for TinyOS considers that a TinyOS application can be represented as a set of states interconnected by control flows and triggers that can change the node's state. We decided to use such approach because of the similarity between a state machine and the TinyOS's programming model (and event-driven OS). Moreover, a sensor behavior is very naturally represented in terms of its different states and the transitions between them, triggered by internal and external events in the network. Therefore, a sensing-based application can be properly represented as a set of states and actions. In addition, TinyOS is a component-based OS and the applications are organized as a set of interconnected (or wired) *components*. Therefore, we decided to adopt a UML's component diagram to represent the organization of the *components* present in a TinyOS application. In

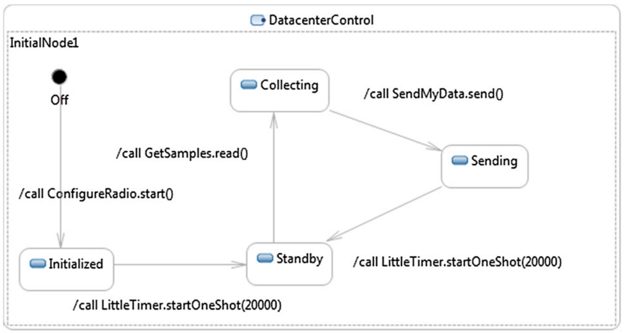

Fig. 5 Platform-specific model of the Smart Home application

our approach, an application model for the TinyOS platform respects the TinyOS's architecture and programming model. In this work, we consider the following possible states for a WSAN node: (1) turned off, (2) initialized (3), standby, (4) collecting, and (5) sending. Figure 5 presents an instance of platform-specific model for the TinyOS platform.

The *turned off* state means that the device is disconnected from its power source, either by direct set by the developers (putting the power switch in the *off* position) or due to the total exhaustion of its energy source (i.e., the battery). From the *off state* the device can only switch to the *initialized* state and no other state may, logically, return to the *turned off* state (as we consider that a node cannot be turned off through TinyOS's commands). However, in the physical world a node can change to the *turned off* state when it has a total discharge of its power source or by the manual switching of the power switch to the off position.

The *initialized* state corresponds to the device state after the realization of the hardware tests automatically triggered by the operating system initialization and the tests added by developers. From the initialized state, the device can change to any of the other defined states (standby, collecting, sending) via transitions indicated as TinyOS's events and triggered by the operating system. The state *standby* indicates that the device is initialized and has performed all its tasks or commands and it is waiting for an event so it can change the state.

The *collecting* state indicates the state where the device is performing data collection using one (or more) of its embedded sensor boards. The device should leave the collecting state when the operating system triggers the *ReadDone* event associated with the sensor target collection. From the *collecting* state, the device may return to the *standby* state or to the sending *state*.

The *sending* state indicates the state where the device is transmitting data through its radio unit or a data communication port (for example, a serial port). The device should leave the sending state when the operating system triggers the *SendDone* event associated with the conclusion of the

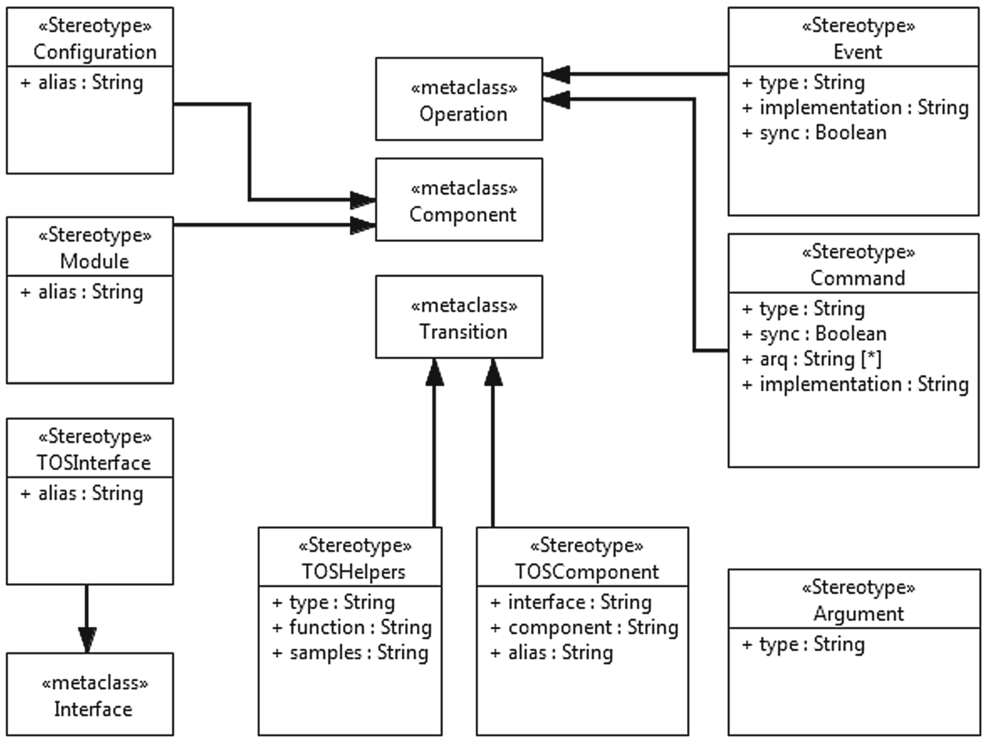

Fig. 6 All stereotypes available in the TinyOS profile

transmission. From the state *sending* the device may return to the *standby* state or to the *collecting* state.

Finally, as these states are only present in a model instance of the PSM (or an application model), using the native properties of the UML state-transition diagram, it was necessary to extend the UML state-transition and component diagrams to add the properties needed to specify the platform features. There is a need to create properties to target the following platform features: *component*, *module*, *event*, *command*, *interface*, and more. Thus, we added string properties that modify UML's *transitions* and *operations* to correctly map from the UML model to the TinyOS platform features.

In this work, we specified a UML profile that extends the UML state transition and component diagrams. Such UML profile defines the specific characteristics for all TinyOS elements (Fig. 6).

The «component» stereotype was created to represent TinyOS components extending the UML meta-class component. The TinyOS's *modules* and *configurations* are represented by the «module» and «configurations» stereotypes, which are both generalizations of the «component» stereotype. Some properties were added to the «module» stereotype to implement the application behavior. Some properties were also added to the «configuration» stereotype to manage the wirings between components. Furthermore, the UML *operation* meta-class was extended to encompass the stereotypes for «events» and «commands» along with the characteristics they require (as type, arguments, implementation, etc).

At last, the UML *transition* meta-class was extended to «TOSComponent», «TOSHelpers» in order to create the relations between the different UML states and the components that are implementing the state change.

As mentioned before, the domain engineering process enables future implementation of new WSAN platforms. For this purpose, the process shown in Fig. 1 must continue from the "Create Platform Specific UML profile" activity and follow the "Build M2M transformation program" and "Build M2T transformation program" activities for the development of the transformation programs.

2.1.3 Transformations

For each WSAN platform added to the MDA infrastructure, the model to model (M2M) and the model to text (M2T) transformations must be defined. The M2M transformations are designed to map domain knowledge described in a PIM model into elements of a specific platform model, thus introducing platform-related characteristics without losing any domain information.

In our current approach, the M2M transformation receives, as source models, the UML class and activity diagrams extended by the WSAN profile, and, as result, produces two target diagrams, which comply to UML state-transition and component diagrams augmented with the UML profile of the TinyOS platform (or the TinyOS PSM meta-model). We defined such transformation using a set of standardized queries over the PIM meta-model in order to make parts of the transformation reusable when implementing the support for a new WSAN platform into the ArchWiSeN.

The *Activity2State* algorithm (Fig. 7) presents the transformation from a behavior composed of a sequence of actions (such as the UML activity diagram used in the PIM of our approach) into an event-based model (such as the UML state-transition diagram used in the PSM). Line 1 shows the algorithm name and its output, a UML state-transition diagram stereotyped with the profile «PSMTinyOS» (PSM!State Machine : UML «PSMTinyOS»), while line 3 is the entry model, a UML activity diagram stereotyped with the profile «PIM» for WSAN (PIM!Activity: UML activity diagram «PIM»). When started, the transformation program performs a search (line 9) by the UML element called *InitialNode* in the entry model (PIM!Activity). If such element is found, the transformation creates the basic states of the output model (turned off, initialized, standby, collecting and sending). It is important to note that the states *collecting* and *sending* will only be added to the outgoing model if at least one element «readSensor» or «send», respectively, is located (lines 12 and 15). From this initial configuration, the transformation program follows conducting an analysis of each of the UML *control flow* element (arrows indicating the direction of activities flow in the UML activity diagram). This analysis is performed as follows. The algorithm looks into the activity diagram transitions combining the information of the source element (line 22) and the target (lines 27, 30 and 33) activity to determine which triggers and effects (lines 24 and 25) must be added to a new UML *transition* element connecting the corresponding states to the output model (line 56). This process is repeated exhaustively the transformation considering all possible transitions between the stereotypical elements in the PIM model.

Figure 8 shows a snippet of the actual ATL transformation code to generate the platform-specific model for the TinyOS platform. Line 1 represents the transformation rule name, while lines from 2 to 10 represent the input UML element (line 3) that is going to be transformed to an element of the output meta-model (line 5). This output model also has inner properties that are going to be filled according to other transformation rules.

The hardware-specific information is included in different properties. For example, the *component* property present in the TOS component stereotype relates to the name of the TinyOS *component* that implements the target hardware. In the TinyOS platform, hardware devices such as antenna, LEDSs, and sensors are implemented by the SO as *components*. Thus, each different hardware has a different *component* implementation, distinguished from each other only by the *component* name (i.e., a string). The access to the component properties is done through standardized *interfaces*. Therefore, hardware properties in the PIM are converted to the name of the correspondent TinyOS *component* that implements such hardware in the TinyOS platform.

It is important to clarify that ArchWiSeN's PIM meta-model also allows modeling properties that are not meant to be targeted as a source of a platform model. For example, the PIM meta-model offers parameters to configure *Regions* in terms of the latitude and longitude properties. Such characteristic will be not be considered when creating the platform model, but they are extremely necessary since they are part of the application documentation and can be also used for decision making, scheduling algorithms, disseminating messages to the sink, and more.

Following our *Smart Home* case, the models presented in Figs. 2 and 3 are automatically transformed into the model presented in Fig. 5. For example, in Fig. 5 there is a transition from the *booted* state to the *idle* state with and effect to call the *LittleTimer* component and configure it to fire after 20,000 ms. When such *LittleTimer* is fired, the node will change state from *idle* to *reading*. Such modeled behavior was automatically generated by reading the model part that defines the *Room* behavior (Fig. 3). When the transformation program identifies that the PIM model has an activity named *LittleTimer* with timer stereotype, it will check which is the next activity (by looking to the UML's *control flow* destination) to be performed after such *LittleTimer*. In this case, after the timer there is a «readSensor» stereotype. Thus, the transformation discovers that, when the timer is trigged, the application has to call the TinyOS's command to read a sensor, and so on.

