A4WSN: an architecture-driven modelling platform for analysing and developing WSNs

Malavolta, Ivano; Mostarda, Leonardo; Muccini, Henry; Ever, Enver; Doddapaneni, Krishna; Gemikonakli, Orhan

published in
Software and Systems Modeling
2019

DOI (link to publisher)
10.1007/s10270-018-0687-0

document version
Publisher's PDF, also known as Version of record

Software & Systems Modeling (2019) 18:2633–2653
https://doi.org/10.1007/s10270-018-0687-0

REGULAR PAPER

A4WSN: an architecture-driven modelling platform for analysing and developing WSNs

Ivano Malavolta[1] · Leonardo Mostarda[2] · Henry Muccini[3] · Enver Ever[4] · Krishna Doddapaneni[5] · Orhan Gemikonakli[6]

Received: 26 July 2016 / Accepted: 1 June 2018 / Published online: 17 July 2018

Abstract
This paper proposes A4WSN, an architecture-driven modelling platform for the development and the analysis of wireless sensor networks (WSNs). A WSN consists of spatially distributed sensor nodes that cooperate in order to accomplish a specific task. Sensor nodes are cheap, small, and battery-powered devices with limited processing capabilities and memory. WSNs are mostly developed directly on the top of the operating system. They are tied to the hardware configuration of the sensor nodes, and their design and implementation can require cooperation with a myriad of system stakeholders with different backgrounds. The peculiarities of WSNs and current development practices bring a number of challenges, ranging from hardware and software coupling, limited reuse, and the late assessment of WSN quality properties. As a way to overcome a number of existing limitations, this study presents a multi-view modelling approach that supports the development and analysis of WSNs. The framework uses different models to describe the software architecture, hardware configuration, and physical deployment of a WSN. A4WSN allows engineers to perform analysis and code generation in earlier stages of the WSN development life cycle. The A4WSN platform can be extended with third-party plug-ins providing additional analysis or code generation engines. We provide evidence of the applicability of the proposed platform by developing PlaceLife, an A4WSN plug-in for estimating the WSN lifetime by taking various physical obstacles in the deployment environment into account. In turn, PlaceLife has been applied to a real-world case study in the health care domain as a running example.

Keywords MDE · Software engineering · Software architecture · WSNs · Energy

1 Introduction

A recent study predicted that in 2020 there will be 50 billion devices connected to the Internet [11]. These are devices capable of performing various operations, such as sensing data, actuating on the external environment. With this perspective, WSNs are becoming an important part of a wide variety of applications and systems including environment

Communicated by Professor Gregor Engels.

✉ Leonardo Mostarda
leonardo.mostarda@unicam.it

Ivano Malavolta
i.malavolta@vu.nl

Henry Muccini
henry.muccini@univaq.it

Enver Ever
eever@metu.edu.tr

Krishna Doddapaneni
Krishna.c@altiux.com

Orhan Gemikonakli
o.gemikonakli@mdx.ac.uk

[1] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

[2] Computer Science Department, University of Camerino, Camerino, Italy

[3] Department of Information Engineering, Computer Science, and Mathematics - DISIM, University of L'Aquila, L'Aquila, Italy

[4] Computer Engineering, Middle East Technical University, Northern Cyprus Campus, Güzelyurt, Mersin 10, Turkey

[5] Wireless Innovation Networking Group, Altiux Innovations Inc., San Jose, USA

[6] Computer Design Engineering, Middlesex University, London, UK

monitoring, energy metering, smart cities, health care and intelligent houses [50].

Wireless sensor networks are composed of low-data-rate, low-cost and battery-operated wireless components called *sensor nodes*. A sensor node is a small digital device with communication, sensing, and limited processing capabilities. WSNs can be *event-driven* or *continuous* operation types. Event-driven WSNs report data to the base station (BS) only when certain events such as intrusion and fire detection occur. Continuous WSNs report data to the BS at regular intervals.

1.1 WSN challenges

The unique characteristics of WSNs introduce new challenges in various fields such as programming, security, and software engineering [7,36,47,57]. The text below points out what we consider the main drivers for this work:

Abstracting implementation details into a design view: the development of a WSN application requires skills across all levels of the communication stack. A WSN application developer has to be knowledgable about programming as well as different layers of the ISO/OSI reference model. Beside programming abstraction, abstracting an implementation view into an architectural design is a well-known need. As stated in [47], *"end users require high-level abstractions that simplify the configuration of the WSN at large, possibly allowing one to define its software architecture based on pre-canned component"*. Abstraction is fundamental for future WSN development, since sensors and WSNs in general are becoming important components in pervasive computing, and mobile systems, with new types of stakeholders (e.g. mobile systems engineers, developers) and reduced domain-specific technical skills. Under this perspective, approaches for abstracting the implementation details from the underlying hardware and physical infrastructure are strongly advised [7,36]. However, when current practices on WSNs are considered, the lack of engineering methods and techniques to manage these challenges is evident. Some initial effort has been made for architecting WSNs [21,27], and this paper goes along that line providing more advanced solutions. A thorough comparison with related work is provided in Sect. 7.

Increase reuse: State-of-the-art approaches mostly mix together software, hardware, and networking perspectives during the coding or design phase. Hardware and software components are locked and tied down to a specific type of nodes, thus hampering the reuse of source code and software components across different projects or organizations [36].

Early quality property assessment of WSNs: In traditional implementation-specific approaches, engineers might afford to take structural and behavioural decisions at deployment time. However, in WSN development it is important to take those sensible decisions as early as possible, enabling the predictive analysis of both functional and extra functional properties. This possibility becomes especially valuable in all those cases where the sensor nodes cannot be easily accessed once deployed (e.g. WSN nodes embedded into concrete walls or WSNs deployed in hostile environments).

In addition to challenges while designing and implementing a WSN, engineers and developers may face various concerns such as application energy efficiency [16], dependability,[1] coverage [22], networking and communication, and performance.

1.2 Our contribution

This article proposes A4WSN,[2] a model-driven engineering platform for modelling and analysing WSNs. A4WSN increases *separation of concerns* and *abstraction*, favouring the possibility of *reusing* software and hardware components across projects and organizations. It supports system engineers in the design of WSNs without requiring specialized knowledge about WSN low-level details. It favours the earlier, predictive, analysis of both functional and extra functional properties.

This article contributes to the state of the art on WSN engineering by:

- defining three modelling languages enabling the description of a WSN from different viewpoints. A4WSN, being a multi-view approach including different models which cover different concerns;
- providing a programming framework that enables the implementation of analysis and code generation plug-ins by third-party developers. The A4WSN programming framework supports extensibility and customization by design;
- developing the PlaceLife plug-in that analyses A4WSN models and automatically assesses the lifetime of the network. Its implementation provides evidence for the applicability of the proposed modelling approach and for the extensibility of its programming framework.

The platform is available at the project website.[3] It has been used in our software and system architecture courses for the past two years.

The rest of the paper is organized as follows. Section 2 provides the case study, while Sect. 3 overviews the A4WSN platform. Section 4 describes the proposed modelling languages for WSNs. Section 5 presents the programming framework. Section 6 focusses on the PlaceLife analysis

[1] http://www.dependability.org/.

[2] It stands for **A**rchitecting platform for (**4**) **W**ireless **S**ensor **N**etworks.

[3] http://a4wsn.di.univaq.it/.

plug-in. Finally, Sect. 7 presents related work, while the paper concludes in Sect. 8.

2 The healthcare system case study

In this paper the healthcare system case study is used as running example in order to help the reader in understanding the main concepts and design decisions considered when engineering the A4WSN modelling languages. The case study is used to show the effectiveness of the architectural approach and the PlaceLife plug-in.

Recent technological advancements in WSNs have opened up new prospects for a variety of applications, including healthcare systems [2,26]. WSN implementations on pervasive omputing-based healthcare systems avoid various limitations and drawbacks associated with the wired sensors providing a better quality of care, quicker diagnosis, more intense collection of information and at the same time keeping the cost and resource utilization to a minimum. Monitoring facilities introduced by using WSNs are particularly useful for early detection and diagnosis of emergency conditions, as well as keeping track of the diseases. WSN-based healthcare systems are also useful for providing a variety of health-related services for people with various degrees of cognitive and physical disabilities [2].

In the context described above, the case study (see Fig. 1) represents the concept of an in-hospital WSN that allows monitoring patients' conditions with the help of pulse oximeters. The monitoring system consists of two types of nodes: a monitoring station and seven oximeter nodes, forming a star network. Each pulse oximeter monitors a patient continuously, and a measurement is sent to the monitoring station every 3 s. In case the oximeter reads a value below a threshold, an alert message is sent to the monitoring system, and the system goes into a *warning* mode in which sensor readings are sent to the monitoring station more frequently (i.e. once every 200 ms), hence facilitating continuous monitoring of patients and allowing real-time responses in case of emergency conditions.

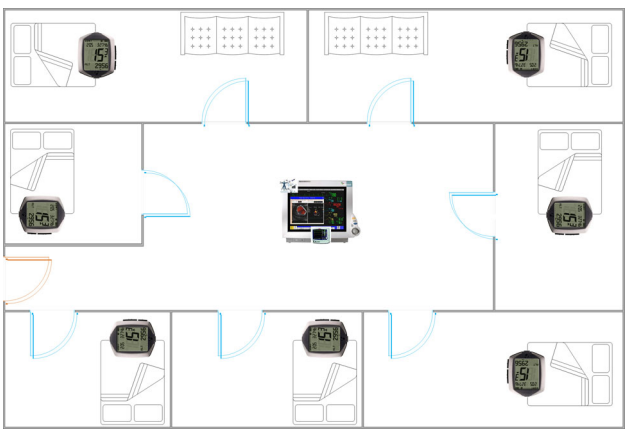


Fig. 1 Hospital scenario considered: (i) the central component represents the monitoring station, (ii) a pulse oximeter is included in each room around the central one

3 Overview of the platform

In this section we provide an overview of the A4WSN platform. This research takes advantage of MDE techniques to support an architecture-driven development and analysis of wireless sensor networks. Figure 2 shows the main components of the framework: the *WSN modelling environment* for describing the architecture of a WSN, and the *programming framework*. We emphasizes that we use the platform to denote the programming framework plus the modelling environment.

The **WSN modelling environment** exposes three modelling languages for describing specific architectural views of a wireless sensor network: the *Software Architecture Modelling Language for WSN* (*SAML*), the *Node modelling Language (NODEML)*, and the *Environment Modelling Language (ENVML)* (see Fig. 2).

- The **SAML** language focuses on the *application layer* of the WSN. It is used to break down the application into a set of software entities (e.g. components), to show how they relate to each other, to better reason about their distribution throughout the network, and to reason about the business logic of the WSN.
- The **NODEML** language concerns the low-level aspects underneath the application layer of the WSN. In this context, stakeholders reason about routing protocols, middleware, hardware configuration of the nodes, etc.
- The **ENVML** language is about the physical environment where the WSN will be deployed. This viewpoint could be specially useful for developers and system engineers when they have to reason about the network topology, the presence of possible physical obstacles (e.g. walls, trees) within the network deployment area, and so on.

