Process-Based Design and Integration of Wireless Sensor Network Applications

Stefano Tranquillini[1], Patrik Spieß[2], Florian Daniel[1], Stamatis Karnouskos[2], Fabio Casati[1], Nina Oertel[2], Luca Mottola[3], Felix Jonathan Oppermann[4], Gian Pietro Picco[1], Kay Römer[4], and Thiemo Voigt[3]

[1] University of Trento, Via Sommarive 5, 38123 Povo (TN), Italy
(tranquillini,daniel,casati,picco)@disi.unitn.it
[2] SAP AG, Vincenz-Prießnitz-Straße 1, 76131 Karlsruhe, Germany
(parik.spiess,stamatis.karnouskos,nina.oertel)@sap.com
[3] Swedish Institute of Computer Science, Isafjordsgatan 22, Kista, Sweden
(luca,thiemo)@sics.se
[4] University of Lübeck, Ratzeburger Allee 160 23562 Lübeck, Germany
(oppermann,romer)@iti.uni-luebeck.de

Abstract Wireless Sensor and Actuator Networks (WSNs) are distributed sensor and actuator networks that monitor and control real-world phenomena, enabling the integration of the physical with the virtual world. They are used in domains like building automation, control systems, remote healthcare, etc., which are all highly process-driven. Today, tools and insights of Business Process Modeling (BPM) are not used to model WSN logic, as BPM focuses mostly on the coordination of people and IT systems and neglects the integration of embedded IT. WSN development still requires significant special-purpose, low-level, and manual coding of process logic. By exploiting similarities between WSN applications and business processes, this work aims to create a holistic system enabling the modeling and execution of executable processes that integrate, coordinate, and control WSNs. Concretely, we present a WSN-specific extension for Business Process Modeling Notation (BPMN) and a compiler that transforms the extended BPMN models into WSN-specific code to distribute process execution over both a WSN and a standard business process engine. The developed tool-chain allows modeling of an independent control loop for the WSN.

1 Introduction

Today there is still lack of high-level, model-driven programming tools for Wireless Sensor and Actuator Network (WSN) applications and the integration with enterprise services requires significant effort and expertise in embedded programming of WSNs. Organizations are reluctant to install large-scale WSNs, as this still requires significant, costly, low-level programming of sensing and actuation logic for the WSN, in addition to the physical deployment of the WSN nodes (e.g., inside a building). Additionally, setting up the communication channel between a WSN and an enterprise's information system requires an even larger set

of technologies and manually writing of custom code. Domain experts typically lack the necessary low-level programming skills.

To foster widespread adoption and more efficient use of sensor networks for enterprise information systems, a need for a specifically tailored integration technique that is able to bring together sensor networks and business applications [1] is perceived. The aim is to drastically improve the ease of programming of WSNs by enabling the graphical *modeling* of WSN applications, leaving low-level details to a model compiler and a run-time system. WSN programming should be accessible to domain experts, such as business process modelers. They should further be empowered to design the WSN's interaction with enterprise information systems using the methods of business process modeling they are familiar with. Our approach aims to:

- Provide a *conceptual model* that abstracts typical WSN programming knowledge into reusable tasks that can be integrated into modeling notations, such as the Business Process Modeling Notation (BPMN).
- Develop an *extension of BPMN* [2], BPMN4WSN, that enables the graphical modeling of WSN applications and their integration with BPs based on an abstraction layer that hides low-level details of the sensor network.
- Introduce *tools* that enable the design, deployment, and execution of integrated WSN/BP applications. We do not reuse existing APIs toward the WSN; we *program* the WSN and automatically generate the necessary APIs.
- Evaluate our approach with a realistic *prototype* deployment, including a self-optimizing run-time system layer, and a report on the first experiences with its usage in the context of the EU project `make`*Sense*.

In order to create applications that span both a BPMN process and a WSN application, knowledge in both fields is required. We do not expect the *application developer* (the domain expert) to model an executable process. Rather, we suggest a two-phase approach, where a descriptive process model is created by the developer, which is then refined by a more technical *system developer* using the WSN extension integrated in the process diagram.