The model to text (M2T) transformation takes as input the UML PSM diagram (UML component and state machine diagrams augmented with the PSM TinyOS profile) and generates, as output, NesC [18] source code. Most of the M2T transformation rules consist of simple mappings that transform each UML PSM elements into their corresponding NesC code following the BNF rules for the NesC programming language.

Figure 9 shows a snippet of the M2M transformation code. In line 1 it is possible to notice the creation of the file. In lines 5–16 this transformation reads the application's PSM model to insert into the *module* all the *used* interfaces. Finally, Fig. 10 presents a snipped of the code automatically generated after the consecutive execution of the M2M and the M2T transformations for our Smart Home case.

To support another WSAN platform into the ArchWiSeN approach, developers have to perform the domain engineering process to create the platform-specific models and transformations. Moreover, as the transformations provided in our approach are defined by using a set of standardized queries over the PIM meta-model, the effort to add another platform is concentrated on defining a new PSM, reusing such queries (and/or creating new queries) to develop the new M2M transformation and developing the M2T transformation.

Fig. 7 Algorithm for the PIM to TinyOS transformation

```
program Activity2State (PSM!StateMachine : UML «PSMTinyOS»)
const:
  PIM!Activity: UML Activity Model «PIM»;
var:
  off, booted, idle, reading, sending : UML State «PSM»;
  flow : UML Control Flow «PIM»;
  transition : UML Transition «PSM»;
begin:
  if PIM!Activity.getInitialNode() == TRUE then
      PSM!StateMachine.Region.name = PIM!Activity.InitialNode.name;
      PSM!StateMachine.Region.subvertex ∪ {off ,booted,idle}
      if PIM!Activity.hasStereotype('DSL::ReadSensor') == TRUE then
           PSM!StateMachine.Region.subvertex ∪ {reading};
      endif
      if PIM!Activity.hasStereotype('DSL::Send') == TRUE then
           PSM!StateMachine.Region.subvertex ∪ {sending};
      endif
  else
      return;
  flow:= getNextControlFlow(PIM!Activity);
  repeat
      if flow.source.hasStereotype('DSL::Radio') == TRUE then
          --Add alias values for inteface and component
          transition ∪ AddTrigger();
          transition ∪ AddEffect();
        --Check the action after the radio initialization
          if flow.target.hasStereotype('DSL::Timer') == TRUE then
              --Define the apropriate model for the RADIO->TIMER flow
          endif
          if flow.target.hasStereotype('DSL::ReadSensor') == TRUE then
              --Define the apropriate model for the RADIO->READ flow
          endif
          if flow.target.hasStereotype('DSL::Send') == TRUE then
              --Define the apropriate model for the RADIO->SEND flow
          endif
          (...)
    endif
      if flow.source.hasStereotype('DSL::Timer') == TRUE then
          -- Add alias values for inteface and component component
          transition ∪ AddTrigger();
          transition ∪ AddEffect();
         -- Check the action after the radio initialization
          if flow.target.hasStereotype('DSL::Timer') == TRUE then
          -- Define the apropriate model for the TIMER->TIMER flow
          endif
          if flow.target.hasStereotype('DSL::ReadSensor') == TRUE then
              --Define the apropriate model for the TIMER->READ flow
          endif
          if flow.target.hasStereotype('DSL::Send') == TRUE then
              --Define the apropriate model for the TIMER->SEND flow
          endif
          (...)
      endif
      --Define the other rules analogously
      (...)
 PSM!StateMachine ∪ transition;
 flow:= getNextControlFlow(PIM!Activity);
until(flow == NULL)
return PSM!StateMachine;
end.
```

```
rule InitialNode {
      from
              input : PIM!InitialNode
      to
              output: TOS!Region(
                      name <- input.name,
                      subvertex <- Set{off},
                      subvertex <- Set{booted},
                      subvertex <- Set{idle},
                      transition <- input.refImmediateComposite().edge
              ),
              off:TOS!Pseudostate (
                      name <- 'Off'
              ),
              booted:TOS!State (
                      name <- 'Initialized'
              ),
              idle:TOS!State (
                      name <- 'Standby'
              ),
              reading:TOS!State (
                      name <- 'Collecting'
              ),
              sending:TOS!State (
                      name <- 'Sending'
              ),
              (…)
              do {
                      (…)
              }
}
```

Fig. 8 A snippet of the M2M transformation code to generate the PSM for the TinyOS platform

```
[file (aStateMachine.name.concat('C.nc'), false, 'UTF-8')]

#include "[aStateMachine.name.toLower().concat('.h')/]"

module [aStateMachine.name.concat('C')/] {
[for (r : Region | aStateMachine.region)]
    [for (t : Transition | r.transition)]
         [if (not t.getValue(t.getAppliedStereotype('TinyOS::TOSComponent'),
'interface').toString().equalsIgnoreCase(''))]
         uses interface [t.getValue(t.getAppliedStereotype('TinyOS::TOSComponent'),
'interface')/] as [t.getValue(t.getAppliedStereotype('TinyOS::TOSComponent'), 'alias')/];
              [if (t.getValue(t.getAppliedStereotype('TinyOS::TOSComponent'),
'interface').toString().equalsIgnoreCase('AMSend'))]
         uses interface Packet;
         uses interface AMPacket;
              [/if]
         [/if]
    [/for]
[/for]
```

Fig. 9 A snippet of the M2T transformation to generate TinyOS platform source code

```
#include "room.h"

module RoomC {
            uses interface Read<uint16_t> as GetSamples;
            uses interface Boot as InitialNode1;
            uses interface AMSend as SendMyData;
            uses interface Packet;
            uses interface AMPacket;
            uses interface SplitControl as ConfigureRadio;
            uses interface Timer<TMilli> as LittleTimer;

} implementation {
(...)
}
```

Fig. 10 A snippet of the code generated for the Smart Home application

2.2 Application engineering

This section presents all the activities that the WSAN application developers should perform during the application engineering process to generate executable source code for a chosen WSAN platform. The artifacts needed to perform the application engineering are provided by the domain engineering and were developed by the ArchWiSeN's developers.

The activity diagram presented in Fig. 11 is divided in three swimlanes separating the activities by their respective actors, namely domain expert, network expert, or automatically performed by the ArchWiSeN infrastructure.

The first activity in the diagram (Fig. 11), called "Requirements Analysis," is performed by both developers (domain expert and the network expert), where these actors gather all information needed to build the target WSAN application. During this activity, the domain and network experts get all information needed to build the application. This requirements analysis is a traditional activity of requirements elicitation carried on as part of any software development process. The software artifacts produced as outcome of this activity (UML diagrams as *use cases*, textual documents, etc) represent the system requirements and compose the CIM, which will be used in further phases. Therefore, the CIM is actually a set of requirement documents that include functional (related to the application logic) and non-functional requirements (related to the configuration of the WSAN platform). These requirements are used by the domain expert in the "Model PIM" activity. The outputs of this activity are the UML activity and class models enhanced by the WSAN UML profile. At the same time, as a typical WSAN development scenario, the network expert starts the "Choose Platform" activity, to evaluate the available platforms and choose the one that best meets the elicited requirements. The *"Choose Platform"* activity is not a modeling activity. Instead, it represents a design decision to be made by the network expert on the implementation platform to be used. Network experts are developers that have a deep knowledge in existing WSAN platforms, including their requirements and limitations. Thus, knowing the application requirements from the CIM and having the knowledge of the WSAN field, this expert can chose the platform that best fits the applications needs.

With the information modeled by the domain expert in the *Model PIM* activity and knowing the platform that best fits the application requirements, the network expert is able to perform the "Apply Configuration Marks" activity and edit the model to insert the system's properties according to the non-functional requirements. The MDA approach specifies the use of UML stereotypes to detail models by adding marks to the model. These properties denote characteristics such as the network protocol, node hardware specification, and more.

Following, the "Verification" activity will be manually performed by both developers in order to check whether all modeled information meets the application requirements. If any requirement is not met, all development process should start again with the *"Requirement Analysis"* activity, but maintaining all the already modeled information so as only the necessary editions need to be done. Sometimes not all the requirements can be guaranteed due to the particular characteristics of these systems. Therefore, developers need to re-analyze the requirements to re-model parts of the application. This does not mean that the entire requirements elicitation phase must be re-done. Instead, it should be refined as it is commonly done in iterative and incremental software development processes. Usually the initial version of the requirements document can have a lot of imprecision because it can be based in an informal and/or incomplete notion of the system scope. The requirements refining process aims to adjust the requirements into more realistic and precise documents to help the developers to ensure that all the requirements are met.

If all application requirements are met, the "*Apply transformation M2M*" activity will be performed by the MDA infrastructure (ArchWiSeN). Such activity takes as input the PIM instance (*Application UML Model*) to generate as output a PSM instance that represents the realization of the application in the specific platform chosen by the network expert. After this activity, the generated PSM can be refined by the network expert in the "Refine Model" activity in order to augment the model with information like network-related specificities of the target platform or choosing an application library that best fits the application non-functional requirements. For instance, the network expert can define policies for saving network resources, as energy, by selecting a particular set of components that are not automatically configured by the M2M transformation. This refinement will be usually performed through extra M2M transformations at the PSM level but can also be performed manually by editing the model to add or modify properties. Considering the TinyOS platform, such refinement encompasses the switching of a target *component* implementation automatically added by the transformation into a specific, and more tailored, *component* as decided by the developers. A PSM refinement can be done by following three steps: (1) removing the components that are declared, but are no longer used, (2) adding the new components, and (3) carrying out the wirings between the components and interfaces being used. As these changes occur at the PSM level and our approach does not implement bi-directional transformations, such changes are not transposed to the PIM level. On the other hand, the code generated from M2T will respect the PSM modifications. Unfortunately, re-running the M2M transformation will erase the manual changes made to the PSM model. Our approach currently does not tackle the issue of synchronization between PIM and PSM when a manual edition at the PSM level occurs.