The three proposed modelling languages are linked together via two auxiliary modelling languages. These languages are called Mapping Modelling Language (**MAPML**) and Deployment Modelling Language (**DEPML**), and they link together SAML to NODEML and NODEML to ENVML, respectively (see Fig. 2). More specifically, the MAPML modelling language weaves together an SAML model and a NODEML model. It allows designers to define a set of mapping links, each of them weaving together components in the SAML model and node definitions in the NODEML model. The DEPML modelling language weaves a NODEML model to an ENVML model. A DEPML model allow designers to consider each node type defined in the NODEML model and to *instantiate* it in a specific area within the physical environ-

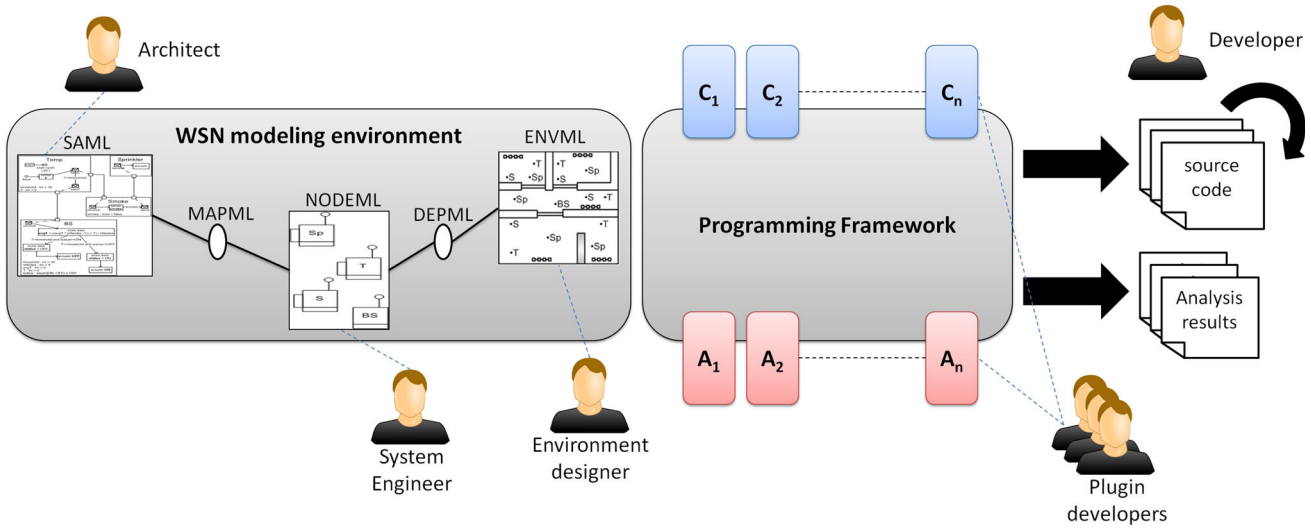


Fig. 2 Overview of the A4WSN platform

ment defined in an ENVML model. Each node configuration in NODEML can be instantiated *n* times within a specific area in ENVML with a certain distribution strategy.

The two auxiliary modelling languages are necessary for achieving a number of goals. Firstly, weaving models allow the engineers (i) to tame the complexity of architecting WSNs by ensuring the adoption of a clear *separation of concerns* and thus better concentrating their effort towards the concerns of more interest to each of them (e.g. a software architect can focus on the application layer in the SAML model, while a system engineer may focus on the nodes configurations in the NODEML model). Secondly, the auxiliary models allow engineers to (ii) make the SAML, NODEML, and ENVML models *reusable* across projects and organizations; this will open the possibility to, e.g. (i) reuse (parts of) the application layer defined in an SAML model across different systems deployed in different physical environments, (ii) build a catalogue of NODEML models which can be exploited for investigating how using different hardware configurations may impact the energy efficiency of the system as a whole, etc. Thirdly, the auxiliary models allow engineers to (iii) improve the *accuracy of model-based analyses* by considering a view that combines software, nodes, and environment of the WSN (such a combined view is automatically produced by A4WSN). In Sect. 6 we discuss in details a concrete case in which the integrated specification of software, hardware, and environment strongly benefits the accuracy of a simulation-based analysis for energy consumption and system-level lifetime estimation. Finally, the auxiliary models allow engineers to (iv) achieve *more comprehensive code generation*. Indeed, many programming languages can be used today for implementing WSNs, mostly depending on the targeted hardware nodes [9]; having a formalized link between the application layer and the low-level configuration of the nodes allows engineers to consider the specificities of the considered nodes when generating code from SAML models (e.g. generating differently optimized versions of C++ code when targeting an Arduino UNO or MEGA 2560 boards). The main concepts of each modelling language are described in Sect. 4.

Feasibility is surely important. A4WSN allows the feasibility checks about the mapping to be performed in quite a straightforward manner (e.g. by taking into consideration suitably annotated SAML models in an OCL constraint).

The **programming framework** (see Fig. 2) provides a set of facilities for supporting the development and integration of *code generation* and/or *analysis* engines. In Fig. 2, C_i and A_i represent code generation and analysis engines, respectively. The proposed programming framework knows at runtime which plug-ins are installed in the framework, and automatically provides the user with the available target implementation languages or the available analysis techniques.

Code generation and analysis plug-ins are structurally similar. An analysis plug-in manages the analysis of WSNs (e.g. coverage, connectivity, energy consumption analysis), instead of a code generation plug-in which is tailored to the generation of implementation code conforming to a set of specific target languages. More specifically, in A4WSN the main difference between code generation and analysis plug-ins resides in their returned output: the main output of a code generation engine can either be a set of source files or binary packages, whereas the main output of an analysis engine can be a violated property, a counterexample, a set of numerical values, and so on. The detailed description of the programming framework is presented in Sect. 5.

The A4WSN platform is generic since it is independent from the programming language, hardware, and network

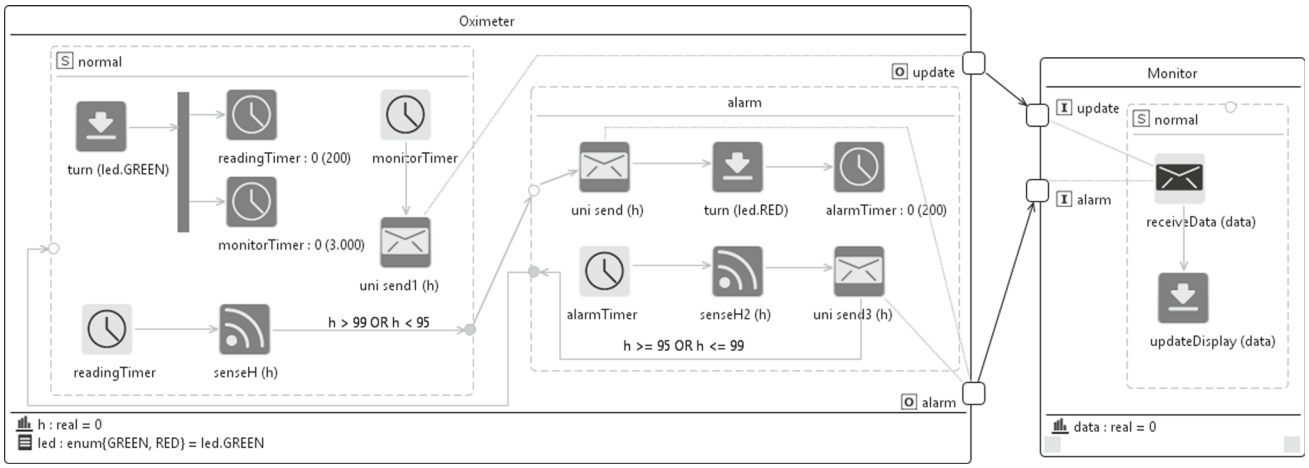


Fig. 3 Software architecture of the hospital scenario WSN

topology. Starting from a set of models (each one reflecting a certain WSN viewpoint), the code generation and analysis components can be plugged into the framework for generating executable code or analysing outcomes.

4 The modelling environment

As shown in the previous section, the modelling environment is composed of three main languages, which are SAML, NODEML, and ENVML. Each language allows the user to frame the problem of describing the architecture of a WSN from a specific viewpoint [24]. It is important to point out that the modelling environment has been realized by: (i) carefully and extensively checking the state of the art in WSN development and modelling (see Sect. 7) and (ii) discussing with WSN and embedded systems engineers with many iterations of changes. We formalize the structure and concepts of all the modelling languages of A4WSN by defining their underlying metamodel (see Appendix A). In the next sections each modelling language is discussed. For the sake of brevity, we do not describe in details each element of the A4WSN modelling languages, they are presented in a dedicated technical report available online [29].

4.1 Software architecture modelling language (SAML)

The SAML modelling language allows architects to define the software architecture of the WSN application.

The **software architecture** of a WSN is defined as a collection of software **components** and **connections**. Components interact with other components through (input or output) **message ports**; they specify the interaction points between a component and its external environment. Communication happens by message passing. The actual communication method of a message (i.e. broadcast, multicast, or unicast) is specified in the send message action described later in this section. In this context, a **connection** represents unidirectional communication channel between two message ports of two different components. The data contained in a message are accessible by specific actions and events defined in the behaviour of the involved components.

Figure 3 shows the SAML model of the WSN of the hospital scenario introduced in Sect. 2. It is important to note that this figure is actually a screenshot of the real A4WSN tool available at: http://a4wsn.di.univaq.it. From a structural point of view, the whole WSN is composed of two main components: the *Oximeter* component represents the software running on each oximeter node, while the *Monitor* component represents the software running on the monitoring station.

The internal state of a component is represented by the values of its **application data** and its current behavioural **mode**. An application data can be seen as a local variable declared in the scope of the component; application data are manipulated by actions, events, and conditions defined in the behaviour of the component. Application data can be either primitive (e.g. integer, boolean) or structured (e.g. enumeration, array, map). The *Oximeter* of the hospital scenario stores the current percentage of oxygen in the patient's blood as a real number in the *h* application data, and the current state of its status led in the *led* application data, which can be either RED or GREEN. A **mode** represents a specific status of the component at the application layer. At any given time, one and only one mode can be active in a component. The component reacts only to those events which are defined within its currently active mode. Each mode can contain a set of behavioural elements that represent actions, conditions, and events which together make up the control flow within the component from an abstract point of view. Actions and events are connected via **links** representing the control flow

among them. Optionally, a condition can be specified in a link, meaning that the behavioural flow goes through a link only if its condition evaluates to true.

An **action** represents an atomic task that can be performed by an SAML component. It is important to describe a new kind of action introduced called *scoped send message*; basically, this action tells that the set of nodes receiving the message is computed at run time, depending on the value of a boolean expression; only the nodes whose application data values satisfy the boolean expression will receive the specific message, thus enabling dynamic scope-based interactions within the WSN [35]. For example, a scoped send message may be used in order to send a message to all the nodes whose *floorName* application data is equal to "*ground*" and whose *temperature* application data is greater than 21 degrees.

An **event** is triggered in response to either an external stimulus of the component (e.g. the message reception on a input message port), or some internal mechanism of the component (e.g. a timer fired). Examples of event include entering a specific mode, receiving a message at a given port, an activation of a timer, the receiving of a call from an external service, the receiving of an interrupt from either a sensor or an actuator.

By considering the *Oximeter* component of the hospital scenario, at start-up it turns the LED into green via the *turn(led.GREEN)* actuate action and starts two cyclic timers in parallel. Every time the *monitorTimer* is triggered (every 3000 ms), the component sends the current value of the *h* application data to the *Monitor* component via the update message port. When the *readingTimer* is triggered (i.e. every 200 ms), the component senses the current oxygen percentage in the patient's blood via the *senseH* action: if the read value is not below or above the norm (i.e. if it is not between 95 and 99%), then the component switches to the *alarm* mode. In this specific mode, the component firstly sends the current read value to the Monitor component via a dedicated *alarm* message port, then it turns the LED to red, and starts a new cyclic timer with a period of 200 ms. From this point onwards this component senses the percentage of oxygen in the blood of the patient and sends it to *Monitor* every 200 ms. If the read value comes back in the acceptable range, then the component switches back to the *normal* mode. The *Monitor* component is straightforward. It has a single operating mode in which every time a message from the *Oximeter* component is received, its data are shown on a display via the *updateDisplay* actuate action. This component temporarily stores the value received by the various oximeter nodes in *data*.