In the following, we outline an application scenario to better describe our approach. Then, in Section 3, the typical characteristics and components of WSNs are analyzed. In Section 4 it is outlined how the challenges identified in the scenario are approached conceptually, and in Section 5 the according extension of BPMN is described. Subsequently in Section 6 the implementation of the prototype, including the code generation logic for WSNs is described. Section 7 critically discusses the results achieved so far. Section 8 reviews related work before concluding the paper.

2 Scenario: Convention Center HVAC Management

Our application scenario showcases the operation of a convention center (see Figure 1) that has a variety of meeting rooms, which can be booked for various events. Each room can be booked at a rate that partly depends on room

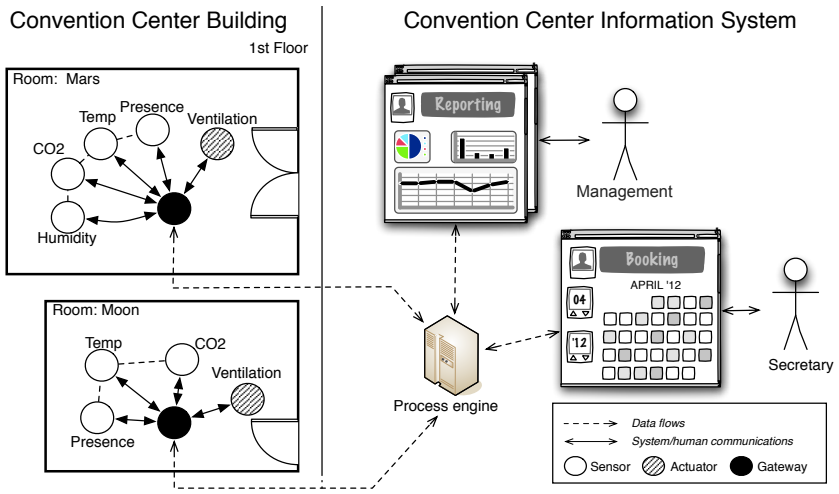

Figure 1. Integration of a convention center's BP engine with a WSN for HVAC.

characteristics (e.g., its size) and partly on the energy consumption of the event organized in the room. For this purpose, the convention center is equipped with a Heating, Ventilation, and Air Conditioning (HVAC) system including a WSN, which ensures comfortable levels of temperature, humidity, and CO_2 for each individual room for the booked duration of the respective event. In order to do so, the HVAC system must be instructed automatically by the convention center's information system about when to activate the ventilation and how long to control the room's temperature and CO_2 concentration for each room. Room conditions are only maintained during the booked times to save energy and only if presence of people is detected by presence sensing; air conditioning is shut off when a meeting is not attended at all or ends prematurely. In turn, the HVAC system feeds back sensor data to the information system, which allow the information system to precisely compute the HVAC cost for each individual event. The information system is used for the booking of rooms, the reporting on energy consumption, and the billing of customers. This mode of operation is more energy efficient that today's common practice, where one would simply run the HVAC system at a fixed rate, independently of room occupation or environmental conditions — a practice that wastes much energy.

Technically, it is necessary to develop (i) the BP logic running inside the BP engine, (ii) the code running on the nodes inside the WSN, and (iii) a suitable set of communication endpoints supporting the interaction of the BPM with the WSN and vice versa. Note that it is *not* the goal of this work to optimize convention center operation or more generally building automation, but to provide a basic set of abstractions, tools, and methodologies that can be used in all scenarios where also WSNs are used. We use it merely as a device to depict a concrete application of our approach.

3 Relevant Properties of Wireless Sensor Networks

Before going into the details of the approach, the special properties of WSNs that are relevant at the application layer and that therefore underpin our model of the system are explained. A WSN is a distributed system, namely a network of wireless, battery-powered, autonomous, small-scale devices, so called nodes, each of which is equipped with one or more sensors or actuators or both. Nodes are battery-powered and replacing the battery is mostly not intended or not feasible from a Total Cost of Ownership (TCO) perspective. Therefore, they make use of ultra-low-power hardware, that is drastically limited in processing power, memory, and transmission bandwidth and the application software running on the nodes, including wireless communication protocols, needs to be optimized for low power consumption to extend network lifespan. These limits typically prevent executing a regular BPMN engine on the devices that interprets BPMN models serialized as XML.