Finally, the "Apply M2T Transformation" activity is accomplished. This activity has two inputs: (1) the PSM model refined by the network expert and (2) the chosen platform code templates. It generates, as output, the application source code to be deployed in the sensor nodes. The generated code is then refined by both developers (each one regarding his/her specific knowledge) in the "Refine Code" activity in order to add improvements such as application-specific func-

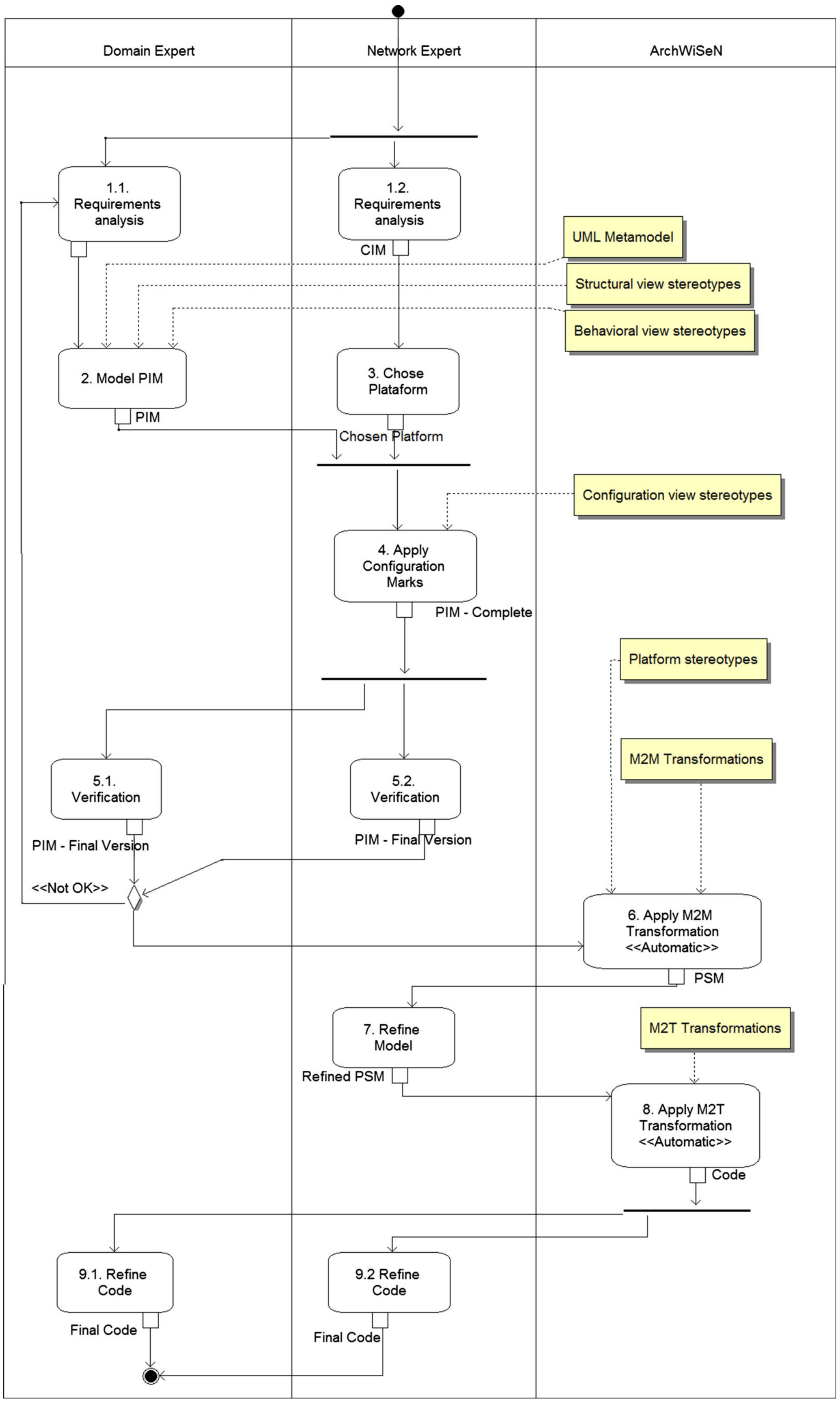

Fig. 11 Activity diagram representing the process to develop a WSAN application

tions or protocol parameters not automatically generated by the MDA transformation.

As previously mentioned, WSAN applications are developed according to two viewpoints. The first is the application domain experts' viewpoint and the second is the network experts' viewpoint. Building WSAN applications using the proposed MDA approach promotes the division of responsibilities among developers of these different viewpoints, allowing them to use their specific knowledge and skills and unburdening them from dealing with requirements that do not belong to their expertise field. Through the diagram presented in Fig. 11, it is possible to notice that most activities that involve some type of specific knowledge of a platform or a domain are assigned to its respective expert. In this case, the only exceptions are "Requirement Analysis" and "Validation" activities that are performed by both developers to create a bridge between both specifications. Furthermore, the application engineering process promotes the separation of the domain logics from the network-related logic. The domain expert only needs to handle UML elements and later to include code of a function regarding his/her specific knowledge. On the other hand, the network expert only needs to deal with network-related elements, both at modeling and at code level.

Maintenance is a critical activity in WSAN applications. The most common issues in such activity are the need of handling platform-specific code and the lack of software documentation. However, using the proposed application engineering process, an application can be defined regardless of the chosen sensor platform; all information is kept by UML models at a platform-independent level and can be transformed into a platform-specific model for any actual or future WSAN platform, depending only on the existence of a respective transformation program and the platform metamodel. Furthermore, actual and future developers will always be able to read and understand UML models patterns, making an update on a WSAN system to be reduced to a simple change in a standardized model.

3 Evaluation

The MDA approach presented in this paper was evaluated in two ways: through a controlled experiment (detailed in Sect. 3.1) and a proof-of-concept (detailed in Sect. 3.2). In the controlled experiment, a quantitative evaluation was carried out in order to assess the ArchWiSeN in terms of productivity, comprehension, and reuse. In the proof-of-concept, several applications from different WSAN domains were developed aiming at demonstrating in practice the employment of the ArchWiSeN and to show ArchWiSeN's usefulness in providing the separation of concerns between the two different developers (domain expert and the network expert), and its capability to manage the application's requirements. The technical measured informations that were used to present the benefits of our approach are: the time necessary to develop an application, the generated number of lines of code, the time necessary to introduce new requirements, and the developers opinion on ease of use and usability.

3.1 Controlled experiment

We have conducted a controlled experiment in order to verify whether the proposed application engineering process and its associated MDA artifacts meet the goals of our paper (stated in Sect. 1) and also to assess the complexity and benefits of using ArchWiSeN when compared to traditional methods for the development, requirement elicitation, and representation of WSAN applications. The planning (Sect. 3.1.3) of the experiment reported in this section is based on the guidelines proposed in [2], a widely cited reference for experimental design in software engineering. The execution of the controlled experiment is based on obtaining the development time from two different groups composed of participants with the same profile, performing the same tasks (Sect. 3.1.4) and using different approaches (ArchWiSeN or code-and-fix). The two groups were both trained and had access to the same materials. In order to obtain most of the data required to analyze this controlled experiment, a post-execution questionnaire was used to identify user's thoughts about ArchWiSeN's usefulness, ease of use, and understandability. We selected the questions used in the questionnaires presented to the participants and we tested the internal consistency of the given answers based on psychology studies such as [7,9,20,28] and [23].

3.1.1 Experiment goals

Considering the stated goals of our work, we followed the *Goal Question Metric* [4] (GQM) method to define our primary research goals, which are used to guide the conducted experiment. As already mentioned, the goals of our work are: (1) to define a process to create all MDA artifacts; (2) to develop the MDA artifacts (PIM, PSM, and transformations); and (3) to define a process to build WSAN applications where developers can benefit of better productivity, comprehension, separation of concerns, and reuse. Once defined the research goals, we refined them in a set of questions. Finally, we defined metrics to provide the grounding to answer the questions.

The following two research goals are considered by the conducted controlled experiment.

First Goal To analyze the application engineering process and the MDA infrastructure (ArchWiSeN) with the purpose of evaluating their effectiveness with respect to the **under-**

standability and **productivity** in the development of WSAN applications.

Second Goal To analyze the application engineering process and the MDA infrastructure with the purpose of evaluating their effectiveness with respect to the **reuse** of artifacts for WSAN application development.

The experiments were performed in the context of developers endowed with WSAN programming expertise using both the *code-and-fix* and ArchWiSeN approach, creating periodic, event-driven, and request–reply applications.

3.1.2 Questions

Following the guidelines presented in [4], the goals of the controlled experiment are further detailed as research questions. The GQM questions are defined to characterize the object of measurement with respect to a selected quality issue and to determine its quality from a selected viewpoint. Questions Q1–Q4 are related to the first research goal, targeting the understandability and productivity characteristics, while questions Q5–Q6 refine the second research goal, targeting the reuse provided by the ArchWiSeN usage. The answers to these questions were obtained through the assessment of the metrics defined for the controlled experiment (see Sect. 3.1.6).

Q1. Is the ArchWiSeN application engineering process effective in terms of time to develop WSAN applications when compared to the code-and-fix approach?
Q2. How effective are the available UML profiles in ArchWiSeN to deal with the application development?
Q3. Do WSAN developers state that ArchWiSeN is easy to use?
Q4. Do WSAN developers state that ArchWiSeN aids them to develop a WSAN application?
Q5. How effective are the UML mechanisms used in ArchWiSeN to deal with the implementation of new requirements?
Q6. Do WSAN developers state that ArchWiSeN aids them to reuse a WSAN application?

3.1.3 Planning, experimental units, and materials

The experiment was planned to occur in two phases. In the first phase, the participants were trained in two topics: (1) WSN application development using the TinyOS programming model (including NesC language), and (2) WSN application development using UML models and ArchWiSeN. The participants answered a characterization questionnaire aiming to collect information on professional experience and skills in software development. In addition, reference materials as TinyOS documentation, ArchWiSeN's documentation, and slide presentations used in the training were given to the participants. The second phase consisted in developing applications using the two different approaches: ArchWiSeN and code-and-fix. In this phase, the participants were randomly divided into two groups (group I and group II) and it was raffled which group worked first with the ArchWiSeN approach and which group worked first with the code-and-fix approach. Group I used the ArchWiSeN approach first and code-and-fix latter, while group II used the approaches in the reverse order (code-and-fix first, then ArchWiSeN). The two groups performed the same five tasks (described in Sect. 3.1.4). However, we assigned for each participant a different order of task to be performed (Table 1 for further details).

Essentially, the profile of participants involved in the experiments includes: (1) educational level; (2) level of knowledge on the field of distributed computing; and (3) level of knowledge on the field of WSAN platforms and programming. The participants of this experiment are ten Master Students of the Computer Science Course from the Federal University of Rio de Janeiro, randomly organized into two groups. All students have a similar profile, with expertise in WSAN application development, specifically in the TinyOS platform, learned during a distributed sys-

Table 1 Participants and tasks configuration

Participant	First day		Second day	
	ArchWiSeN	Code-and-fix	ArchWiSeN	Code-and-fix
P1	T1, T2, T3, T4, T5	–	–	T1, T2, T3, T4, T5
P2	T2, T3, T4, T5, T1	–	–	T2, T3, T4, T5, T1
P3	T3, T4, T5, T1, T2	–	–	T3,T4,T5,T1,T2
P4	T4, T5, T1, T2, T3	–	–	T4, T5, T1, T2, T3
P5	T5, T1, T2, T3, T4	–	–	T5, T1, T2, T3, T4
P6	–	T1, T2, T3, T4, T5	T1, T2, T3, T4, T5	–
P7	–	T2, T3, T4, T5, T1	T2, T3, T4, T5, T1	–
P8	–	T3, T4, T5, T1, T2	T3, T4, T5, T1, T2	–
P9	–	T4, T5, T1, T2, T3	T4, T5, T1, T2, T3	–
P10	–	T5, T1, T2, T3, T4	T5, T1, T2, T3, T4	–

tems course. The participants signed a confidentiality and consent form that explained the experiment procedure, benefits, and their freedom to quit the experiment whenever they wanted.

3.1.4 Tasks

To collect the metrics used in the experiment, a set of tasks was planned to be executed by the participants as follows. Table 1 summarizes the order of tasks given to each participant.

Task 1 Create a WSN application for the TinyOS platform to collect periodic data of light. The application requests the WSN nodes to perform a sensing of the environment's light (every 10 s) and repeat this procedure until gathering 10 samples of the light data. After the 10 samples are collected, only the highest value obtained by the nodes are sent to the sink node and then the data stored in the node's memory are cleaned. The data are received in the sink through a radio broadcast communication. After receiving the information, no particular action is taken by the sink.