4.2 Node modelling language (NODEML)

NODEML is a language that allows the abstraction of low-level details. More precisely, NODEML allows the definition of specific *nodes* that can be used to define a WSN. Once the nodes have been defined, they can be reused across different applications. Our node abstraction is based on the work that is described in [36]. More precisely, a node configuration can specify information such as operating system (e.g. TinyOS, Contiki, Mantis, LiteOS), **middleware** (such as TeenyLIME, MiLAN, RUNES [37]), **transportProtocol** (such as UDP and TCP), **macProtocol** (such as T-MAC, S-MAC, WiseMAC, SIFT [14]) and **routingProtocol** (such as SPIN, LEACH, GEAR [1]). From a structural perspective, in NODEML a WSN node contains one or more **energy sources** (e.g. batteries), a **microcontroller** (i.e. the component mainly devoted to computation and memory management), a set of **sensors**, a set of **actuators**, a set of **additional memories** representing external storage memories of the node, a set of **radio communication devices** to communicate with other nodes within the WSN, and a set of **power modes** in which the node can be at any given time.

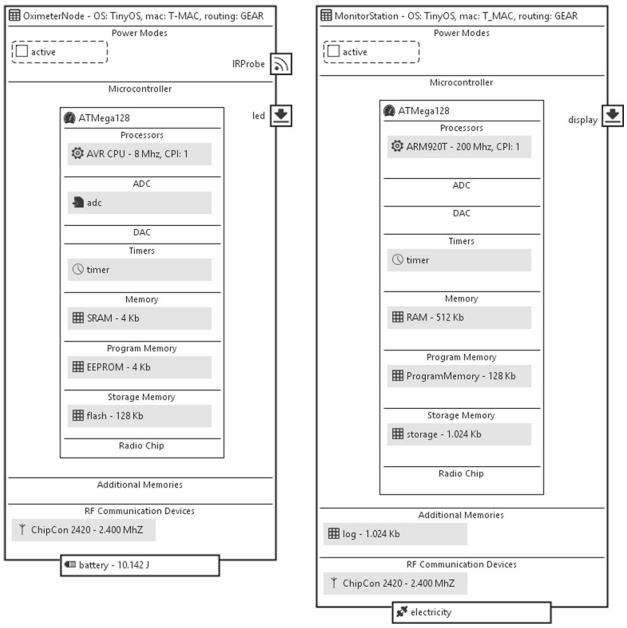


Fig. 4 Node configuration of the hospital scenario WSN

As shown in Fig. 4, the NODEML model developed for our hospital scenario is composed of two node configurations using TinyOS[4] as the operating system, GEAR as the routing protocol, and T-MAC as the MAC protocol. The specified node configurations are detailed below:

– *OximeterNode* is equipped with an *IRProbe* sensor for sensing the percentage of oxygen in the patient's blood and a LED actuator for showing the current status of the node to the personnel of the hospital. This node is powered by two AA batteries with up to 18,720 J and uses a Texas Instruments ChipCon 2420 RF transceiver.

[4] http://www.tinyos.net/.

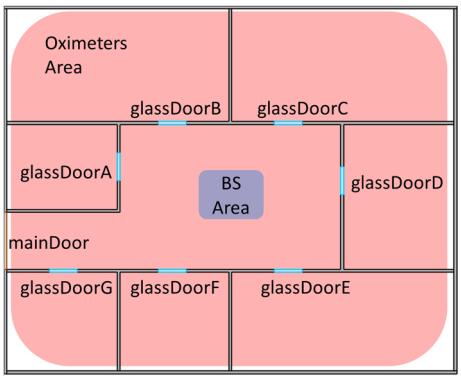


Fig. 5 Physical environment of the hospital scenario WSN

The microcontroller used is the low-power Atmel AVR ATmega128 equipped with an ADC for converting the analogue values read by the IRProbe sensor into their corresponding digital values. The oximeter node is always active (see the *active* power mode).

– *MonitorStation* has a single actuator device for graphically showing the values received by various oximeter nodes on a digital display. Similar to *OximeterNode*, it uses a Texas Instruments ChipCon 2420 RF transceiver and uses a low-power Atmel AVR ATmega128 microcontroller. The monitoring station is always active (see the *active* power mode) and is powered by a classical electrical plug connected to the main electrical system of the hospital. Finally, it is equipped with an additional storage memory for storing a log of all the values received by the oximeter nodes over time.

4.3 Environment modelling language (ENVML)

The ENVML modelling language allows the designers to specify the physical environment in which the WSN nodes are deployed.

The **Environment** represents the overall area in the 2D space in which the WSN nodes are deployed. In ENVML an image can be associated with the specified environment, allowing environment designers to provide a more detailed view of the environment by means of external CAD software; in this case the proposed ENVML models can be seen a projection of these models which focusses on obstacles and inner areas only. Any kind of relevant **obstacle** can be placed in the environment. Each obstacle is characterized by the name of the *material* it is made of (e.g. concrete wall, wooden door, glass), and its *attenuation* coefficient. The shape of the obstacle is given by its *shell*: a sequence of **coordinates** representing the perimeter of the obstacle in the 2D space.

Figure 5 shows the ENVML model representing the physical environment of our hospital scenario. It is a rectangle with 16 and 13 metres of width and height, respectively, and it contains three kinds of obstacles that are concrete walls dividing the whole environment into rooms and corridors, a main wooden door on the left, and a glass door for each patients room. Each obstacle is represented by a unique name, its attenuation coefficient, and the coordinates of all the points of its perimeter.

In ENVML an **area** identifies a portion of physical environment in which nodes of the same type can be distributed according to a distribution policy (defined in the DEPML modelling language, see Sect. 4.4). Similar to obstacles, the perimeter of the area is defined by means of its shell.

The physical environment of the hospital scenario contains two main deployment areas:

– *BSArea* is a square area at the centre of the environment and will contain the central monitoring station.
– *OximetersArea* its perimeter is the same as the whole physical environment and will contain all the oximeter nodes, one for each patients' room.

It is important to note that the above mentioned solution is one of the possible deployment configurations; another solution could also consist of the creation of a single area for each oximeter, where each oximeter could be placed in the centre of the area. The aforementioned solutions share the same network topology

4.4 The two auxiliary modelling languages: MAPML and DEPML

The two auxiliary modelling languages are MAPML (the Mapping Modelling Language that assigns software components to the corresponding hardware node configuration they will be executed on) and DEPML (the Deployment Modelling Language used to virtually deploy WSN nodes into the physical environment).

A MAPML model semantically represents the classical notion of deployment of software components onto hardware resources [59].

A MAPML model is made of a set of **node mappings**, each of them linking a node definition from the NODEML model and a component from the SAML model. The semantics of a node mapping is that the linked component in the SAML model will be physically deployed on the linked node in the NODEML model.

For what concerns our hospital scenario, the MAPML model (which links the SAML and NODEML model of Figs. 3, 4) has the following form:

– *NodeMapping_oximeter* links the *Oximeter* component to the *OximeterNode* node;

  - *ModeMapping_active* links both the *normal* and *alarm* modes of the *Oximeter* component to the *active* power mode of the *OximeterNode* node. It is important to note that operating modes defined in SAML are pure logical modes, whereas NODEML power modes depend on the hardware configuration of the node itself.
  - *SensorMapping_irProbe* links both the *SenseH(h)* and *SenseH2(h)* SAML sense actions to the hardware *IRProbe* sensor in the NODEML model.
  - *ActuatorMapping_led*, which is similar to *SensorMapping_irProbe*, links both the *turn(led.GREEN)* and *turn(led.RED)* SAML actions to the hardware *led* actuator in the NODEML model.
  - *CommunicationDeviceMapping_2420* links the *update* and *alarm* SAML message ports to the *ChipCon2420* RF transceiver defined in the NODEML model.
- *NodeMapping_monitor* links the *Monitor* component to the *MonitorStation* node;
  - *ModeMapping_active* links the *normal* mode of the *Monitor* component to the *active* power mode of the *MonitorStation* node.
  - *ActuatorMapping_display* links the SAML actuate action called *updateDisplay(data)* to the hardware *display* actuator in the NODEML model.
  - *CommunicationDeviceMapping_2420* links the *update* and *alarm* SAML message ports of the *Monitor* component to the *ChipCon2420* RF transceiver defined in the NODEML model.

DEPML is our language for virtually deploying WSN nodes into the physical environment. DEPML allows designers to consider each NODEML node configuration and to *instantiate* it in a specific area within the physical environment defined in a ENVML model. A DEPML model contains a single type of link called **deployment link**, which links together a node configuration in NODEML and an area in ENVML. The semantics of the deployment link is that the linked node configuration is instantiated and virtually deployed in the linked area multiple times. This allow designers to focus on generic components and node types in SAML and NODEML, while in DEPML they can reason on the final deployment of the WSN. The number of nodes that are instantiated in the area is specified in the *numberOfNodes* attribute. Within a certain area each node configuration can be *distributed* in three different ways: random, grid, and custom.

For what concerns our hospital scenario, the DEPML model contains the following elements:

- *DeploymentLink_oximeter* links the *OximeterNode* node in the NODEML model to the *OximetersArea* ENVML area. Since we want to specify that exactly one oximeter node must be deployed in each patient's room, we define a custom node distribution. Thus, we manually define the exact position of the deployed nodes by means of ten *deployed node* elements, each of them containing the coordinates of its position in the environment.
- *DeploymentLink_monitorStation* links the *MonitorStation* NODEML node to the *BSArea* ENVML area. In this case we specify that the number of deployed nodes is only one, with a random distribution within the area (we can do this because the area is a square with a side of 0.5 metres, which is exactly the size of the monitoring station node).

4.5 Model correctness and feasibility

All the proposed languages have been designed to provide a good trade-off between genericity, expressivity, and accuracy in capturing the various facets of the WSN domain. In this respect, it is fundamental to allow designers to check whether their models are correct with respect to the semantics of the proposed languages. A4WSN provides two different mechanisms for checking the correctness of the developed models, namely: (i) model conformance and (ii) a set of OCL constraints.

A4WSN allows designers to check whether a model adheres to the structural semantics of its corresponding language (e.g. SAML). A4WSN supports this feature by leveraging the well-known notion of **conformance** in model-driven engineering; in other words, in A4WSN a model *m* adheres to the structural semantics of its corresponding language (e.g. SAML) if and only if *m* actually conforms to its metamodel (e.g. the SAML metamodel introduced in Sect. 4.1).

In order to ensure a more precise semantics of the languages described in Sect. 4, we complemented the metamodel of each language with a set of OCL[5] constraints. OCL is based on first-order predicate logic, and it is a language to describe expressions and constraints predicating on models in an object-oriented fashion. In A4WSN we use OCL constraints for enforcing the correctness and feasibility of the designed models. For example, the OCL constraint shown in Listing 1 is defined in the context of an SAML connection between components (line 1) and ensures that in SAML models each instance of *Connection* links together ports belonging to different components (line 3); when the constraint is violated, the architect is informed about it via a dedicated error message (line 4).

[5] Object Constraint Language (OCL) specification: http://www.omg.org/spec/OCL/2.3.1.

```
context Connection{
  constraint sameComponentConnection {
    check: (self.source.eContainer() = self.target.
     eContainer())
    message: 'Source and target ports of the ' + self.source
     .name + '–' + self.target.name + ' connection cannot
     belong to the same component'
  }
}
```

Listing 1 Example of OCL constraint checking if an SAML connection links ports belonging to the same component

The A4WSN platform contains fourteen OCL constraints on the various modelling languages of the platform. For example, another constraint in DEPML ensures that the coordinates of each manually positioned node must be within the boundaries of the area it is deployed in, and so on. For the sake of readability, the description of such constraints is not discussed extensively in this article.