Sensors are used to sense information from the real world (e.g., temperature) while actuators perform actions that change the state of the environment (e.g., control a motor or a lamp). The typical number of nodes inside a network can vary from a few to hundreds or even thousands. Via radio links, a node can generally communicate with all other nodes in its transmission range and with nodes further away by multi-hop, routed communication. WSNs are able to self-organize, overcome network failure, and execute distributed computation logic, such as computing the average of sensor values while those are routed to a destination node. Often, WSNs are composed of heterogeneous nodes, each equipped with a custom set of sensors, so that, for example, one type of node can sense CO_2 and humidity while another type of node is able to control an automatic door, while a third has enough special hardware to compute complex arithmetics.

As a basis for modeling WSN application logic, a very simple model of the physical set-up that is sensed and acted upon is assumed: A given WSN monitors real world entities, each is referred to as an Entity of Interest (EoI) which can be a *location* or a *thing.* A thing is any physical object, while a location is a space that the sensor network is monitoring, e.g., a room or a building. A domain expert is usually only interested in the EoIs and the operations that can be applied to them, but not in the technical layer of sensors that sense or the actuators that influence them.

To overcome the limitations of WSN hardware and to maximize efficiency of operations, the research community has introduced a large number of programming abstractions to program wireless sensor networks [3]. By abstracting existing programming concept into high-level constructs [4] (described in Section 5.2) and assuming that all existing functionality can be expressed using them, one can use high-level constructs as basic building blocks for graphical modeling. Usually, a sensor network will perform some or all of these tasks:

Sensing: measuring one or more environmental parameters of an EoI, such as temperature or humidity, making use of the sensing equipment of the nodes.

Actuation: enacting operations physically affecting an EoI, e.g., controlling or moving it or flashing a LED. WSNs are often used to actuate or control the environment in reaction to sensed parameters, creating a control loop (as the actuation eventually triggers changes in the sensed values).

Task distribution: distributing operations that coordinate a subset of nodes, e.g., any in-network aggregation on the input values or the election of a controller node based on certain criteria. As WSNs consist of several nodes, several of which can monitor the same EoI, especially data aggregation operations are often required, e.g., to compute the average temperature of a room observed by many sensor nodes.

From the perspective of a domain expert, it is irrelevant which part of a WSN performs a task, e.g., whether an operation is carried out by a single node or the network as a whole as long as the operations are addressable by an EoI.

4 Requirements and Approach

In the convention center scenario, there is a need for collaboration between the reservations and billing systems in the back-end and the sensor network that executes the sensing and actuating operations. Thus, the application runs on different types of systems which can be seen as two distinct participants in the process. This raises the need to model both the intra-WSN logic and its interactions with back-end systems as a collaboration of two process participants. While the back-end part is orchestrated using classical Business Process Management (BPM), modeling the process logic to be executed inside the WSN needs certain provisions (e.g., model extensions) to enable the specification of WSN logic in a high-level fashion and the creation of code that can be executed in the network.

Typically, the integration of WSNs into BPs is based on the invocation of services exposed by the network [5,6,7]. This results in a modeling approach that uses the network as set of available operations on which a process can be constructed, but that prohibits the programming of the WSN itself. This limits the possibility to define custom WSN logic to be carried out by the network as part of the process. Instead, the key idea of our approach is to develop a business process modeling notation that allows a domain expert to program both the BP *and* the actual network logic, without the need to know and specify all the low-level details. The created process model is later used to derive the code that will be executed by the WSN. In this way, the WSN logic is fully specified at the process level.

The specific requirements we identify can be divided into supporting *modeling*, *deployment* and *runtime*. Supporting modeling means defining a modeling paradigm that fits the needs of a domain expert and integrates back-end business processes and WSN logic using a single modeling language. This requires to:

- Provide an easy to understand and familiar way of expressing WSN logic; enable integrating WSN processes into back-end processes, coupling them and allowing for easy data sharing.