Task 2 Create a request-response WSN application for the TinyOS platform. For this application, after booting and starting the radio, the application does not have to perform any task, it only expects to receive a request from the sink. The request should be received through a radio broadcast communication. With the request arrival, the node will trigger a counter to perform the temperature sensing every 5 s after the request reception. When the reading is finished, the node triggers a 20 s timer to send the temperature data over the network through broadcast. After sending the collected data, the node should return to the initial state to wait for request messages.

Task 3 Study the code/model of an oscilloscope application. The application goal is to collect the current voltage of the battery and to send this information to the sink node. This task involves changing the source code/model considering the following new requirements. Requirement 1: change the number of samples collected by the application. Requirement 2: collect temperature data in addition to the data already obtained and to save the information in a similar way used in oscilloscope application. Requirement 3: decrease the frequency with which data arrive at the sink.

Task 4 List the functional and non-functional requirements of a given structural heath monitoring application.

Task 5 List the functional and non-functional requirements of a given Smart Home application.

3.1.5 Hypotheses, variables, and constructs

This subsection describes: (1) the null hypotheses; (2) the constructs, the hypothetical variables that are being measured, grouped into constructs to calculate their correlation; and (3) the dependent and independent variables existing in the experiment. For each primary research goal defined for our experiment, the null hypotheses, denoted $\mathrm{H0_{ij}}$, and their corresponding alternative hypotheses, denoted $\mathrm{H1_{ij}}$, need to be derived, where i corresponds to the goal identifier, and j is a counter when there is more than one hypothesis per goal. For example, the null hypotheses $\mathrm{H0_{11}}\mathrm{H0_{12}}$ and $\mathrm{H0_{13}}$ are related to the first goal, while the hypothesis $H0_{21}$ is related to the second goal. The null hypotheses defined for this experiment are:

H0$_{11}$ The use of ArchWiSeN is equivalent to the code-and-fix development in terms of development time;
H0$_{12}$ The use of ArchWiSeN maintains the same degree of efficiency in the production of software artifacts compared to the code-and-fix approach;
H0$_{21}$ The use of ArchWiSeN maintains the same degree of reuse of components compared to the code-and-fix approach;
H0$_{13}$ The use of ArchWiSeN maintains the same degree of understanding of the system compared to the code-and-fix approach.

To answer the defined research questions as well as to prove the formulated hypotheses true or false, the following constructs were defined: *development effort*, *requirements comprehension, perceived ease of use*, *perceived usefulness, reuse effort*, and *perceived reuse*. Such constructs represent the properties that we wish to evaluate in the experiment and they are measured by a set of metrics defined in Sect. 3.1.6. These metrics constitute the dependent variables (aka. response variables) of our experiment. The independent variables (aka. predictor variables) defined to this experiment are: (1) development methods, (2) participants profile, and (3) development environment.

3.1.6 Metrics

Since there is no standardized set of metrics available to evaluate the defined research questions, we specified our own metrics tailored to the purpose of our evaluation. The intent of these metrics is to quantitatively evaluate the ArchWiSeN and code-and-fix approaches in terms of constructs: (1) development effort, (2) requirements comprehension, (3) perceived ease of use, (4) perceived usefulness, (5) reuse effort, and (6) perceived reuse. All the metrics used are specified in Table 2. In all the equations, P represents the total number of participants.

$\mathbf{M_1}$ is specified to evaluate the **Development Effort** construct—this metric represents the total time required to execute the experiment tasks by each group considering both approaches (ArchWiSeN and code-and-fix). The goal of $\mathbf{M_1}$ is to measure the time required to specify a com-

Table 2 Metrics used in the evaluation

Metric	Variables
$M_1 = \sum \textbf{Task Duration}_{\{Ap;Gr\}}$	Ap = approach used by the participant to execute the tasks
	Gr = group where the participant was allocated
$M_2 = \frac{\sum \textbf{Comprehension}}{P}_{\{Ap,Q\}}$	Ap = approach used by the participant to execute the tasks
	Q = number of the question regarding requirements comprehension
$M_3 = \frac{\sum \textbf{Perceived Ease of Use}}{P}_{\{Q\}}$	Q = number of the question regarding perceived ease of use
$M_4 = \frac{\sum \textbf{Perceived Usefulness}}{P}_{\{Q\}}$	Q = number of the question regarding perceived usefulness
$M_5 = \frac{\sum \textbf{Reuse Task Duration}}{P}_{\{Ap\}}$	Ap = approach used by the participant to execute the tasks
$M_6 = \frac{\sum \textbf{Perceived Reuse}}{P}_{\{Q\}}$	Q = number of the question related to the perceived reuse

plete WSAN application using the available development artifacts.

M_2 is specified to evaluate the **Requirements Comprehension** construct—the goal of the **M_2** is to measure the participant's comprehension about application functional and non-functional requirements, using the available models for the different system views. Since a clear representation of requirements positively affects the application development, this metrics aids to answer question Q2. This metric is measured computing data obtained through a questionnaire based on the Likert scale [23].

M_3 is specified to evaluate the **Perceived Ease of Use** construct—the goal of **M_3** is to measure the participant's opinion about the usability of a given approach. This metric captures the degree to which a person believes that using a particular system would be free from effort [9].

M_4 is specified to evaluate the **Perceived Usefulness** construct—the goal of **M_4** is to measure the participant's opinion about the utility of a given WSAN development approach. This metric indicates the degree to which a person believes that using a particular system would enhance his or her performance in a given development task [9].

M_5 is specified to evaluate the **Reuse effort** construct—the goal **M_5** is to measure the time needed to implement new requirements in a preexisting WSAN application using the development artifacts available in a given software development approach. This metric is measured in terms of the average time needed to implement new requirements into an existing application.

M_6 is specified to evaluate the **Perceived Reuse** construct—the goal of **M_6** is to evaluate the participant's opinion about the reuse of the ArchWiSeN artifacts regarding WSAN application models. This metric indicates the degree to which a person believes that he/she is capable of reusing any existent software artifact in a given application.

Finally, Table 3 presents the correlations among the GQM elements (goal, questions, and metrics) defined in the experiment with the tasks performed by the participants and used to extract the metrics.

Table 3 Correlation among GQM elements and tasks

Goals	Questions	Metrics	Tasks
G1	Q1	M_1	T1, T2, T3, T4, T5
	Q2	M_2	T4, T5
	Q3	M_3	T1, T2
	Q4	M_4	T1, T2
G2	Q5	M_5	T3
	Q6	M_6	T3

3.1.7 Obtained data

This section summarizes the data collected and the performed treatment of the data. The metrics regarding both goals are obtained through the execution of the controlled experiment. The data for M_1 and M_5 metrics are obtained through the observation of the time spent by each participant to perform each one of the tasks, while M_2, M_3, M_4 and M_6 metrics are collected from the answers given to the survey questionnaires.[1] All the results are organized by each metric. Since the questionnaires were based on the Likert-type scales, a validation of the internal consistency of the adopted scales is needed [19]. We use the Cronbach's alpha [7] index to verify the reliability since this index is widely used to analyze Likert-type questionnaires. The Cronbach's alpha index is associated with the variation within an underlying construct (Sect. 3.1.5) where as correlations between the items of a construct increase the alpha index will generally increase as well. The alpha index (α) ranges in value from 0 to 1; the higher the score, the more reliable the generated scale is. An acceptable value for the reliability coefficient is 0.7 [28], but lower thresholds are sometimes used in the literature.

The metric **M_1** (development effort construct) was calculated based on the time recorded for each participant

[1] A table with the questions and the obtained answers can be found at http://www.consiste.dimap.ufrn.br/projects/archwisen/.

Table 4 Experiment duration per participant

	ArchWiSeN	Code-and-fix	ArchWiSeN	Code-and-fix	Total
	Duration (hh:mm:ss)				
P1	02:33:00	–	–	03:23:00	05:56:00
P2	01:54:00	–	–	03:36:00	05:30:00
P3	01:56:00	–	–	00:59:00	02:55:00
P4	01:36:00	–	–	01:10:00	02:46:00
P5	02:43:00	–	–	04:05:00	06:48:00
P6	–	02:34:00	01:30:00	–	04:04:00
P7	–	02:09:00	01:08:00	–	03:17:00
P8	–	02:36:00	01:37:00	–	04:13:00
P9	–	02:39:00	01:46:00	–	04:25:00
P10	–	02:45:00	01:30:00	–	04:15:00
Total	10:42:00	12:43:00	07:31:00	13:13:00	20:09:00

Table 5 Experiment duration per task

	Average time (hh:mm:ss)		Reduction (%)
	ArchWiSeN	Code-and-fix	
Task 1	00:35:42	00:44:06	19
Task 2	00:22:18	00:31:18	29
Task 3	00:11:30	00:22:12	48
Task 4	00:19:06	00:27:18	30
Task 5	00:20:42	00:30:42	33

to finish each task. Table 4 presents the experiment duration for each participant considering the execution of both approaches. Table 5 shows the average time for each task considering the results for all participants. The results show that there is an average of 25.2 % time reduction for the task execution time when comparing ArchWiSeN with the code-and-fix approach. This result points to a reduction in development time of WSAN applications when using the proposed approach. Tasks 1 and 2 (development) had an average reduction of 24 % regarding the development time when compared to code-and-fix approach considering a simple application development project. Furthermore, when using ArchWiSeN the time for execution of Task 3 (Reuse) was 48 % lower than when using the code-and-fix approach. Finally, Tasks 4 and 5 (understanding) had an average reduction of 31.5 % compared to code-and-fix approach meaning a reduction in the time required to understand the requirements of an application. Considering the development of WSAN applications, the obtained results are quite satisfying because the applications presented as development tasks (1 and 2) were not complex like the applications presented in the comprehension tasks (4 and 5). This could mean that the more complex the task is, the best benefits the developers will have when using our approach.

For $\mathbf{M_2}$ (comprehension construct) the obtained data shows that all participants (100 %) stated that it is hard to comprehend an application developed using the code-and-fix approach, while with the ArchWiSeN approach the participant's opinions are divided among Neutral (40 %), Easy (30 %), and Very Easy (30 %). Moreover, ArchWiSeN also achieved better results when compared to code-and-fix when considering the understanding of functional requirements. However, for the non-functional requirements the participants did not reach a consensus about comprehension when using ArchWiSeN. Nevertheless, 80 % of the participants stated that it is hard to comprehend non-functional requirements using the code-and-fix approach. Finally, 70 % of the participants answered that to understand the physical organization of the network in the application developed with ArchWiSeN is a very easy task, while 60 % answered that it is very hard to perform the same task with the code-and-fix approach. The α indexes obtained for the ArchWiSeN and code-and-fix *comprehension* constructs are, respectively, $\alpha = 0.7593$ and $\alpha = 0.56329$. However, the second α index indicates a potential poor internal consistency among the answers given to the code-and-fix approach. The inconsistency may have been caused by the participant's different expertise with low-level programming.