It is important to stress that our set of OCL constraints are defined in the context of all the A4WSN modelling languages. When considering the auxiliary A4WSN languages (i.e. MAPML and DEPML), our OCL constraints help architects and designers in actually evaluating the functional feasibility of either the software and hardware mapping (when considering MAPML models) or the virtual deployment of the nodes in the environment (when considering ENVML models). These OCL constraints are especially important since their checks cross-cut multiple models conforming to different modelling languages; this is a non-trivial situation, where a manual analysis to identify and fix their violations could be very challenging for architects and designers. Listing 2 shows two OCL constraints performing two of those cross-model checks.

```
1  context Sense{
2    constraint senseActionNotMapped {
3      check {
4        // mapModel is a reference to a MAPML model
5        for (m in mapModel.mappings.select(ele.eClass().name='
         SensorMapping') {
6          /* resolveLinkEnd() is a custom operation for
         obtaining a
7          model element from a link end referring to it */
8          if(m.senseAction.resolveLinkEnd() = self) {
9              return true;
10         }
11       }
12       return false;
13     }
14     message: 'The ' + self.name + ' SAML sense action is not
         mapped to any NODEML sensor'
15   }
16 }
17
18 context SensorMapping{
19   constraint sameComponentDifferentNode {
20     check{
21       var component = self.getComponent();
22       var node = self.getNode()
23       return not SensorMapping.allInstances.exists(p | (p.
         getComponent() = self.getComponent()) and (p.getNode()
         <> self.getNode()))
24     }
25     message: 'Two SAML sense actions belonging to the ' +
         self.getComponent() + ' component are mapped to two
         different NODEML nodes'
26   }
27 }
28
29 // returns the SAML component containing the mapped sense
       action
30 operation SensorMapping getComponent() : Component {
31   return self.senseAction.resolveLinkEnd().eContainer().
       eContainer();
32 }
33
34 // returns the NODEML node containing the mapped sensor
35 operation SensorMapping getNode() : Node {
36   return self.sensor.resolveLinkEnd().eContainer();
37 }
```

Listing 2 Examples of OCL contraints checking inter-model conditions between SAML and NODEML models

The *senseActionNotMapped* constraint is defined in the context of SAML sense actions (line 2) and checks among all MAPML sensor mappings (line 5) if there is one involving the current sense action (lines 8–12); an error message is shown to the architect if the current SAML sense action is not mapped to any NODEML sensor (line 14). Another non-trivial constraint is shown in lines 18–27 of Listing 2. This constraint is defined in the context of MAPML sensor mappings (line 18) and checks if there are SAML sense actions belonging to the same components, but at the same time they are mapped to more than one NODEML node (lines 21–24); this situation is erroneous in A4WSN because we assume that every SAML component is an atomic unit of deployment, i.e. an SAML component cannot be deployed to more than one NODEML node at the same time. As shown in lines 29–37, the OCL engine we use in the current implementation of A4WSN allows us to abstract complex operations on the models as auxiliary operations (see Appendix A for more detail). For example, the *getComponent()* operation is defined in the context of a MAPML sensor mapping; it identifies the linked SAML sense action (via another *resolveLinkEnd()* operation) and returns the SAML component containing the sense action by navigating upwards twice in the containment hierarchy of the SAML model. The *getNode()* operation performs a similar logic by identifying the NODEML node containing a NODEML sensor mapped by a specific MAPML sensor mapping.

If the need for more strict semantics of the proposed languages arises (for instance, in order to define WSN applications with specific styles or special configurations), additional OCL constraints can be added to every element of all languages by extending the A4WSN platform with a suitable analysis plug-in. Extensibility in terms of additional

checks on the models is one of the main drivers that lead us to adopt a plug-in-based architecture in A4WSN. Indeed, in principle it is very difficult (if even possible) to anticipate what are all (possibly project- and application-specific) constraints that all users of A4WSN may need to enforce; in this context, our solution is to (i) provide default OCL constraints in A4WSN for covering the general cases (e.g. checking if all links within an SAML model are connecting compatible ports) and (ii) to allow plug-in developers to define and enforce their own additional constraints when the base ones are not enough for satisfying specific needs. Please refer to Sect. 5 for more details on this feature of the A4WSN platform.

4.6 Discussion

This section has presented a 3+2 modelling framework that by using three main modelling languages and two auxiliary ones, supports the specification and analysis of WSNs. While a comparison with related work is provided in Sect. 7, the aim of this section is to briefly discuss why a new set of domain-specific modelling languages (DSMLs) is presented, instead of extending existing ones.

According to [34] *"domain-specific languages (DSLs) are languages tailored to a specific application domain. They offer substantial gains in expressiveness and ease of use compared with general-purpose programming languages in their domain of application"*. While DSL and DSML are not synonyms of the same concept [8,44], a number of advantages pointed in [34] still apply to DSML:

- domain-specific constructs defined for the domain of interest (WSNs, in our specific case) are far more fine-grained and specific than user-definable operators of existing modelling languages. More specifically, we could have expressed the SAML by profiling UML State Machines. However, this would have implied forcing SAML to strictly follow the semantics of UML state machines and to introduce a number of other concepts by specializing UML constructs. Furthermore, the definition of the ENVML model would have required an under-specification of the 2D space.
- The use of DSMLs offers possibilities for analysis, verification, and transformation that are far beyond what is supported by general-purpose languages. In our specific case, through the definition of (and with an option to extend) the A4WSN modelling languages, we can run a multitude of domain-specific predictive analysis techniques (such as the PlaceLife plug-in we developed to estimate the WSN lifetime—see Sect. 6).
- overall, DSMLs offer gains in reuse and maintenance. Accordingly, the A4WSN SAML, NODEML, and MAPML models are reusable in different applications and applications domains.

When designing the modelling languages of A4WSN, we aimed to represent the domain of WSNs in order to cover its most representative concepts and entities. We identified the set of concepts of the A4WSN modelling languages by working closely with industry partners and continuously performing informal interviews with engineers, developers, researchers, and other involved stakeholders within WSN-based projects. So, it is important to stress the fact that with the proposed modelling languages we do not aim to address all the possible concerns in all possible situations about WSNs (e.g. dependability, sensing coverage, networking and communication, performance), especially because of the intrinsic multidisciplinary nature of the WSN problem space. Also, as already discussed in the literature about architecture description languages [15,32,33], having a comprehensive modelling language containing all the possible concepts related to a given domain may be unfeasible, or at least may lead to large and complex languages, which may be cognitively difficult to manage and maintain. The A4WSN platform targets the following concerns in the domain of WSN engineering: (i) separation of concerns, addressed via the multi-view modelling paradigm adopted in the modelling framework, (ii) reuse, addressed via the (independent) sharing of architecture, nodes, and environment models across projects and organizations, and (iii) model-based analysis, addressed via the A4WSN plug-ins system and its programming framework.

It is also important to note that the A4WSN modelling languages can be easily extended by means of generic and language-independent composition engines proposed in the literature. For example, in [15] we proposed a language composition engine that extends architectural languages with domain-specific concerns, with new architectural views, with analysis constructs or with methodology and process concepts, depending on the system's stakeholder concerns. In this case, the needed additional concepts may live in dedicated plug-ins of the A4WSN platform and can be used by the WSN engineers when needed.

5 The A4WSN programming framework

As introduced in the beginning of Sect. 3, the A4WSN platform is composed of two main parts that are (i) a modelling environment to allow architects to model WSN applications and (ii) a programming framework devoted to code generation and analysis of WSN application models. The motivation for performing code generation and analysis of WSN application models are well understood both in academia and in practice [26,47]. Basically, code generation helps in reducing

the cost of developing a WSN application since the developers can automatically obtain an executable application from the model by applying some specific transformations. Also, performing analysis is fundamental while developing a WSN application due to the intrinsic complexity of the WSN domain. For example, if we consider typical aspects in WSN development such as node connectivity, real-time communication, energy consumption, performance, security, it is extremely difficult and demands a lot of effort to ensure that a developed WSN is correct with respect to those aspects. Moreover, analysis engines can also be used to reason about the WSN configuration in order to find reasonable trade-offs in terms of network topology, employed protocols, etc., for a specific task.

In this section we present the *generic and extensible* programming framework of A4WSN. It is tailored to support the development of code generation and analysis engines against WSN application models conforming to the modelling languages described in Sect. 4.

Our programming framework offers a generic workbench and a set of extension points to support the development and integration of third-party code generation and analysis engines. More specifically, through its components, it enables the storage of WSN models, supports the merging of linked models, validates A4WSN models, provides error/warning/information messages to the user, defines a UI manager to make plug-ins interact, and provides facilities for managing code generation and analysis engines.

Third-party engines are realized as plug-ins extending the generic A4WSN workbench. It detects at run time which plug-ins are available and automatically provides to the user the available target implementation languages and analysis techniques.

Figure 6 shows an overview of the A4WSN programming framework. All the boxes within the programming framework represent the various components of the generic programming workbench, whereas the $C_1..C_n$ and $A_1..A_n$ boxes represent third-party code generation and analysis plug-ins, respectively. Third-party plug-ins extend the *Code Generation Manager* and *Analysis Manager* components which provide the needed extension points, and they communicate with all the other components of the programming framework. (For the sake of clarity, we do not show those connectors in the figure.) In the following we introduce the facilities and duties of the various components.

Models It is a repository storing all the WSN models developed by architects and designers. Stored models can conform to any A4WSN modelling language, which are SAML, NODEML, ENVML, MAPML, and DEPML. The model repository can be implemented in different ways. For instance, it may directly rely on the file system of the machine running the A4WSN platform (this is the solution implemented in the current version of the A4WSN tool), it may point to resources stored in the cloud or it may refer to some in-memory model representation. If on the one hand this feature of the model repository is very flexible in terms of resources consumption and localization, on the other hand it opens for possible problems of interoperability between all the other components of the A4WSN programming framework. This is exactly why the Model Adapter component exists.

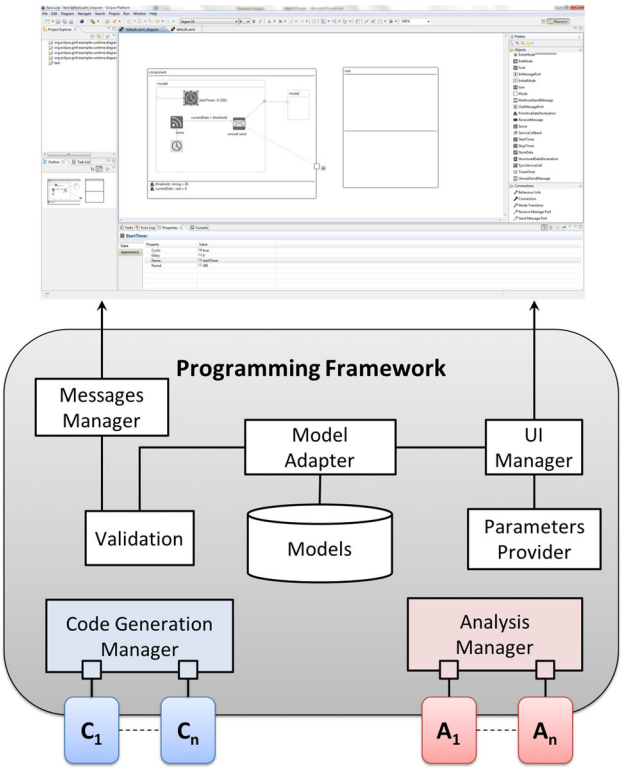


Fig. 6 The A4WSN programming framework

Model Adapter It is a component which abstracts the nature of the model repository to the other components of the A4WSN programming framework. The model adapter is composed of a set of connectors (each of them tailored to a specific model storage type) that expose a common interface to all the other components to access various elements of the models in a homogeneous way. The Model Adapter component has a built-in model transformation, called *Merger*, that takes as input an instance of each A4WSN modelling language and produces a single model conforming to a unique metamodel. The reason behind the existence of the Merger transformation is that currently many approaches and tools for code generation and analysis assume to have a single model as an input, rather than a set of models conforming to different languages. In order to alleviate this issue with current approaches and tools (which could have hampered the usefulness of the whole A4WSN platform), we decided to implement the Merger as an internal transformation to merge

separate models into a single one. Merger can be executed at any time by plug-in developers by calling a dedicated Java method.