– Define a set of concepts to describe the logic and operations that can be combined for creating reusable, high-level WSN modeling constructs. We have to supply the modeler with the possibility to specify operations like sense, actuate and aggregate for measurements over EoIs.
– Model WSN capabilities and details. WSNs are usually heterogeneous regarding the type of sensors and actuators. Knowing the characteristics of the network is fundamental to have an overview of which things and locations can be controlled and monitored by the WSN as well as which operations the WSN is able to perform. Having such a model will give the domain expert the ability to express the desired processes in the familiar terms of EoIs and irrespective of technical systems.
– Supporting the modeler in designing only feasible processes by restricting the available modeling constructs to him to what the WSN is capable of executing.

Supporting the deployment of the process requires to:

– Split the process model into an intra-WSN part and a WSN-aware part (back-end). The process is divided between two actors that participate in the execution. These two parts of the process have to be separated and handled differently.
– Create WSN binary code. The intra-WSN part of the process has to be translated to binary code and injected into the nodes. This code is generated based on the flow of the process model and tasks that describe the operations.
– Create the endpoints and communication channels to handle the messages from and to the network. After having split the process in two parts and after having translated the WSN part into binary code the communication between these two participants has to be guaranteed. To do so, the endpoints and the communication channels through which the messages will be sent/received need to be available.

Supporting the execution requires to:

– Provide a process engine to execute the WSN-aware business process part. The process engine also handles the communication with the WSN.
– Run the code in the WSN. Part of the process actually runs inside the network without the need for external communication and control. The process is executed on the gateways and the actions are distributed on the nodes, guaranteeing the correctness of the process depicted by the modeler.

Figure 2 illustrates the conceptual model of how we approach WSN programming. The model is not meant to be an extension of the BPMN meta-model. Only part of it is related to BPMN4WSN, the other part is related to our own modeling formalism for the definition of low-level WSN logic. The two entities on the top represent the physical WSN. The *WSN Operation* represents an action executed by a *node*, which can be a sensor or an actuator node. We allow the domain expert to model WSN operations via a dedicated task type, the *WSN*

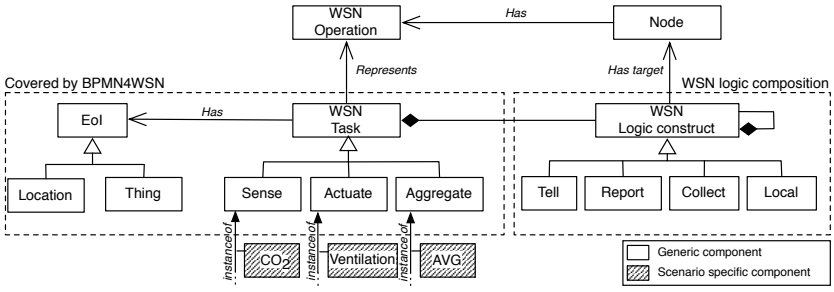

Figure 2. Conceptual model of WSN operations

Tasks. A *WSN Task* is a generic action that can be executed by the network and can be used to express *sense*, *actuate*, and *aggregate* operations. Each of these action types has concrete implementations; for example, in our scenario, *Sense* CO_2 is a concrete implementation of a sensing action. The *WSN Task* is logically connected to an EoI, which allows the modelers to scope the action. That is, the EoI specifies *where* the action will be executed; it could be a *thing* or a *location*. *WSN Task* and EoI represent the high-level constructs used to model WSN logic in BPMN4WSN. This level of abstraction is however not enough to describe all the needed details to generate binary code that runs on the nodes. The detailed specification is based on *WSN logic constructs*, which abstract operations that can be configured (e.g., by adding a concrete target node resolving a logical EoI) and translated into binary code. The composition of WSN logic components (the *WSN logic composition* box) allows the system developer to refine the process model designed by the domain expert and to fill WSN tasks with concrete logic.

Figure 3 shows the architecture of the tool chain for developing WSN/BP applications containing an extended BPMN editor in which the process is modeled, and a compiler for translating the high-level specifications into low-level executable binary code for the sensor network and for the process engine. Next, the *modeling* and *deployment* part are discussed in more detail; a first prototype of the tool is discussed afterwards.