The data obtained for $\mathbf{M_3}$ shows that 70 % of the participants strongly agreed that was easy to use the application engineering process defined in ArchWiSeN. Moreover, 60 % of the participants agreed that they find easy to become skillful at using ArchWiSeN. However, 50 % of the participants (neutral or disagree) answered that ArchWiSeN is not flexible enough to interact with. The α index calculated with the results obtained for the construct *perceived ease of use* is $\alpha = 0.7898$ which denotes an acceptable result.

For the metric $\mathbf{M_4}$ (perceived usefulness construct) the results show that 80 % of the participants find ArchWiSeN useful for WSAN application development. The α index calculated with the results obtained for the construct *perceived usefulness* is $\alpha = 0.8287$ which denotes an acceptable result.

The data obtained for metric $\mathbf{M_5}$ (reuse effort construct) show that when using ArchWiSeN, the participants were able to obtain the best result (48 %) in terms of time reduction for the reuse task (Task 3) as shown in Table 5.

The last metric, $\mathbf{M_6}$, had an unacceptable result ($\alpha = 0.0624$) for the *perceived reuse* construct regarding the questions related to the ArchWiSeN. With such α value nothing can be assumed from the obtained answers. However, for the *code-and-fix perceived reuse* construct the α index value was $\alpha = 0.7866$ which denotes acceptable results. For the *perceived reuse* while using the code-and-fix, the participants responses were in average neutral (50 %) about making application changes. Moreover, for platform changes, the participants' average opinion is that making these changes are between hard and very hard (23 %).

3.1.8 Hypothesis testing

This subsection aims to analyze the obtained results of the controlled experiment in order to verify the hypotheses presented in Sect. 3.1.5. For this purpose, we must answer the questions (Q1 to Q6) using the respective collected metrics.

Q1. Is the ArchWiSeN application engineering process effective in terms of time to develop WSAN applications when compared to the code-and-fix approach?

The time spent to execute all tasks for the group that started with the ArchWiSeN approach (named group 1) was calculated by $\mathbf{M_1}$ 10:42:00 hours, while the time spent to execute all tasks by this same group using the code-and-fix approach was calculated by $\mathbf{M_1}$ as 13:13:00 hours. The time spent to execute all tasks for the group that started with the code-and-fix approach (group 2) was calculated by $\mathbf{M_1}$ as 12:43:00 hours, while the time spent to execute all tasks by this same group using the ArchWiSeN approach was $\mathbf{M_1}$ 7:31:00 hours.

As shown in Fig. 12, $\sum \textbf{Task Duration}_{\{\textbf{ArchWiSeN},\textbf{Group 1}\}} + \sum \textbf{Task Duration}_{\{\textbf{ArchWiSeN},\textbf{Group 2}\}} < \sum \textbf{Task Duration}_{\{\textbf{code-and-fix},\textbf{Group1}\}} + \sum \textbf{Task Duration}_{\{\textbf{code-and-fix},\textbf{Group 1}\}}$. With this answer, the null hypothesis $H0_{11}$ can be rejected and the alternative hypothesis $H1_{11}$ "The use of ArchWiSeN is more efficient to the code-and-fix development in terms of development time" is accepted. Therefore, the answer to this question is: The obtained results indicate that using ArchWiSeN costs less time to the developers perform the WSAN programming tasks and then using the code-and-fix approach independently of which approach is used first.

Q2. How effective are the available UML profiles in ArchWiSeN to deal with application development?

The average value results for each question related to $\mathbf{M_2}$ are shown in Fig. 13. The high values of $\mathbf{M_2}$ for ArchWiSeN in all questions mean that the participants consider that it is very easy to read and understand the application's requirements from the available models. Therefore, the null hypothesis H0$_{13}$ can be rejected and we can accept the alternative hypothesis H1$_{31}$ "The use of ArchWiSeN increases the understanding of the system compared to the code-and-fix approach". Therefore, the answer to this question is: *The use of models and profiles provided the developers with an abstraction level that was inexistent in the code-and-fix approach. Such abstraction level positively helped the developers to comprehend the applications presented in this experiment. Thus, we can consider that the proposed UML profiles are effective to properly abstract domain and network requirements.*

Q3. Do WSAN developers state that ArchWiSeN is easy to use?

For each question, the average values for $\mathbf{M_3}$ (Fig. 14a) were 4.7, 4.5, 4.2, 3.6, 4.6, and 4.5, respectively. Considering all the cases, it is possible to assume that the participants consider ArchWiSeN easy to use. Therefore, the answer to this question is: *In this experiment, the developers state that the ArchWiSeN is an approach that is easy to use.*

Q4. Do WSAN developers state that ArchWiSeN aids them to develop a WSAN application?

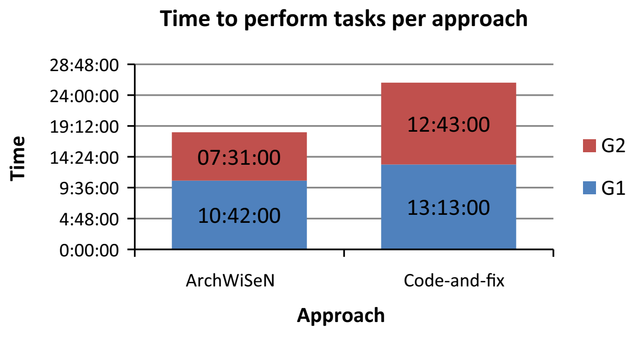

Fig. 12 Chart showing the sum of time required to perform each task by each approach

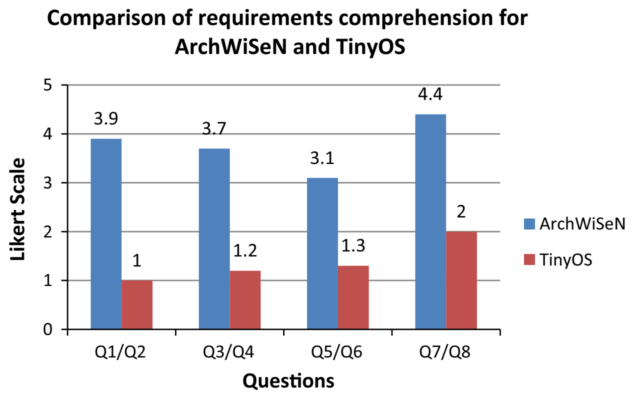

Fig. 13 Chart comparing the answers given by the participants about their comprehension using ArchWiSeN and TinyOS

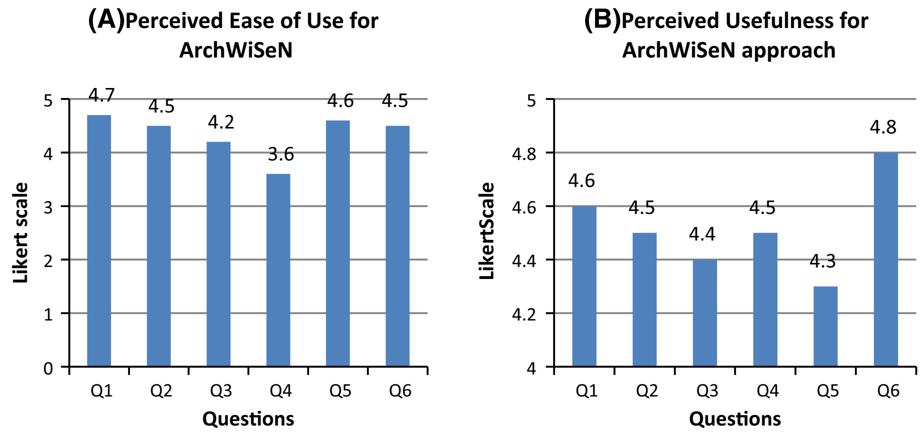

Fig. 14 **a** Chart for the perceived ease of use answers given by the participants. **b** Chart for the perceived usefulness answers given by the participants

For each question, the average values for $\mathbf{M_4}$ (Fig. 14b) were 4.6, 4.5, 4.4, 4.5, 4.3, and 4.8, respectively. Considering all the cases, it is possible to assume that the participants consider ArchWiSeN useful. The values of $\mathbf{M_4}$ in Q1, Q2, Q3, and Q4 allow the null hypothesis $H0_{12}$ to be rejected and the alternative hypothesis $H1_{12}$ "The use of ArchWiSeN increases the efficiency in the production of software artifacts compared to the code-and-fix approach" to be accepted. The answer to this question is: *In this experiment, the developers state that they consider ArchWiSeN useful to develop WSAN applications.*

Q5. How effective are the UML mechanisms used in ArchWiSeN to deal with the implementation of new requirements?

The obtained value $\mathbf{M_5}$ for the ArchWiSeN approach was 00:11:30 hours, while with the code-and-fix approach the obtained value was 00:22:12 hours. Since $\sum \textbf{Reuse effort}_{\{\textbf{ArchWiSeN}\}} < \sum \textbf{Reuse effort}_{\{\textbf{code-and-fix}\}}$, the reuse effort for the implementation of new requirements with the ArchWiSeN approach is lower than using the code-and-fix approach. In other words, it takes less time to implement new requirements with ArchWiSeN when compared to the code-and-fix approach. The answer to this question is: *The developers took less time to perform the reuse tasks with ArchWiSeN.*

Q6. Do WSAN developers state that ArchWiSeN aids them to reuse a WSAN application?

The results for $\mathbf{M_6}$ were inconclusive. Therefore, the answer obtained in question Q6 does not help to evaluate the null hypothesis. However, considering the answer obtained in Q2 and Q5 the null hypothesis $H0_{21}$ can be rejected and the alternative hypothesis $H1_{21}$ "The use of ArchWiSeN increases the reuse of components compared to code-and-fix approach" can be accepted if we consider the reduce effort for reuse and that a better comprehension can help developers to maintain and reuse a WSAN system. The answer to this question is: *The developers did not declare that ArchWiSeN helps to reuse an application.*

3.1.9 Threats to validity

We can enumerate three aspects as threats to the validity of the performed experiment: construct validity, internal validity, and external validity.

Construct validity means that the independent and dependent variables accurately model the abstract hypotheses [33]. This threat was addressed in our work: (1) by using the GQM approach to organize the necessary steps to determine the links between the experiment goals and the hypothesis testing; (2) by using the strategy proposed by [34] (concept, design, preparation, execution, and analysis) to design and execute the software engineering experiment; and (3) by using standardized questions [9] and scale [23] to measure the participants opinions. Moreover, some of the defined metrics have a certain subjective degree, based on the judgment of the participants. To address this threat, the questionnaire was formulated considering the participant's opinion in a Likert scale [23].

Internal validity means that changes in the dependent variables can be safely attributed to changes in the independent variables [33]. In the conducted experiment, the dependent variables are (1) *development effort*, (2) *requirements comprehension,* (3) *perceived ease of use*, (4) *perceived usefulness,* (5) *reuse effort*, and (6) *perceived reuse*. They all are strictly related to the set of metrics defined specifically for this experiment. Regarding internal validity, we identify the following threats:

(A) Differences between participants: Different levels of experience with UML and NesC programming language could lead to biased results. We tried to minimize this threat by including a training session on such languages.
(B) Differences between tasks: Five different tasks were presented to the participants. We mitigated this threat with the design of the experiment, in which each group worked over two laboratory sessions, alternating the different tasks with random order. The survey question-

naire showed the participants completely understood their tasks.