Validation The Validation component executes all the operations to validate A4WSN models, namely:

- it checks whether one of the A4WSN models conforms to its corresponding metamodel;
- it executes all the OCL constraints defined in each metamodel within the A4WSN platform and checks whether they are satisfied or not;
- if defined, it executes the additional OCL constraints that are defined in some code generation or analysis plug-ins and checks whether they are satisfied or not.

The Validation component communicates with Model Adapter in order to access various elements of the models to be validated. Also, it communicates with the Messages Manager and the UI Manager components to (i) show informative messages to the user and to (ii) highlight the model elements violating the constraints, respectively.

Message Manager The Message Manager component serves to show informative messages to the user. A4WSN supports three kind of informative messages which are *error, warning, and information*. Plug-in developers can decide the type of each message to be shown, depending on its severity.

UI Manager The UI Manager component is responsible for the main facilities interacting with the user interface of the A4WSN platform.[6] The UI Manager component provides all the graphical facilities to interact with the plug-ins and elements of the A4WSN platform, such as a dedicated view showing a list of all the available code generation and analysis plug-ins, contextual menus implemented by plug-ins, validation triggers (i.e. contextual menus and buttons) for validating the currently edited model, and a dedicated view in which users can provide additional parameter to be passed to the code generation or analysis engine being triggered.

Parameter Provider It manages the additional parameters that a code generation or analysis plug-in may require for carrying on its activities. As previously mentioned, additional parameters are defined by using a specific extension point of the A4WSN programming framework. Additional parameters can be plain strings, numbers, Booleans, files, or resources in the cloud which can be accessed by a standard HTTP GET request to a given URL. Once the user has provided the values of the additional parameter of a code generation or analysis engine, the Parameter Provider component makes them available to the plug-in realizing the engine so that it can access them before actually executing the activity which is being triggered by the user.

Code Generation Manager The Code Generation Manager provides a set of facilities for managing code generation engines and the extension point that is used by code generation plug-in developers. For instance, it checks which plug-ins are currently extending its extension point and makes their facilities available to the end user. It includes all the registered code generation plug-ins into their specific view in the UI Manager. It loads plug-ins into the contextual menus of the A4WSN modelling environment. It automatically triggers the validation operations defined by the plug-ins before executing the actual code generation operation. Also, the Code Generation Manager component exposes a common *Java API* to plug-in developers, so that they can easily interact with all the other components of the A4WSN programming framework. For example, it allows developers to access elements of the models in the Model Repository, or to push messages to the end user via the Message Manager.

Analysis Manager The internal logic of the Analysis Manager component is analogous to that of Code Generation Manager. The only difference is that it is designed for analysis plug-ins, rather than for code generation plug-ins.

Extension Points The concept of extension point is nicely described in the Eclipse Wiki,[7] it says that *the extension point declares a contract, typically a combination of XML markup and Java interfaces that extensions must conform to. Plug-ins that want to connect to that extension point must implement that contract in their extension. The key attribute is that the plug-in being extended knows nothing about the plug-in that is connecting to it beyond the scope of that extension oint contract. This allows plug-ins built by different individuals or companies to interact seamlessly, even without their knowing much about one another.* The last part of the Eclipse definition of extension point says exactly what we are suggesting to the WSN research community, i.e. not to rebuild the wheel by focussing on modelling languages, graphical editors, but rather to focus on code generation and analysis of WSN applications by developing A4WSN plug-ins.

The extension points defined in the A4WSN programming framework are used to group code generation and analysis engines into two different groups, so that the end user knows where those engines can be found. Also, they are used to provide a common, standard behaviour to various engines that may be defined upon the A4WSN modelling environment. Both the Code Generation Manager and Analysis Man-

[6] Also the Messages Manager interacts with the UI of the A4WSN platform; however, its impact to the UI is much more limited than that of UI Manager.

[7] http://wiki.eclipse.org/FAQ_What_are_extensions_and_extension_points\%3F.

ager provide a standard management of the workflow that must be followed when executing those engines. For example, they automatically call the set of preliminary actions defined by using a specific extension point (the same holds for post-actions). Managers also automatically manage the success and error messages to be shown after the execution of either a code generation or analysis operation. Additional operations performed by managers include updating the UI of the modelling framework depending on the available plug-ins extending A4WSN, keeping intermediate resources produced by plug-ins, provide the means for specifying additional parameters needed by specific plug-ins. The complete set of extension points that can be used by third-party plug-in developers is described in details in [28].

6 PlaceLife: an A4WSN plug-in

In order to validate the expressivity of the A4WSN modelling languages and to exercise the provided extension points, we developed an analysis plug-in called PlaceLife which takes advantage of the three modelling views (namely SAML, NODEML and ENVML) to estimate the WSN's lifetime. More precisely, all modelling views are analysed, combined and translated into low-level simulation scripts that can be executed to evaluate the WSN's lifetime. This translation has been useful to verify that our models have an appropriate level of detail for simulation purposes. In order to simulate the system realistically the following is desirable:

- **Abstraction:** The models abstract all the details needed to generate scripts that can run in various well-accepted simulators such as OPNET [10] and OMNeT++ [61];
- **Fine-grain simulation:** The details should be able to sufficiently combine different information such as physical environment, hardware and various layers of OSI.

The abstraction is verified by considering the information required by well-established simulation tools for each OSI layer. In most of the cases A4WSN models successfully abstract the information but missing ones can also be easily added using a specific plug-in. More precisely:

- Application layer: Includes the structure and behaviour related information of the WSN such as the type of components, number of instances and their interaction. The modelling languages of A4WSN provide ways to define the aforementioned application layer information. The SAML view contains structure and behaviour of the WSN application. This information is complemented by the NODEML view that specifies, among the other information, the type of operating system and the middleware used. Finally, ENVML and DEPML models provide information about the number of nodes within the WSN and their deployment position in the environment;
- Networking and data link layers: Information at networking layer should specify the routing protocol. A4WSN provides such information through its NODEML model where either multi-hop routing protocols (e.g. AODV) or some clustering approaches (e.g. LEACH) are specified. Static routing can also be defined by explicitly specifying the connection among nodes. The NODEML also includes a wide range of possible MAC protocols such as CSMA, T-MAC and S-MAC [60,63].
- Physical layer and hardware: Information should support the definition of an energy consumption model for realistic estimate of the WSN lifetime. An advanced energy consumption model should consider the path-loss [40], modulation scheme, hardware, coding scheme, and so on. The modulation scheme, the hardware, and the coding scheme are specified in the NODEML model, and the path-loss model is defined according to the environment and its obstacles as explained in Sect. 6.1. NODEML, ENVML, and path-loss definitions are used to generate low-level settings and scripts that can provide an accurate estimate of the energy required to transmit a bit over the physical channel.
- Analytical model: The analytical framework for calculation of the lifetime of nodes is presented in Sect. 6.2.

We keep the core modelling languages SAML, NODEML, ENVML as clean and minimal as possible, without polluting them with analysis or code generation-specific constructs. However, the users can add their own analysis-specific models and concepts via dedicated plug-ins in order to streamline the analyses that they need to perform. For instance, in the healthcare case study A4WSN models the application behaviour (e.g. sampling rate and event notification policy) while PlaceLife outputs Castalia simulations [46]. Castalia is a widely used simulator that is built on the top of OMNeT++. Castalia adds to OMNeT++ realistic wireless and radio model channels for WSNs plus well-known WSN mac protocol simulations.

Fine-grain simulations are easily obtained thanks to the reuse and the weaving of multiple models into a single one. Models such as NODEML or ENVML that contain low-level information can be created once and reused multiple times with different application models. Technical details such as the effects of the path-loss and the hardware which require theories of telecommunication are specified in pre-built PlaceLife models. Furthermore, these details are complemented with the application model (i.e. SAML) and the physical environment (i.e. ENVML). These models are transformed into various complex simulation scripts. In Sect. 6.3 we describe the PlaceLife implementation and the simulation

tool used as a target language for simulation script generation.

In Sect. 6.4 we compare the simulation results obtained with a basic Castalia simulation with a PlaceLife simulation based on pre-built hardware and path-loss models, applied to the hospital application. Numerical results are presented to show the effects of a realistic simulation scenario, where environmental factors are taken into account. The former has the default ideal free-space model for the path loss, while the latter considers pre-built PlaceLife models that consider the real environment that is made of physical objects. We see that not considering the real environment may cause significant overestimations of the lifetime which is particularly undesirable.

6.1 The path loss

The path loss is a reduction in transmitted signal strength as a function of distance, which determines how far apart two sensor devices can be and have reliable communication between the devices [42]. In sensor networks, path loss can play a crucial role since neglecting the path loss may cause overestimation of WSN lifetime [40].

Outdoor propagation models are commonly used for indoor propagation estimations as well. However, the indoor environment differs widely due to the increased number of obstacles, layout of rooms, presence of multiple walls and floors, windows, and open spaces. These factors can have significant impacts on path loss in an indoor environment. Due to the irregularity in the position of obstacles and layout of the rooms, the indoor propagation modelling becomes quite challenging. The propagation and path-loss models are usually based on empirical studies of the system considered.

Most of the simulation studies make use of the models that are developed based on empirical measurements over a given distance in a given operational frequency range and a particular environment [48]. Some of the most common empirical models include Okumura model, Hata model, Erceg model, ITU indoor path-loss model, log-distance path-loss model [48,55]. When considering an indoor propagation environment for path loss, the material used for walls and floors, the layout of rooms, windows and open areas, location, etc., should be taken into account, as all of these factors can have a substantial impact on the path loss. The complexity of indoor signal propagation makes it difficult to obtain a single model that illustrates path loss across a wide range of environments. The following is a commonly used simplified model for path loss as a function of distance [18].

$$P_r\,(dBm) = P_t\,(dBm) + K(dB) - 10\gamma\,\log_{10}\left(\frac{d}{d_0}\right) \quad (1)$$

Table 1 Partition-dependent losses for 2.4 GHz

Attenuating material	Signal attenuation in dB
Wood	2
Metal frame, glass wall into building	6
Office wall	6
Metal door in office wall	6
Cinder wall	4
Metal door in brick wall	12.4
Brick wall next to metal door	3

where P_r is the received signal strength and P_t is the transmitted signal strength; K is the path-loss factor (depends on antenna characteristics and the average channel attenuation), γ is the path-loss exponent, and d_0 is the reference distance for the antenna far field and is typically assumed to be 1–10m for indoor scenarios and 10–100m for outdoor scenarios. The path-loss factor K can be calculated as:

$$K\,(dB) = 20\,\log_{10}\frac{\lambda}{4\pi d_0} \quad (2)$$

The value of λ depends on the propagation environment. This path-loss model, together with the ENVML physical environmental model, is used to define the path loss between any two nodes. Please note that existing simulation packages and modelling architectures do not consider the effects of path loss to best of our knowledge. We fix the value of γ at 2 for free space and introduce the losses for each partition (obstacle) that is encountered by a straight line connecting the receiver and the transmitter. Please refer to Table 1 for the decibel loss values measured for different type of partitions, at 2.4 GHz [42]. In order to add the effects of obstacles between the transmitter and the receiver, we add the fixed path losses per existing obstacles to the free-space path loss.