5 BPMN4WSN

As illustrated in Figure 3, two types of developers jointly develop a process model: the application developer and the system developer. The application developer is the person who models the coarse process; he is an expert of the domain with experience in business process modeling and has some WSN background. The system developer is a WSN expert and has the task of creating the refined, XML-formated model of the system (see the bottom left corner of Figure 3). This model contains information of the network such as the EOIs, nodes and available sense and actuate operations. The two roles collaborate mainly in the

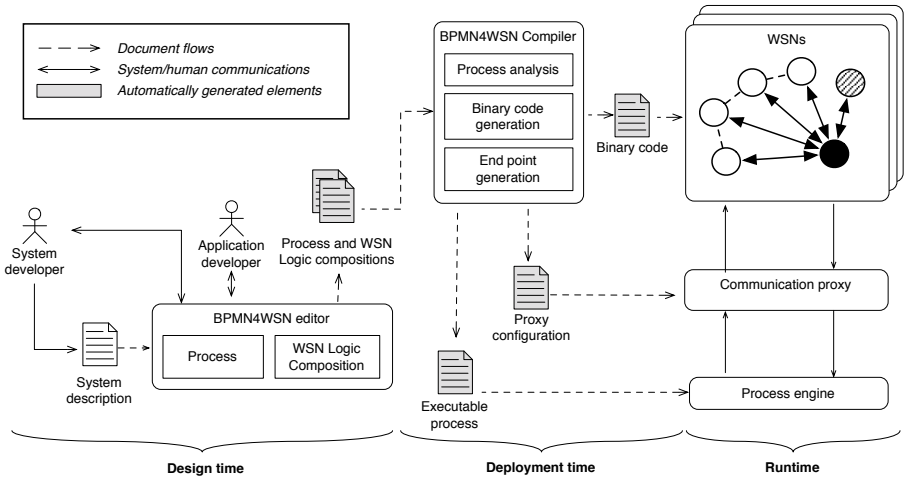

Figure 3. Architecture

design of WSN Tasks. The application developer creates a process that crosses the system boundary between standard IT and WSN including the specification of the behavior of the latter. He defines a descriptive, not yet executable version of the process. For instance, in the convention center use case, the application developer would specify a task for reading the latest sensor values or driving an actuator based on the system descriptor model. Later, the system developer would refine this model by adding WSN logic components to make the tasks that involve the WSN executable.

5.1 Process Logic

In our solution the design of the business process is mainly carried out by the application developer, who uses BPMN [8] with some additions based on the extension points defined in the standard (without touching the BPMN meta model), designed to model the salient characteristics of the WSN. The extended language is referred to as *BPMN4WSN*. This extended version comprises both new components and modeling rules.

A BPMN4WSN process must be composed of at least two pools: an *intra-WSN* pool and *WSN-aware* pool; Figure 4 contains a minimal example. The intra-WSN pool is the part where the WSN logic is specified, while the WSN-aware pool is a classical BPMN process. The splitting into process logic executed inside and outside the WSN forces the modeler to explicitly model interactions between the two parts as messages, directly mapping the run-time behavior (where messages are the only way of interaction between the parts) to the model. This separation also enables the clean generation of code.

In the intra-WSN pool, constructs that directly orchestrate WSN functionality (made available through high-level abstractions) are needed. This need is

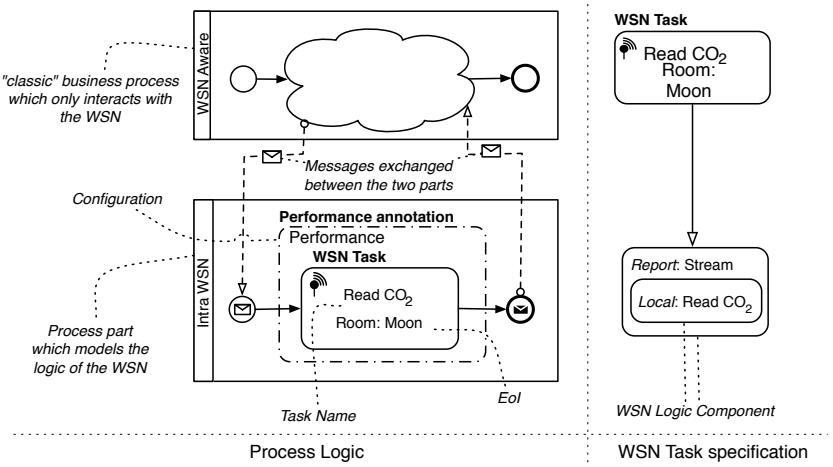

Figure 4. WSN-specific modeling constructs in BPMN4WSN.