(C) Fatigue effects: The experiment was organized in two daily sessions of 5 hours. During the experiments, the participants were observed by a researcher and no signs of fatigue were detected because the participants were free to leave the laboratory once a Task was completely done.

(D) Experiment bias: The experiment design could introduce a learning process that affects the participant opinion. Moreover, the participant previous information about the used approach could lead to biased results. These threats were addressed by a consistent experiment planning phase supported by the available literature. The participants were randomly organized: (1) by treatments, (2) tasks,; and (3) computers. As explained in [34], by randomly assigning treatments to experimental units it is possible to keep some treatment results from being biased by sources of variation over which you have no control. Moreover, the randomization of tasks order also contributes to reduce the impact of a possible learning process since no participant will perform the tasks in the same order with the same treatment.

(E) Tester and participant background influence: The tester opinion or the participant's background information about the experiment execution could lead to biased results. This threat was addressed by not involving the participants with the experiment in any other level than the experiment execution. The participants and the tester were not classmates; some of them were only under supervision of the same professor. The participants were invited only knowing that the experiment would involve wireless sensor networks and a software engineering technology. Furthermore, neither the participants nor the tester could know how the tasks were organized before the experiment execution due the randomly assignment.

External validity means that the study's results generalize to settings outside the study [33]. It is necessary to perform experiment replication with different tasks to confirm or contradict the results. However, the process and materials presented in this paper are enough to replicate the experiment.

3.2 Proof-of-concept

This subsection illustrates the use of ArchWiSeN to build several WSAN applications as a proof-of-concept to evaluate the proposed development process. In this subsection, we assess the ArchWiSeN's goal "to define a process to build WSAN applications where developers can benefit of better productivity, comprehension, separation of concerns, and reuse." While in Sect. 3.1 the focus was to quantify productivity, comprehension, and reuse, here in Sect. 3.2 we focus on depicting the process behind the numbers achieved in the controlled experiment and to clarify how each ArchWiSeN's artifact is used during the process so as to create a new application. This proof-of-concept also demonstrates how the separation of concerns is achieved when using the application engineering process. For this PoC, we developed all the applications presented in Table 6. However, in this paper we describe only the application engineering process the most complex application, namely the *structural health monitoring* (SHM), for the smart building domain followed by two scenarios of changes where we illustrate how developers should proceed when the application's requirements are modified.

Finally, in Sect. 3.2.2 we discuss the results through a qualitative analysis. All developed applications will be summarized in Table 6 describing the number of lines of code considering the TinyOS as the target platform for this proof-of-concept.

Table 6 Proof-of-concept applications

Application (only considering the node)	Generated LoC
Smart home for lighting and temperature control	98
Forest fire detection	56
Oscilloscope	109
Perimeter access control	112
Supply chain control	73
Control of humidity and temperature	109
Remote home control	63
Monitoring of volcanoes	72
Fall detection	140
Structural health monitoring	174

3.2.1 Structural health monitoring application description

The focus of structural health monitoring (SHM) applications is to detect and localize damages in civil structures. All structures react to vibrations, either forced or caused by the environment (natural). Natural vibrations can be generated by earthquakes, winds, moving vehicles, waves, among others, while forced vibrations are generated by shakers and other devices.

The work presented in [37] proposes a localized SHM algorithm supported by multilevel information fusion techniques to enable detection, localization, and extent determination of damage sites. We describe the requirements of this SHM application.

The application works with three different types of devices: sensor nodes, cluster heads (CH), and sinks. As described in [37], the SHM application operates following five stages (setup, data collection, extract peaks, damage detection, and report). In the setup stage, the network is configured and the sensors collect the accelerometers data of the monitored structure. Such initial reading, also known as *initial signature*, consists in the response of a "healthy" structure and is used to compare to further sensors measurements. Each sensor is responsible for sensing the structure during a data collection stage, which starts with the sink node transmitting messages to the CHs. CHs request the sensing of the structure by the sensors in their respective clusters. Then, each sensor node collects the acceleration measurements relative to its physical position. Such data collection is represented in the time domain. After that, a *fast Fourier transform* (FFT) is performed by each sensor over the collected acceleration signals to convert measurements to the frequency domain. Next, a method for extracting frequency values from the peaks of the power spectrum generated by the FFT is used. At later stages, each CH also receives the subsequent signatures, denoting accelerometer information provided by the sensors of the structure. CHs are responsible for performing the damage detection, determining the damage location and extent through the calculation and analysis of damage coefficients. Each CH, after collecting the signatures from all sensor nodes of its cluster, performs a comparison between these values and the respective initial signatures from the respective sensors, to check whether the structure is damaged or it has been temporarily changed due to some external event.

Following the process shown in Fig. 11, both developers (domain and network experts) perform "Requirements analysis," generating as output the MDA CIM (computation-independent model), a document that describes the domain and the requirements of the system. This activity can be done using any existing requirements analysis technique. The process is divided into parallel activities that will be performed by each developer within his/her area of knowledge. The domain expert starts the "Model PIM" activity to create a UML activity diagram that best represents the required WSAN application.

Figure 15 illustrates the actions specified by the domain expert to be executed by the sensor nodes running the SHM application. At the initialization of the application, all nodes will perform their radio configuration (cluster head and sink not shown in figure). Then, the sensor will wait for some messages sent either by the cluster head or by the sink to perform its tasks that are: (1) sample the accelerometer and perform the *fast Fourier transform* function; (2) send a message with the frequency sample data; and (3) send a message with the accelerometer sample data. Then, the domain expert will model the application's structural view.

Figure 16 illustrates the model developed by the network expert representing the structural view for the application's PIM. The described application works with two different regions, TargetArea and ControlRoom. The TargetArea region represents the building floor where the sensor nodes and cluster heads will be deployed, while the ControlRoom region represents the room where the sink node is deployed. The TargetArea region contains two *Nodegroups* (sensor and cluster head) and the ControlRoom region contains one *Nodegroup* called sink. This division between *Nodegroups* allows the network expert to define which type of hardware will be used at each different group and how they communicate with each other. As a parallel activity, the network expert analyzes the CIM model and chooses which sensor platform will best fit the requirements of the target application. In this case study, we consider that TinyOS was chosen. Finally, both developers will perform the "Verification" activity. Then, the transformations will be executed to generate the application's source code.

Scenario of Change 1 Changing the network-level protocols. A change in the network logic topology or in the nodes density (e.g., the need to deploy additional nodes or to remove nodes from the network, thus increasing/decreasing its density) may require the selection of a new communication protocol to run on the sensor nodes. By using the proposed approach to address such need, the network expert must make changes to the generated PSM. This process is carried out during the activity "Refine Model." In order to modify the protocol used by the model representing the application configuration (The TinyOS component), which has been automatically generated by the MDA infrastructure, three steps are required: (1) to remove the components that are declared, but are no longer used, referring to the former communication protocol, (2) to add the new components that implement the new protocol, and (3) to carry out the wirings between the components and interfaces being used. This scenario shows that when the required change occurs at the network level, all the domain-related aspects remain unchanged and are transparent to the network expert. In addition, these changes are carried out without the participation of the domain expert.

Scenario of Change 2 Changing in the application-specific requirements. Such changes should be handled by the domain expert and occur at the PIM level. A typical case is the change in application QoS parameters. Such requirement can be interpreted as a change in the adopted energy policy or a change in the data transmission rate. To modify the energy policy, the domain expert should increase the value

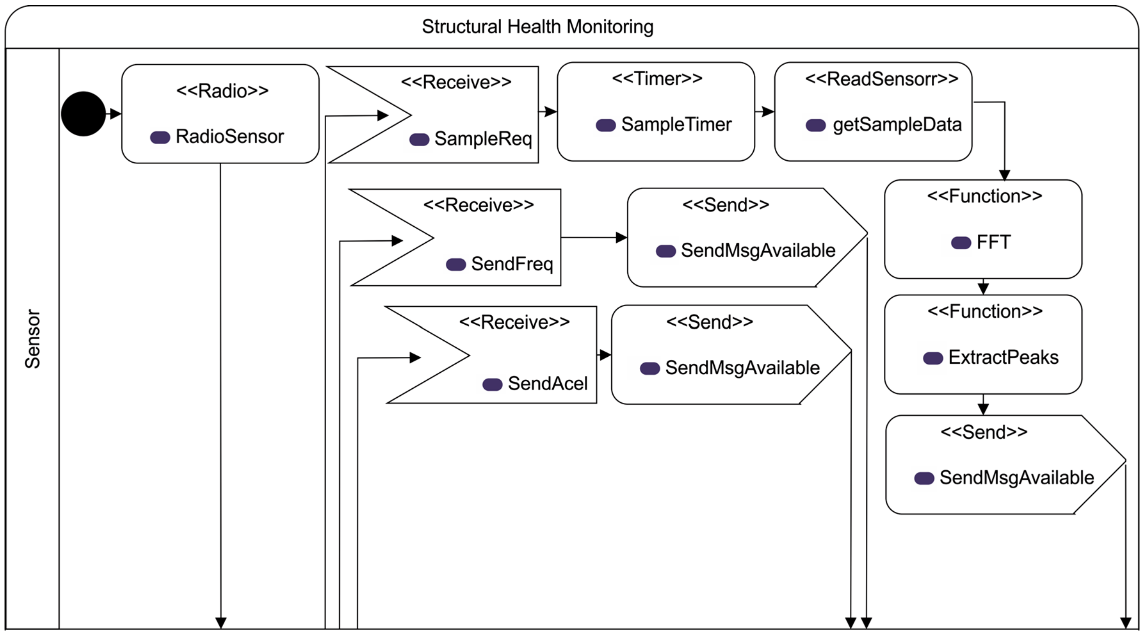

Fig. 15 SHM activity diagram excerpt showing the actions performed by the sensor node

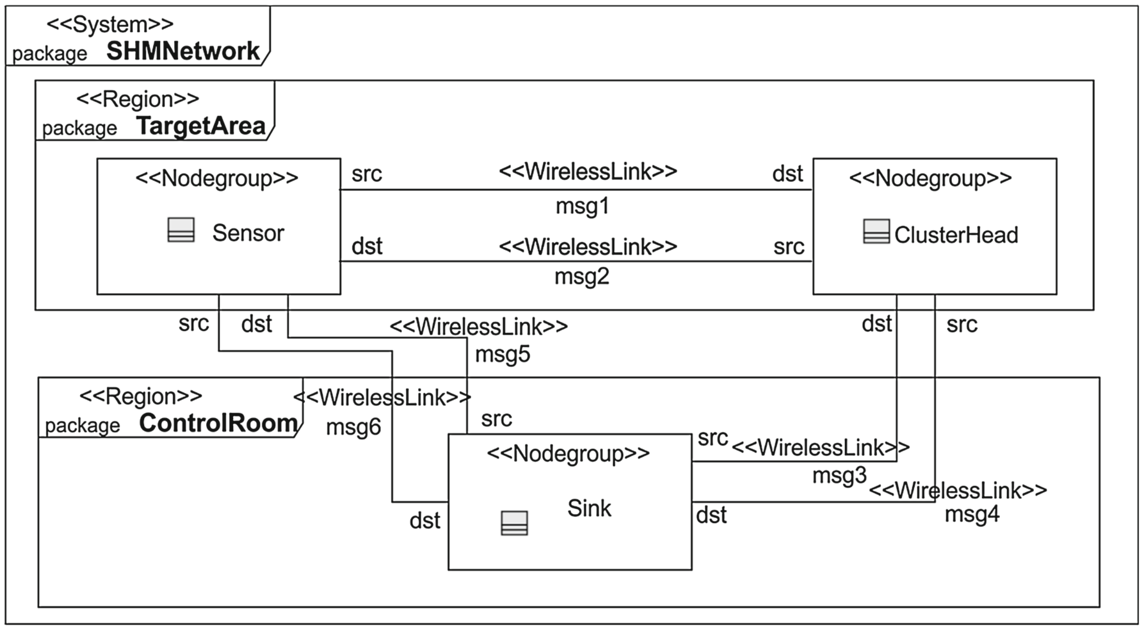

Fig. 16 Structural view of the SHM application

of the RadioConfigUnit parameter to change the duty cycle of the nodes and extend the interval of the radio sleep timer. To change the data transmission rate, the domain expert can simply change the interval value of the timer that controls the waiting time before each data transmission. Thus, the M2M transformation will generate the PSM model using the settings that match the choice made by the domain expert. All these changes can be made without requiring the participation of the network expert.