The physical layer has a fundamental role when energy consumption is considered, and choosing a model for energy consumption can be complicated. Different studies and simulation tools employ various models for this purpose [6,20,61].

6.2 Analytical model

In this section an analytical framework for calculation of the lifetime of nodes is presented to verify the simulation results. The analytical framework considers the characteristics of the transmission process which affects the lifetime of a node. Without loss of generality, the lifetime L of a node n can be expressed as:

Table 2 Selected values

Link safety margin	$M = 10$
Receiver noise figure	$N_f = 5$
Ambient noise power spectral density	$N_0 = -204$ dBJ
Power amplifier efficiency	$\gamma = 0.35$
Combined gain of the transmit and receive antennas	$G = 1$
Required signal-to-noise ratio per bit τ = transmission technique $\Im$ = fading characteristics and B = target bit error rate	$f_{\tau,\Im}(B) = 15$ dB
circuitry	$E_{ct} = E_{cr} = 1$ μJ

$$L_n = \frac{E_{tot}}{E_{pr} \cdot R_r + E_{pt} \cdot R_t} \tag{3}$$

where L_n is the lifetime of the nodes, E_{tot} is the total energy in Joules available for the node considered (e.g. initial energy for two AA battery is 18720 joules), E_{pr} and E_{pt} are the energy spent to receive and transmit a single packet, and R_r and R_t are the average number of packets received and sent per second, respectively.

For the analytical framework introduced in this paper, the energy consumption model described in [6] is combined with the effective number of transmissions including the retransmissions caused by obstacles. The energy consumption of a node n is affected by E_{ct}, E_{cr}, and E_t which represent the transmitter circuit energy, receiver circuit energy, and the transmission energy, respectively. The calculation of the transmission energy E_t is based on the following expression:

$$E_t = \frac{L \cdot M \cdot N_f \cdot N_0}{\gamma \cdot G} \cdot f_{\tau,\Im}(B) \tag{4}$$

where L is the path loss (see [43] for details), M is the link safety margin, N_f is the receiver noise figure, N_0 is the ambient noise power spectral density, γ is the power amplifier efficiency, G is the combined gain of the transmit and receive antennas, $f_{\tau,\Im}(B)$ is the required signal-to-noise ratio per bit corresponding to transmission technique τ, fading characteristics $\Im$, and target bit error rate B (Table 2).

The energy spent for receiving a packet of size s is:

$$E_{pr} = E_{cr} \cdot s \tag{5}$$

6.3 PlaceLife implementation

PlaceLife generates Castalia and OMNeT++ simulation scripts. Castalia is a WSN simulator based on the OMNeT++, where OMNeT++ platform is an extensible, modular, component-based C++ simulation library and framework, primarily for building network simulators. Castalia provides OMNeT++ simulation of WSN radio channel plus MAC and network protocols.

The application behaviour is needed to derive application-level simulation parameters. The environment and the path-loss models allow the calculation of the path loss. In fact, while Castalia assumes that the user provides path-loss related parameters in a complex path-loss matrix, PlaceLife presents an abstract view of the environment where the path loss is derived from the characteristics of the environment specified in the ENVML model. OMNeT++ is used for additional simulation components such as the sensing devices and the middleware library.

6.4 PlaceLife applied to the wireless health monitoring system: numerical results and discussions

In order to show the effectiveness of the architectural approach and the PlaceLife plug-in, in this section (i) we consider the health monitoring case study that is presented in Fig. 1, and (ii) we present numerical results of its simulation. The numerical results also show the effects of realistic simulation scenarios, where environmental factors are taken into account.

The system considered monitors vital signals in a hospital environment. Wireless networked sensors enable dense spatio-temporal sampling in spaces, ranging from personal to physical environment [2]. We consider the feasibility of continuously monitoring the heart rate and saturation of patients' haemoglobin.

The lifetime of the nodes considered is presented as a function of the number of packets per minute in Fig. 7 for obstacles with exponent two, and six. Please note that exponent two can be used for wood and six is mainly for metal obstacles. The obstacles considered are mainly the walls used for indoor segmentation in Fig. 1. Numerical results are also presented for a scenario where the path loss caused by the obstacles is avoided (exponent is zero). The results clearly show that avoiding path loss would cause overestimation of the WSN lifetime. More precisely, for node six the lifetime is 281.90 h when the obstacle is metal, 307 h when the obstacle is wood and 350.06 when the path loss is avoided. In other words the resources can be overestimated up to 19.5% if the path-loss factor is avoided. PlaceLife's ENVML model allows the users to incorporate various path-loss models (in this study we used the one in [42]) for the estimation of the WSN's lifetime.

It is clear that the energy consumption of all the nodes is significantly higher when the partition is metal. When the partition is wood, the lifetime of the node is approximately 5–10% more than that of the nodes with metal as the parti-

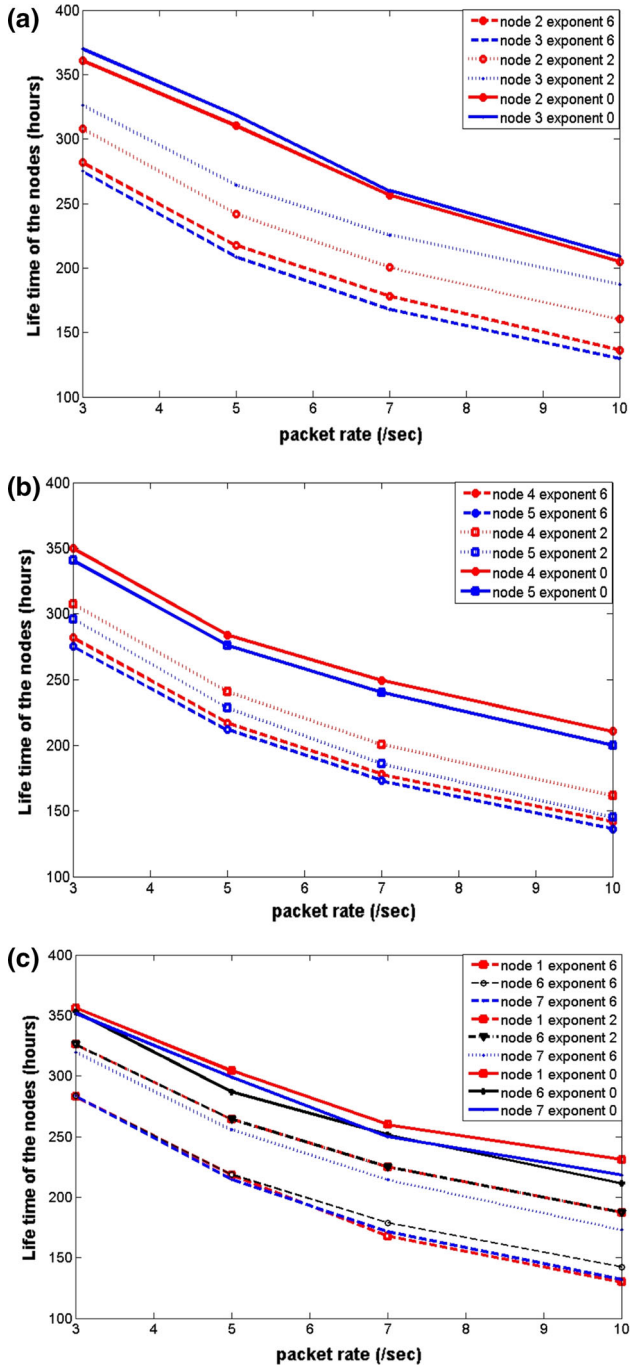


Fig. 7 Lifetime of the nodes

tion. The lifetime of a node can decrease from 320 h, down to 270 h, when the exponent of the obstacle increases. In other words, as the exponent of the obstacles increases, the effects of path loss also increase since the amount of retransmissions leads to higher energy consumptions. Numerical results presented in Fig. 7 show the effects of potential overestimation of resources in case the obstacles of a specific scenario are avoided. Our PlaceLife plug-in allows engineers to consider the nature of the obstacles of the environment in details, thus providing a more realistic performance measurement.

The simulation tool employed allows us to consider other measurements in addition to the lifetime of the WSN nodes as well. Various performance measures such as response time (latency) and the number of dropped packets can be analysed in detail.

7 Related work

In order to simplify the design and configuration of the WSN at large, and abstract from technical low-level details, a number of MDE approaches or of modelling notations for WSN engineering have been proposed. Those approaches are used to specify a WSN at different levels of abstraction (hardware, application, communication protocols, etc.) with the recurrent goals of code generation, communication overhead analysis, and energy consumption.

The rest of this section is structured so as to cover three related research areas: i) frameworks for the engineering of WSNs (or related domains), ii) domain-specific modelling languages for WSN, iii) other modelling and analysis approaches for engineering WSNs, and iv) surveys related to the modelling and analysis of WSNs.

Frameworks for Engineering WSNs (or related domains): Engineering frameworks strongly related to A4WSN have been presented in [5,45,56].

In Reference [5] the authors propose DiaSuite, a tool suite proposed for the design, analysis, and deployment of Sense/Compute/Control applications. The DiaSuite domain-specific design language supports the modelling of a taxonomy layer and an application design layer. Those models are successively used to generate a dedicated Java programming framework (to guide and support programmers to implement the various parts of the software system), for simulation purposes, and for deploying the application on a specific execution platform. When compared with A4WSN, DiaSuite has similar goals (modelling for analysis and code generation), but covers a different domain. As a result, the modelling languages are extremely different, and are manipulated (for analysis and code generation) in different ways.

In [56] a set of modelling languages is the starting point for code generation and performance (with energy consumption) analysis. Those languages are based on concepts such as sampling task, aggregation task, network communication tasks and they are the starting point of a model-driven process to enable a low-cost prototyping and optimization of WSN applications. In [39], a framework for modelling, simulation, and code generation of WSNs is presented. The framework is based on Simulink, Stateflow and Embedded Coder and allows engineers to simulate and automatically generate code with energy as one of the main issues.

In Reference [45] a multi-stage model-driven approach for IoT application development has been proposed. Such an approach takes into explicit consideration the existence of five different types of IoT stakeholders, and according to their needs, proposes five different modelling languages. Those models are successively used for code generation and task mapping techniques. Similarly to DiaSuite [5], while sharing the same goals of A4WSN, the framework in [45] covers a different (still, related) domain.

Domain-specific modelling languages for WSN: many approaches propose to use DSMLs for representing WSNs from different viewpoints. For example, in [62] the proposed modelling language contains concepts such as node group, region, resource, wireless link; whereas, in [16] authors propose a set of languages spanning from application-level actions (e.g. sense, send message, store data) to hardware specifications (e.g. processor, sensing devices, radio transceivers), and so on. In [4] the authors propose Verisensor, a DSML based on concepts such as system, node class, application with the possibility to automatically translate models towards a formal language for checking the lifetime of the WSN and its correct behaviour.

In [13], the authors propose the LWiSSy domain-specific language for wireless sensor and actuator network systems. The LWiSSy metamodel comprises three views: structural behavioural, and optimization. Those three views are described in details, and successively evaluated through a controlled experiment.

Other approaches, such as those proposed in [17,38], are based on *generic modelling languages*. They mainly use extensions of UML and Simulink for representing a WSN.