3.2.2 Qualitative analysis

As previously mentioned in Sect. 3.2, ten applications were developed in order to complete this proof-of-concept. Although the application engineering process is described only one application, all the ten applications were modeled and implemented as part of the proof-of-concept. Table 6 summarizes the data obtained during the execution of the application engineering process for each of these applications. It can be noted that ArchWiSeN was able to generate code for applications in many different domains while preserving the separation of interests among the developers involved in building application process. The generated code can hardly be divided into the pieces of code representing application's logic, configuration, and routing, since such parts are mostly scattered throughout all the TinyOS's *events* and *commands* implementations. In fact, such a characteristic of scattering application logic in the code, while coupling application code with lower-level protocol code, is one of the limitations of traditional programming for WSAN platforms and which motivated our proposal.

Through this proof-of-concept, it is evident that the approach proposed in this paper can be successfully applied for WSAN application development. Thus, in this proof-of-concept we demonstrated through the development of different WSAN applications the process to use ArchWiSeN for generating all the necessary code while preserving the separation of interest among developers even in situations when changes of parameters are required.

4 Related work

We start this section comparing our approach with two others that share our main goal (that is, to facilitate building WSAN applications) but use a different approach, namely *software product lines* (SPL). In [17] the authors present a process for the SPL configuration of an AmI (Ambient Intelligence) middleware. In such approach, in order to build a custom middleware to perform a specific task, it is only necessary to define a set of high abstraction features and a few parameters. Using this process, the approach presented in [17] is able to generate all the source code required to run a target application in the J2ME platform. Unlike our approach, in [17] there is not a separation of concerns between the application developers involved. Moreover, in the proposed process and feature model there is no explicit separation between domain and network features. In the work described in [6], the authors present an SPL approach for generating transparent autonomic systems to the end user. For the development of such systems, the authors use a model-driven development approach following the variability modeling principles. The authors also use a simple example of Smart Homes to analyze their approach. Despite the success of the work described in [6], the authors do not present techniques for exploiting reuse of models, and they only focus on the application maintenance. Moreover, features like communication and routing data are not mentioned for the configuration of the wireless sensor network, making it unclear how this information could be implemented and deployed in the nodes running the application.

Baobab [1] is a framework based on meta-modeling for designing WSN applications and generating the corresponding source code. A component-based approach is adopted as the (*domain-specific languages* (DSL) and the WSAN application is interpreted as a set of components that interact via an event-based push model. The transformation of the model into executable code and the generation of the middleware are performed by a template-based code generator. Baobab allows users to separately define functional and non-functional aspects of a system (as software models), to validate them, and to generate the source code. Although Baobab provides an independent platform approach, it is not designed to enable the development of WSAN applications without requiring specific platform knowledge. Thus, the separation of concerns between developers can be affected mainly by the need for WSAN-specific knowledge by the domain expert. Moreover, despite to be open for other platforms, the authors do not demonstrate how the procedure to extend Baobab is done, or which methodology should be adopted in case of the need to implement new software artifacts for the Baobab framework. Differently from such a work, our proposal does not adopt a component-based solution. Instead, ArchWiSeN was designed to be extended through the domain engineering process using UML profiles and it is agnostic to the adopted design solution or patterns. Our approach was created to support the addition of multiple platforms by following the domain engineering process. The component-based approach used in Baobab does not separate components in a platform-independent way, making future extensions to other platforms to require a complete reconstruction of all components. Moreover, it do not address the separation of concerns between the different types of developers involved in WSAN application building. Thus, the development process may require knowledge that is not from the developer's expertise area.

Make-sense project [11] aims to improve WSN programming by allowing developers to express their applications mainly via the usage of business process modeling. Its main goal is to break the barrier for adoption of WSNs in real-world applications, notably in business processes. Such project has similar goals with ours but do not specify which methodology is employed to achieve such goals.

In [39] the authors propose a WSAN development process with three different models according to the granularity of specification: (1) network, (2) group of nodes, and (3) individual node. According to the authors, this separation has been proposed because in WSAN development it is necessary to build or modify a low-cost prototype in order to optimize performance, and such feature cannot be solved by a single model. Although the construction of various models is proposed in [39], such models describe only static features and are not able to describe the application behavior, which is a crucial part of the WSAN system modeling. Moreover, the authors did not consider the participation of developers with different expertise areas and excluded several important features of lower abstraction level specification, such as the communication protocol, the network topology, and the device types.

In [16] the authors present ACOOWEE, a framework where Sun SPOT sensors can be programmed to perform collaborative tasks by setting a sequence of tasks through UML activity diagrams extended by a profile. Despite the similarity with our proposed MDA approach, there are differences in the presentation of different views where experts should act during the application development process. ACOOWEE framework does not separate the model elements in different views; it only includes the view where the developers indicate the application behavior. No information about how the nodes are organized or the differences between nodes capabilities can be modeled. Moreover, our MDA approach uses an independent platform level, which enables portability of the application modeled across different platforms.

Since WSAN can be considered as a specialization of *distributed, real-time, and embedded system (DRE),* it is also important to compare our approach with other works that are related to this type of systems. DRE and WSAN systems share the same issues regarding heterogeneity, hardware limitations, and the low-level programming languages available to developers. Middleware platforms are often used as solution to tackle these issues in DRE systems; however, choosing the right middleware can also be a challenging problem.

In [5] the authors present an MDA approach using UML for the specification of WSAN applications for TinyOS platform. The presented approach uses and extends the *modeling and analysis of real-time and embedded systems* (MARTE) profile. Although our proposal also uses views to separate the concerns within the WSAN profile, the inclusion of a network configuration view to adapt the applications to the chosen platform provides more modeling capabilities. Moreover, in [5] there is no separation of concerns between domain and network characteristics. This feature carries the platform concepts to the application engineering process, thus requiring that the developers have knowledge of both domain and network aspects.

CoSMIC [38] is an MDA generative tool for DRE (distributed real-time and embedded) middleware and applications. The CoSMIC tool allows developers to model their applications in a high abstraction level and to generate a middleware specifically customized for the target application while respecting the defined QoS requirements. By using CoSMIC tool, developers can achieve the reduction in the lifecycle costs of complex DRE applications. Another feature in the CoSMIC tool is the capability of ensuring that requirements are met and to validate system structure and/or behavior during the application design phase. Differently from CoSMIC, ArchWiSeN separates the development models in two different levels (PIM and PSM). The PSM level in ArchWiSeN allows the portability of the approach potentially for any current of future WSAN platform. The capability to generate middleware components available in CoSMIC can be implemented in ArchWiSeN due the flexibility to introduce any new platform, even at the middleware level. Actually, our research group is working in a future project to include the modeling of middleware platforms in the ArchWiSeN approach. ArchWiSeN also offers design-phase validation tools that were presented in [35].

The work described in [21] proposes a development process for distributed applications where the middleware is considered the core element. It also extracts three non-functional requirements of middleware and proposes a middleware-based distributed systems software process. The proposed software process consists of five phases: requirements analysis, design, validation, development, and testing. Such an approach leverages the combined use of component-based software engineering, separation of concerns, model-driven architecture, formal methods, and aspect-oriented programming in the context of a new software development method. Differently from such an approach, ArchWiSeN proposes two different processes: (1) the first process consists in building the MDA infrastructure itself, allowing future developers to introduce new platforms to allow code generation; (2) the second process consists in building a specific application using the related MDA infrastructure. ArchWiSeN clearly specifies how each developer contributes to the application development. Moreover, as mentioned before, ArchWiSeN is open to be integrated with middleware platforms.

The paper presented in [22] describes a model-integrated approach (also called model-integrated computing—MIC) for embedded software development based on domain-specific, multi-view models that are used in all phases of the development. As pointed in [22] the use of a meta-modeling approach is very powerful, but changes in meta-models often invalidate existing domain models, thus requiring extensive rebuilding. With the use of UML profiles as the extensibility mechanism in ArchWiSeN, it is possible to make changes in the PIM or PSM meta-models without invalidating any previously defined model thus circumventing such drawback.

5 Conclusions and future work

The use of an MDA approach can benefit the developers regarding effort, reuse, and maintainability. Our approach has shown that such benefits are present in the context of WSAN applications enabling the rapid development even when considering multi-domain and multi-platform applications and providing an appropriate tool chain for application development. As stated by Fowler [15] "one of the themes that winds constantly across both forms of language oriented programming is the involvement of lay programmers: domain experts who are not professional programmers but program in DSLs as part of the development effort" which we can easily correlate with the motivation of our approach, where domain experts are responsible for developing WSAN and do not often have the platform-specific knowledge. The advantage of using a DSL approach is to drop all the baggage of your host platform and present something that is very clear for the domain expert [15]. Moreover, the lack of communication is often the biggest roadblock in software development projects [15]. Therefore, clearing the need of an application engineering approaches improving the communication between developers and achieving the separation of concerns.

In this paper, we advocated that the OMG's standard for the MDD approach (the model-driven architecture) is indicated for the WSAN due its intrinsic characteristics as multi-domain applicability and the use of different hardware platform with different capabilities. We have shown how the use of a model-driven approach aids in the development of WSAN applications by increasing the level of abstraction and enabling the automatic generation of most of the code needed to implement and execute the application. In addition, building WSAN systems using our approach allows the division of responsibilities among developers, allowing them to use their specific knowledge and avoiding the need of learning about requirements that do not belong to their expertise field. With our approach, domain experts are able to create WSAN applications without having specific knowledge on WSAN platform and using only UML models to define their application. It is worth noting that our approach is particularly useful for WSANs given the high heterogeneity and constant technological changes typical of such environment. The results obtained from the performed evaluation indicate that our approach achieves its goals by offering to the developers an infrastructure and an application engendering that augments productivity, comprehension, and reuse in WSAN application development.

Our work also showed how the lack of a methodology can affect the development effort, comprehension, and reuse in WSAN application development. The addition of a well-structured software engineering process can greatly benefit the WSAN domain. The use of an MDA approach works perfectly to the WSAN domain because it creates a high abstraction level where developers can benefit from different views from the domain and network information. Our MDA approach is also open for existing and future WSAN platforms, requiring only the specification of platform-specific models and transformations to port any already developed application to a new emergent platform.

As future directions of our research, we intend to implement new functionalities to allow the model analysis during design phase and help developers in the application validation process. Since platform-independent models encompass the WSAN physical structure and behavior, they can be used to make calculations with the defined model elements or properties to enable the analysis of non-functional requirements during the WSAN, or more generically DRE, system design. With this information in a very early stage of development, developers would be able to make better choices when defining their applications. Moreover, our approach currently does not tackle the issue of synchronization between models. We plan to implement a traceability mechanism between models in future versions benefiting from our domain engineering process. The lack of bi-directional transformations can impact the reuse of the models as the manually added properties are lost if the M2M transformation is performed. Such feature was initially considered as future work of our approach. Finally, the implementation of the support of a middleware technology in our MDA solution is a very promising alternative to define a customizable middleware that could be used for WSANs or DREs. Since middlewares often require a high amount of resources, such as memory and processing power, it remains unlikely to install in the nodes of a WSAN a generic middleware that could provide all common or domain-specific services. Thus, in general, developers are required to select only the essential services manually, which is a complex process that affects the development time and the level of reuse. For a future work, we intent to enable the ArchWiSeN to perform the automatic generation of customized middleware specially tailored to the context of the application modeled by developers.

The analysis of the correctness of the PIM model prior to the transformation process can bring benefits to developers to indicate whether a developed model is semantically correct before generating the source code. For example, an activity defined in the PIM model with the «readSensor» stereotype requires the definition of at least one data type associated with the reading of this sensor. Although if such data are not specified by the developer, the program is likely to be incompletely generated or with errors. Such analysis can be implemented by defining UML restrictions through the use of object constraint language (OCL) that is compatible with the UML version presented in this work.

Concerning the reuse metric used in the performed experiment, the answers obtained with the participants questionnaires did not help to pinpoint the problem. However, the time needed to execute the reuse tasks was lower with the ArchWiSeN approach. Such results indicate that a higher abstraction level approach can benefit the developers to add or remove new functionalities of some applications even when there is no mechanism to keep sync between models and source code. We intend to repeat our experiment in the future to try determining the causes of these results.

Finally, although our goal in this work was defining an instance of an MDA-based development process, specifically tailored for building generic WSAN applications, instead of defining a reusable process framework for this domain, we envision that, as the proposed software process evolves by incorporating the lessons learned over time, it will be important to use a standard meta-process (such as Eclipse Process Framework [13]) for specifying such evolution in a systematic way.

Acknowledgements This work was partly supported by the Brazilian funding agencies CAPES, CNPq, and FAPERJ. Flávia Delicato, Paulo Pires, Thais Batista and Luci Pirmez are CNPq Fellows.

References

1. Akbal-Delibas, B., et al.: Extensible and precise modeling for wireless sensor networks. In: Yang, J., et al. (eds.) Information Systems: Modeling, Development, and Integration. Springer, Berlin (2009)
2. Andreas Jedlitschka, M.C.: Guide to Advanced Empirical Software Engineering. Springer, London (2008)
3. Basaran, C., Kang, K.-D.: Quality of service in wireless sensor networks. In: Misra, S.C., et al. (eds.) Guide to Wireless Sensor Networks. Springer, London (2009)
4. Basili, V.R., et al.: Encyclopedia of Software Engineering. Wiley, Hoboken (2002)
5. Berardinelli, L., et al.: Modeling and analyzing performance of software for wireless sensor networks. In: Proceeding of the 2nd Workshop on Software Engineering for Sensor Network Applications—SESENA '11, p. 13. ACM Press, New York, New York, USA (2011)
6. Cetina, C., et al.: Applying software product lines to build autonomic pervasive systems. In: 2008 12th International Software Product Line Conference, pp. 117–126 (2008)
7. Cronbach, L.J.: Coefficient alpha and the internal structure of tests. Psychometrika **16**(3), 297–334 (1951)
8. Czarnecki, K.: Unconventional Programming Paradigms. Springer, Berlin (2005)
9. Davis, F.D.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Q. 13, 3, 319 (1989)
10. Delicato, F., et al.: Variabilities of wireless and actuators sensor network middleware for ambient assisted living. In: Proceedings of the International Work Artificial Neural Networks Part II Distributed Computing, Artificial Intelligence Bioinformatics, Soft Computing, Ambient Assisted Living, vol. 5518, pp. 851–858 (2009)
11. Easy Programming of Integrated Wireless Sensor Networks. http://www.project-makesense.eu/. Accessed on 20 May 2014
12. Eclipse Project: Acceleo. http://www.eclipse.org/acceleo/
13. Eclipse Process Framework Project (EPF). http://www.eclipse.org/epf/
14. Eclipse Project: Papyrus. http://www.eclipse.org/papyrus/
15. Fowler, M.: Language Workbenches: The Killer-App for Domain Specific Languages? http://martinfowler.com/articles/languageWorkbench.html
16. Fuchs, G., German, R.: UML2 activity diagram based programming of wireless sensor networks. In: Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications—SESENA '10, p. 8. ACM Press, New York, NY, USA (2010)
17. Fuentes, L., Gámez, N.: Configuration process of a software product line for Ami middleware. J. Univers. Comput. Sci. **16**, 1592–1611 (2010)
18. Gay, D., et al.: The nesC language. ACM SIGPLAN Not. **38**(5), 1 (2003)
19. Gliem, J., Gliem, R.: Calculating, interpreting, and reporting cronbach's alpha reliability coefficient for likert-type scales. In: Proceedings of the Midwest Research to Practice Conference in Adult, Continuing, and Community Education, pp. 82–88. Ohio State University, Columbus, OH (2003)
20. Harris, Peter: Designing and Reporting Experiments in Psychology. McGraw-Hill International, New York (2008)
21. Jingyong, L., et al.: Middleware-based distributed systems software process. In: Proceedings of the 2009 International Conference on Hybrid Information Technology—ICHIT '09, pp. 345–348. ACM Press, New York, NY, USA (2009)
22. Karsai, G., et al.: Model-integrated development of embedded software. Proc. IEEE. **91**(1), 145–164 (2003)
23. Likert, R.: A technique for the measurement of attitudes. Arch. Psychol. **22**(140), 1–55 (1932)
24. Losilla, F., et al.: Wireless sensor network application development: an architecture-centric MDE approach. In: Proceedings of the First European Conference on Software Architecture, pp. 179–194. Springer, Berlin, Heidelberg (2007)
25. MEMSIC: MICAz Datasheet. http://www.memsic.com/wireless-sensor-networks/
26. Miller, J., Mukerji, J., (eds.): MDA Guide Version 1.0. 1. Object. Management Group, Needham (2003)
27. Model Driven Architecture (MDA). http://www.omg.org/mda/. Accessed on 20 May 2014
28. Nunnally, J.C.: Psychometric Theory. McGraw-Hill, Michigan (1978)
29. OBEO, INRIA: ATL—A model transformation technology. http://www.eclipse.org/atl/
30. Object Management Group: Model Driven Architecture Guide rev. 2.0. http://www.omg.org/cgi-bin/doc?ormsc/14-06-01
31. OMG: Unified Modeling Language (UML). http://www.uml.org/
32. Oracle: Sun SPOT. http://www.sunspotworld.com
33. Perry, D.E., et al.: Empirical studies of software engineering. In: Proceedings of the Conference on the future of Software engineering—ICSE '00, pp. 345–355. ACM Press, New York, NY, USA (2000)
34. Pfleeger, S.L.: Experimental design and analysis in software engineering. Ann. Softw. Eng. **1**(1), 219–253 (1995)
35. Rodrigues, T., et al.: Model-driven approach for building efficient wireless sensor and actuator network applications. In: Proceedings of the 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA), pp. 43–48. IEEE (2013)
36. Rodrigues, T., et al.: Model-driven development of wireless sensor network applications. In: Proceedings of the 9th International Conference Embedded and Ubiquitous Computing, pp. 11–18 (2011)
37. Dos Santos, I.L., et al.: A localized algorithm for structural health monitoring using wireless sensor networks. Inf. Fusion **15**, 114–129 (2014)
38. Schmidt, D., et al.: CoSMIC: an MDA generative tool for distributed real-time and embedded component middleware and applications. In: Proceedings of the OOPSLA 2002 workshop on generative techniques in the context of model driven architecture. ACM, Seattle, USA (2002)
39. Shimizu, R., et al.: Model driven development for rapid prototyping and optimization of wireless sensor network applications. In: Proceeding of the 2nd Workshop on Software Engineering for Sensor Network Applications—SESENA '11, p. 31. ACM Press, New York, NY, USA (2011)
40. Telos. http://www.tinyos.net/scoop/special/hardware#telos. Accessed on 20 May 2014
41. Uml-diagrams.org: UML profile diagram is a structure diagram which describes UML extension mechanism by defining custom stereotypes, tagged values and constraints. http://www.uml-diagrams.org/profile-diagrams.html#extension

42. Wada, H., et al.: Modeling and executing adaptive sensor network applications with the Matilda UML virtual machine, pp. 216–225 (2007)
43. Yick, J., et al.: Wireless sensor network survey. Comput. Netw. **52**(12), 2292–2330 (2008)

Taniro Rodrigues received his PhD degree in Computer Science from Federal University of Rio Grande do Norte (UFRN), Brazil, in 2015. His research interests involve wireless sensor networks, model-driven development, and ubiquitous systems.

Flávia C. Delicato is an associate professor of computer science at Federal University of Rio de Janeiro (UFRJ), Brazil. She is the author of more than 100 papers and participates in several research projects with funding from International and Brazilian government agencies. Her research interests include wireless sensor networks, adaptive systems, middleware, and software engineering techniques applied to ubiquitous systems. She has a PhD in electrical and computer engineering from the Federal University of Rio de Janeiro. She is a level 1 Researcher Fellow of the National Council for Scientific and Technological Development.

Thais Batista is an Associate Professor at the Federal University of Rio Grande do Norte (UFRN), Brazil. She holds a PhD in Computer Science from the Catholic University of Rio de Janeiro (PUC-Rio), Brazil. She published more than 140 papers and she was supervisor of 24 MSc students and 5 PhD theses. Her current research interests involve internet of things, cloud computing, smart cities, software architecture, middleware, and model-based development.

Paulo F. Pires received his Ph.D. degree in 2002 from Federal University of Rio de Janeiro (UFRJ). He is an Associate Professor at UFRJ and integrates the Center for Distributed and High Performance Computing at the University of Sydney. His research interests are Ubiquitous Computing, Model-driven Development, and Software Architecture.

Luci Pirmez received her Ph.D. degree in 1996 from Federal University of Rio de Janeiro (UFRJ), where she is a researcher and professor of post-graduation courses in computer science. Her research interests include wireless sensor networks, network management, and security.