In order to better understand how MDE has been used for designing and analysing wireless sensor networks, [30] surveys and classifies state-of-the-art MDE approaches for engineering WSNs.

Other modelling and analysis approaches for engineering WSNs: describing a network from a *structural* point of view is very straightforward and easy to reason about (just think about the component-based representation in OMNeT++,[8] one of the most popular network simulators). Also, an approach based on DDS (i.e. the data centric middleware standard introduced by OMG) is presented in [3]; the authors proposed four types of modelling languages (namely for data types, data space, node structure, and node configuration) and use them as input for a set of optimization and transformation steps, eventually delivering deployable application code as output.

Also, in some cases (e.g. when capabilities such as fault tolerance and security analysis are needed) the structure of WSNs may not be enough, and thus describing the *behaviour* of the WSN is fundamental. In [19], the authors address energy-aware system design of wireless sensor networks (WSNs). Energy mode signalling and energy scheduling of nodes within a WSN are represented as SDL models, and then analysed.

Rodrigues et al. in [49] proposes an MDA process where application domain experts model the Platform-Independent Model (PIM) of a WSN application. Such a PIM is successively transformed into a platform-specific model (PSM) and refined by a network expert. Class and Activity diagrams are used to specify the WSN application at the PIM level, while Component and Finite State diagrams are used at the PSM level.

An approach for formal modelling and analysis of WSN in Real-Time Maude is presented in [41]. In [54] Samper et al. propose the GLONEMO formal model for the analysis of ad-hoc sensor networks.

For what concerns the *physical environment* of a WSN, the majority of approaches in the literature does not allow designers to specify the *physical deployment* of the WSN nodes. Among those that support (in some form) this feature, there is great variability. There are some approaches which support an explicit definition of the physical environment (e.g. in [16] the tool allows engineers to model real-world dimensions, obstacles with attenuation coefficients, etc.); others allow designers to define physical quantities (e.g. in [4] engineers can define models of the evolution of each physical quantity in a given scenario), and so on. However, all these approaches do not provide any intuitive and abstract means to easily define the deployment environment of the WSN. A recent study [31] has investigated how WSN engineers currently specify the physical environment and how they would like to do it.

Surveys related to the modelling and analysis of WSNs: A survey on system models in WSNs has been conducted in [58]: there the authors identify several dimensions to be used to classify model (types) used to specify networked computing systems (from models of signal propagation, to models of the application). Existing models are then organized into a taxonomy. In [25] the authors survey 9 WSN modelling techniques. Through this study, they show how each technique models different parts of the system. The models here analysed are extensions to existing notations, such as SDL, Promela, UML, and others.

Final Remarks: A4WSN shares with some of the related approaches above the wish to provide a *clear separation of concerns* between different modelling views, to enhance *reuse*, to *abstract* from low-level details, and to support *early analysis* of WSN applications. What distinguishes A4WSN from other related work are (i) the modelling languages that have been selected for modelling WSN applications, including an explicit graphical modelling of the application physical environment, (ii) the definition of models dedicated to the weaving of the three main modelling languages, (iii)

[8] http://www.omnetpp.org/.

the existence of an extensible programming framework that enables third-party researchers and developers to reuse the A4WSN modelling environment and programming framework when developing new analysis and code generation engines. In this context, third-party researchers can focus exclusively on solving their peculiar issues, while spending minimal effort and implementation time on realizing the facilities already provided by A4WSN out of the box. (iv) The maturity of A4WSN with respect to other approaches that, while sharing some of our desires, seem to still implement only a subset of them.

8 Conclusion and future work

This paper presents a modelling platform supported by a dedicated programming framework for the model-driven engineering of wireless sensor networks. The modelling viewpoints and conceptual elements have been carefully designed in collaboration with colleagues from various domains, such as software engineering, wireless sensor networks, and telecommunications. The programming framework functioning has been tested by realizing a plug-in devoted to energy-related simulation of WSNs.

The modelling and programming framework presented in this paper represents the (starting but mandatory) foundation for a series of goals we are willing to achieve in the midterm.

Firstly, we plan to have the framework used by practitioners involved in the development of WSNs. We wish to record and analyse their usage patterns and collect their feedback for further improving our platform. At the time of writing, the framework is being used by master students to model and analyse course projects, and it is currently used in situational awareness projects handled by one of the co-authors.

Secondly, we are aware that it might be necessary to extend the modelling languages to provide additional concepts for supporting new analysis or code generation engines. For example, we are working on providing new SAML data structures (either primitive or structured), new attributes for better specifying nodes in the NODEML modelling language, and on the extension of the purposefully simple ENVML modelling language (e.g. by adding multi-floor support for indoor deployments, by supporting the specification of properties specific to outdoor set-ups). In this context, introducing changes at the metamodel level might have a strong impact on the already developed plug-ins (model editors, model transformations, etc.). This problem is called metamodel co-evolution management, and it is well-known in the MDE research field [12,51]. If we look at this problem from a different perspective, similarly to what we proposed in a previous work on architectural language interoperability [53], a possible solution could be to provide a systematically defined extension process for our modelling languages. According to this extension process, languages extensions are organized into a hierarchy obtained by systematically extending a root modelling language. Under this perspective, we plan to build on (and adapt, if needed) metamodel co-evolution techniques [23,52] in order to tackle this problem.

Thirdly, we would like to realize an analysis plug-in that, while getting in input a series of environmental configurations options, can tell us which configuration can increase the network lifetime (so far, PlaceLife can evaluate the expected lifetime of a given configuration, but is quite impractical to analyse alternative solutions). We plan to use genetic algorithms and search based approaches to achieve such a goal.

Finally, we are working on a WSN performance analysis plug-in that allows engineers to run a trade-off analysis between energy consumption and performance indices like sensor node throughput, reliability, and network latency.

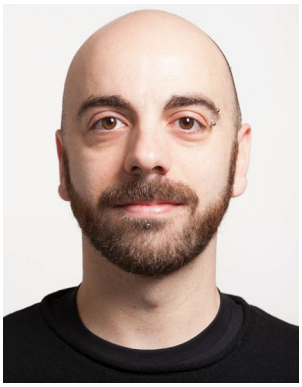
Acknowledgements Funding was provided by RIDITT (Grant No. Italian government).

References

1. Al-karaki, J.N., Kamal, A.E.: Routing techniques in wireless sensor networks: a survey. IEEE Wirel. Commun. **11**, 6–28 (2004)
2. Alemdar, H., Ersoy, C.: Wireless sensor networks for healthcare: a survey. Comput. Netw. **54**(15), 2688–2710 (2010)
3. Beckmann, K., Thoss, M.: A model-driven software development approach using OMG DDS for wireless sensor networks. In: Proceedings of the 8th IFIP WG 10.2 International Conference on Software Technologies for Embedded and Ubiquitous Systems, SEUS'10, pp. 95–106 (2010)
4. Ben Maïssa, Y., Kordon, F., Mouline, S., Thierry-Mieg, Y.: Modeling and analyzing wireless sensor networks with VeriSensor. In: Petri Net and Software Engineering (PNSE), vol. 851, pp. 60–76. CEUR, Hamburg, Germany (2012)
5. Bertran, B., Bruneau, J., Cassou, D., Loriant, N., Balland, E., Consel, C.: DiaSuite: A tool suite to develop sense/compute/control applications. Sci. Comput. Program. **79**, 39–51 (2014). Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010)
6. Bjornemo, E., Johansson, M., Ahlen, A.: Two hops is one too many in an energylimited wireless sensor network. In: Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 181–184 (2007)
7. Blumenthal, J., Handy, M., Golatowski, F., Haase, M., Timmermann, D.: Wireless sensor networks—new challenges in software engineering. In: Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference, vol. 1, pp. 551–556 (2003)
8. Bryant, B.R., Gray, J., Mernik, M., Clarke, P.J., France, R.B., Karsai, G.: Challenges and directions in formalizing the semantics of modeling languages. Comput. Sci. Inf. Syst. **8**(2), 225–253 (2011)
9. Chandra, T.B., Dwivedi, A.K.: Programming languages for wireless sensor networks: a comparative study. In: Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on, pp. 1702–1708. IEEE (2015)
10. Chang, X.: Network simulations with OPNET. In: Proceedings of the 31st Conference on Winter Simulation: Simulation—a Bridge to the Future—Volume 1, WSC '99 (1999)

11. Cheng, C., Lu, R., Petzoldt, A., Takagi, T.: Securing the internet of things in a quantum world. Commun. Mag. **55**(2), 116–120 (2017)
12. Cicchetti, A., Di Ruscio, D., Eramo, R., Pierantonio, A.: Automating co-evolution in model-driven engineering. In: 12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, 15–19 September 2008, Munich, Germany, pp. 222–231. IEEE Computer Society (2008)
13. Dantas, P., Rodrigues, T., Batista, T., Delicato, F., Pires, P., Li, W., Zomaya, A.: Lwissy: a domain specific language to model wireless sensor and actuators network systems. In: 2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA), pp. 7–12 (2013)
14. Demirkol, I., Ersoy, C., Alagoz, F.: MAC protocols for wireless sensor networks: a survey. IEEE Commun. Mag. **44**(4), 115–121 (2006). https://doi.org/10.1109/mcom.2006.1632658
15. Di Ruscio, D., Malavolta, I., Muccini, H., Pelliccione, P., Pierantonio, A.: Developing next generation ADLs through MDE techniques. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering, vol. 1, pp. 85–94. IEEE (2010)
16. Doddapaneni, K., Ever, E., Gemikonakli, O., Malavolta, I., Mostarda, L., Muccini, H.: A model-driven engineering framework for architecting and analysing wireless sensor networks. In: SESENA, pp. 1–7 (2012)
17. 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, pp. 8–13 (2010)
18. Goldsmith, A.: Wireless Communications. Cambridge University Press, New York (2005)
19. Gotzhein, R., Krämer, M., Litz, L., Chamaken, A.: Energy-aware system design with SDL. In: Proceedings of the 14th International SDL Conference on Design for Motes and Mobiles, SDL'09, pp. 19–33. Springer, Berlin (2009)
20. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: Proceedings of the 33rd Annual Hawaii International Conference on System Sciences (HICSS), Washington, DC, USA (2000)
21. Hill, J.L.: System architecture for wireless sensor networks. Ph.D. thesis, University of California, Berkeley (2003). AAI3105239
22. Huang, C.F., Tseng, Y.C.: The coverage problem in a wireless sensor network. In: Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications, WSNA '03, pp. 115–121 (2003)
23. Iovino, L., Pierantonio, A., Malavolta, I.: On the impact significance of metamodel evolution in mde. J. Object Technol. **11**(3), 1–33 (2012)
24. ISO/IEC/IEEE: ISO/IEC/IEEE 42010:2011 Systems and software engineering – Architecture description (2011)
25. Khalil, J., Liscano, J.R., Bradbury, J.: A survey of modeling techniques for wireless sensor networks. In: SENSORCOMM 2011, The Fifth International Conference on Sensor Technologies and Applications, pp. 103–109 (2011)
26. Lorincz, K., Malan, D., Fulford-Jones, T., Nawoj, A., Clavel, A., Shnayder, V., Mainland, G., Welsh, M., Moulton, S.: Sensor networks for emergency response: challenges and opportunities. IEEE Pervasive Comput. **3**(4), 16–23 (2004). https://doi.org/10.1109/MPRV.2004.18
27. Losilla, F., Vicente-Chicote, C., Álvarez, B., Iborra, A., Sánchez, P.: Wireless sensor network application development an architecture-centric MDE approach. In: Oquendo, F. (ed.) ECSA, LNCS, vol. 4758, pp. 179–194. Springer, Berlin (2007)
28. Malavolta, I.: A4WSN—Programming Framework and Implementation details (2018). http://a4wsn.di.univaq.it/files/a4wsnLanguages.pdf. Accessed 4 April 2018
29. Malavolta, I., Mostarda, L., Muccini, H., Doddapaneni, K.: The A4WSN Modelling languages (2018). http://a4wsn.di.univaq.it/files/a4wsnLanguages.pdf. Accessed 4 April 2018
30. Malavolta, I., Muccini, H.: A Study on MDE approaches for engineering wireless sensor networks. In: Proceedings of the 40th Euromicro Conference series on Software Engineering and Advanced Applications (SEAA), August 2014 (2014)
31. Malavolta, I., Muccini, H.: A Survey on the specification of the physical environment of wireless sensor networks. In: Proceedings of the 40th Euromicro Conference series on Software Engineering and Advanced Applications (SEAA), August 2014 (2014)
32. Malavolta, I., Muccini, H., Pelliccione, P., Tamburri, D.: Providing architectural languages and tools interoperability through model transformation technologies. IEEE Trans. Softw. Eng. **36**(1), 119–140 (2010)
33. Medvidovic, N., Dashofy, E.M., Taylor, R.N.: Moving architectural description from under the technology lamppost. Inf. Softw. Technol. **49**(1), 12–31 (2007)
34. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005)
35. Mottola, L., Pathak, A., Bakshi, A., Prasanna, V., Picco, G.: Enabling scope-based interactions in sensor network macroprogramming. In: IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, 2007. MASS 2007. pp. 1–9 (2007)
36. Mottola, L., Picco, G.P.: Programming wireless sensor networks: Fundamental concepts and state of the art. ACM Comput. Surv. **43**, 19:1–19:51 (2011)
37. Mottola, L., Picco, G.P.: Middleware for wireless sensor networks: an outlook. J. Internet Serv. Appl. **3**(1), 31–39 (2012)
38. Mozumdar, M., Gregoretti, F., Lavagno, L., Vanzago, L., Olivieri, S.: A framework for modeling, simulation and automatic code generation of sensor network application. In: 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08, pp. 515–522 (2008)
39. Mozumdar, M.M.R., Gregoretti, F., Lavagno, L., Vanzago, L., Olivieri, S.: A framework for modeling, simulation and automatic code generation of sensor network application. In: SECON, pp. 515–522 (2008)
40. Newport, C., Kotz, D., Yuan, Y., Gray, R.S., Liu, J., Elliott, C.: Experimental evaluation of wireless simulation assumptions. Simulation **83**(9), 643–661 (2007)
41. Olveczky, P., Thorvaldsen, S.: Formal modeling and analysis of wireless sensor network algorithms in real-time Maude. In: 20th International Parallel and Distributed Processing Symposium, 2006. IPDPS 2006, p. 8 (2006). https://doi.org/10.1109/IPDPS.2006.1639414
42. Pahlavan, K., Krishnamurthy, P.: Networking Fundamentals: Wide, Local and Personal Area Communications. Wiley, New York (2009)
43. Pahlavan, K., Krishnamurthy, P.: Networking Fundamentals. Wiley, Chichester (2009)
44. Paige, R.F., Kolovos, D.S., Polack, F.A.: A tutorial on metamodelling for grammar researchers. Sci. Comput. Program. **96, Part 4**, 396–416 (2014)
45. Patel, P., Pathak, A., Cassou, D., Issarny, V.: Enabling high-level application development in the internet of things. In: Zuniga, M., Dini, G. (eds.), Sensor Systems and Software, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 122, pp. 111–126 (2013)
46. Pediaditakis, D., Tselishchev, Y., Boulis, A.: Performance and scalability evaluation of the castalia wireless sensor network simulator. In: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, SIMUTools '10, pp. 53:1–53:6 (2010)

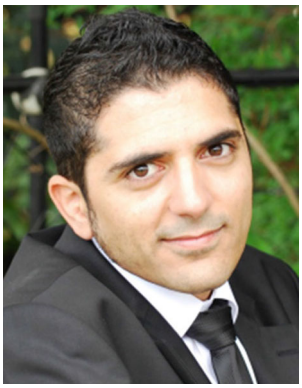
47. Picco, G.P.: Software engineering and wireless sensor networks: happy marriage or consensual divorce? In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research. FoSER. NY, USA (2010)
48. Rappaport, T.: Wireless communications: principles and practice. Prentice Hall communications engineering and emerging technologies series. Prentice Hall PTR (1996)
49. Rodrigues, T., Batista, T., Delicato, F., Pires, P., Zomaya, A.: Model-driven approach for building efficient wireless sensor and actuator network applications. In: 2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA), pp. 43–48 (2013)
50. Romer, K., Mattern, F.: The design space of wireless sensor networks. IEEE Wirel. Commun. **11**(6), 54–61 (2004)
51. Rose, L., Etien, A., Méndez, D., Kolovos, D., Paige, R., Polack, F.: Comparing model-metamodel and transformation-metamodel coevolution. In: International Workshop on Models and Evolutions (2010)
52. Ruscio, D.D., Iovino, L., Pierantonio, A.: Coupled evolution in model-driven engineering. IEEE Softw. **29**(6), 78–84 (2012)
53. Ruscio, D.D., Malavolta, I., Muccini, H., Pelliccione, P., Pierantonio, A.: Model-driven techniques to enhance architectural languages interoperability. In: FASE, pp. 26–42 (2012)
54. Samper, L., Maraninchi, F., Mounier, L., Mandel, L.: Glonemo: Global and accurate formal models for the analysis of ad-hoc sensor networks. In: Proceedings of the First International Conference on Integrated Internet Ad Hoc and Sensor Networks, InterSense '06. New York, NY, USA (2006)
55. Seybold, J.S.: Introduction to RF Propagation. Wiley, Newark (2005)
56. Shimizu, R., Tei, K., Fukazawa, Y., Honiden, S.: Model driven development for rapid prototyping and optimization of wireless sensor network applications. In: Proceedings of SESENA '11, pp. 31–36. ACM, New York, NY, USA (2011)
57. Stankovic, J.A.: Research challenges for wireless sensor networks. SIGBED Rev. **1**, 9–12 (2004)
58. Stanley-Marbell, P., Basten, T., Rousselot, J., Oliver, R.S., Karl, H., Geilen, M., Hoes, R., Fohler, G., Decotignie, J.D.: System models in wireless sensor networks. Technical Report ESR-2008-06, Eindhoven University of Technology (2008)
59. Szyperski, C.: Component Software. Beyond Object Oriented Programming. Addison Wesley, Boston (1998)
60. van Dam, T., Langendoen, K.: An adaptive energy-efficient mac protocol for wireless sensor networks. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys '03, pp. 171–180. New York, NY, USA (2003)
61. Varga, A., Hornig, R.: An overview of the OMNeT++ simulation environment. In: Simutools '08: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, pp. 1–10 (2008)
62. Vicente-Chicote, C., Losilla, F., Álvarez, B., Iborra, A., Sánchez, P.: Applying MDE to the development of flexible and reusable wireless sensor networks. Int. J. Coop. Inf. Syst. **16**(3/4), 393–412 (2007)
63. Ye, W., Heidemann, J., Estrin, D.: Medium access control with coordinated adaptive sleeping for wireless sensor networks. IEEE/ACM Trans. Netw. **12**(3), 493–506 (2004)



Ivano Malavolta is Assistant Professor at the Vrije Universiteit Amsterdam, The Netherlands. His research focuses on empirical software engineering, software architecture, model-driven engineering (MDE), and mobile-enabled systems. He is programme committee member and reviewer of international conferences and journals in his fields of interest. He authored more than 80 scientific articles in international journals and peer-reviewed international conferences proceedings. He received a Ph.D. in computer science from the University of L'Aquila in 2012. He is a member of ACM and IEEE. More information is available at http://www.ivanomalavolta.com.

Leonardo Mostarda is an Associate Professor at Camerino University, Department of Computer Science, Italy. He got his Ph.D. in 2006 at the Computer Science Department of University of L'Aquila. Afterwards, he cooperated with the European Space Agency (ESA) on the CUSPIS FP6 project to design and implement novel security protocols and secure geo-tags for works of art authentication. In 2007 he was Research Associate at the Computing Department, Distributed System and Policy Group, Imperial College, London. There he was working on the UBIVAL EPRC project in cooperation with Cambridge, Oxford, Birmingham, and UCL for building a novel middleware to support the programming of body sensor networks. In 2010 he was Senior Lecturer at Middlesex University in the Distributed Systems and Networking Department. There he funded the Senso LAB an innovative research laboratory for building energy efficient wireless sensor networks.

Henry Muccini is an Associate Professor in Computer Science from the University of L'Aquila, Italy. He received is Ph.D. degree from the University of Rome—La Sapienza in 2002, and he has been visiting professor at the University of California, Irvine. His research interests are on software architecture, model-driven engineering, and mobile computing. I am an associate editor in chief for IEEE software, a member of the IFIP WG 2.10 on Software Architecture, and the Director of the Living Lab and of the CINI laboratory on Smart Cities and Communities at the University of L'Aquila. More detailed information may be found at http://www.HenryMuccini.com.



Enver Ever obtained his B.Sc. Degree from the Department of Computer Engineering, Eastern Mediterranean University, Cyprus, in 2002. He then continued his studies at Middlesex University where he obtained his M.Sc. in Computer Networks and Ph.D. in Performance Evaluation of Computer Networks and Communication Systems in 2004 and 2008, respectively. He worked at Bradford University as a postdoctoral Research Associate for a year. Following that he worked as a Lecturer/Senior Lecturer in Computer and Communications Engineering Department Middlesex University. Currently, he is an Associate Professor in Middle East Technical University, Northern Cyprus Campus. His current research interests include computer networks, wireless communication systems, Internet of Things, wireless sensor networks, parallel computing paradigms, integrated circuits, and performance/reliability modelling. He serves on various Programme Committees and received exemplary reviewer award for his contributions as reviewer.

Krishna Doddapaneni is currently a Solution Architect and Researcher at Altiux Innovations Inc, USA, where he emphasizes on architecting best fit solutions by providing technology direction, ensure project implementation compliance, and utilize technology research to innovate, integrate, and manage technology solutions. Additional to his current role, he is also a co-founder of Hotaru Labs, Technical advisor to KeyPoint Technologies and 3LoQ Labs. Before joining his current position, he was a postdoc at University of Minnesota, USA, focusing on enabling reliable autonomous localization of transient RF sources using UAVs, for search and rescue missions in the absence of GPS and enabling reliable and delay efficient communication in UAVs (single and multi-hop communication). He graduated in Electronics and Communications Engineering from JNTU, India, in 2008, M.Sc. in Computer Networks and Ph.D. in Computer Communications from Middlesex University in 2010 and 2014, respectively. He visited University of Camerino and University of L'Aquila as an Academic Researcher. His doctoral research focused on cross-layer approaches for wireless sensor network's energy efficiency, with an emphasis on the interaction between the physical, medium access and application layers. His main research interests include robotic sensor networks, wireless sensor networks, IoT, and M2M communications. He frequently collaborates with some of the best talent in the industry and scholars from academia to create and implement innovative high-quality solutions and participate in sales and various pursuits focused on business needs.

Orhan Gemikonakli is an Honorary Professor in telecommunications at University of Middlesex, UK. He received is Ph.D. degree from the King's college London, UK, and he has been visiting professor at the University of Camerino, Italy. His current research interests include computer networks, wireless communication systems, Internet of Things, wireless sensor networks, parallel computing paradigms, integrated circuits, and performance/reliability modelling. He serves in various Programme Committees, and he is reviewer for various journals. He is a member of ACM and IEEE.