addressed by introducing a new activity type: the *WSN Task*, that can only be used in the intra-WSN part. It has two properties: a reference to a set of *WSN logic construct* definitions and the *EoI* to which the respective operations should be applied (see Figures 4 and 2). It has an antenna on the top-left corner to distinguish it from other tasks; if specified, the *EoI* value is written below the task name. For example, setting the EoI value to "room Moon" will execute the task on those nodes that belong to the "room Moon". In a nutshell it specifies *where* (i.e., by which subset of nodes) each WSN Task is executed.

The referenced *WSN logic construct* definition is the set of operations that have to be performed by the network. For simplifying the modeling of such low-level programming specification, a set of *WSN logic constructs* that describe the common operations and the way they can be combined is created.

In addition to the WSN Task, a *Performance Annotation* element, i.e., an extension of the BPMN *group* element which shows the chosen performance configuration on the top-left corner, is introduced. It is used for describing the network behavior from a performance point of view. This new component allows the application developer and system developer to decide when the network performance goal has to be changed (e.g., to optimize battery lifetime). For example, when a room is empty, the network will be set to *low energy consumption* mode in order to save battery and prolong node network lifetime at the cost of lower reactiveness and possibly less reliable message transfer. In cases where high performance is needed (at the cost of battery power), other performance annotations are used. At run-time the execution semantics of these annotations is that one performance mode is set for the whole WSN, depending on the number of the tasks in each performance group. The group that contains the most tasks to be executed sets the performance mode.

5.2 WSN Task Specification

WSN Tasks are modeled in two steps: (i) the *process design* and (ii) the *process refinement*. The process design is generally carried out by the application developer. He just specifies a *WSN Tasks* with a speaking name, which can be a *sense*, *actuate*, or *aggregate* operation and the *EoI* on which the operation has to be executed. This part of the modeling is represented in Figure 2 by the items inside the *BPMN4WSN* dashed rectangle.

The process refinement (an example is shown in Figure 4), instead, is generally performed by the system developer. Its goal is to transform all high-level WSN Tasks into executable operations by combining *WSN logic constructs* which model the network behaviors. As shown in Figure 2, each WSN Task represents *WSN logic constructs* that are the basic functionality and instances of so called meta abstractions [4] that must be configured and instantiated:

Local actions are executed locally on each sensor node.

- The *tell/report actions* represent one-to-many/many-to-one communication.
- The *tell action* enables a node to delegate an embedded action to a set of other nodes.
- The *report action* enables the gathering of information from many nodes.
- *Collective actions* enable distributed, many-to-many collaborations.

Each of these distributed actions has a *target*, which is used to select the subset of nodes the action refers to (obtained by resolving logical EoIs into physical nodes, based on the system description). In addition there is also the possibility to specify *data operators* useful to perform mathematical operations during transmission of data (e.g., to compute the average).

Each specific WSN deployment has its unique system-description, which is the starting point for modeling. It describes the details of the network and it is used as configuration for the model editor. The document provides a high-level description of application-specific details of the concrete WSN deployment to the business process editor and to the model compiler. It is used by the editor to list only those attributes to the system developer that are actually available in a concrete deployment, such as the list of EoIs (simple or composed ones like "First Floor" comprising "room1" and "room2") and to restrict the selectable operations (e.g., CO_2 sensing can only be selected if EoI "room2" has been selected, because only that room is equipped with CO_2 sensors).

6 Prototype

The approach described in the previous sections has been implemented as a proof-of-concept prototype. Figure 3 depicts the architecture of the prototype, showing the document flow and the actors involved. The modeling process, defined by our tool chain, is divided into three phases: *modeling*, *translation*, and *execution*.

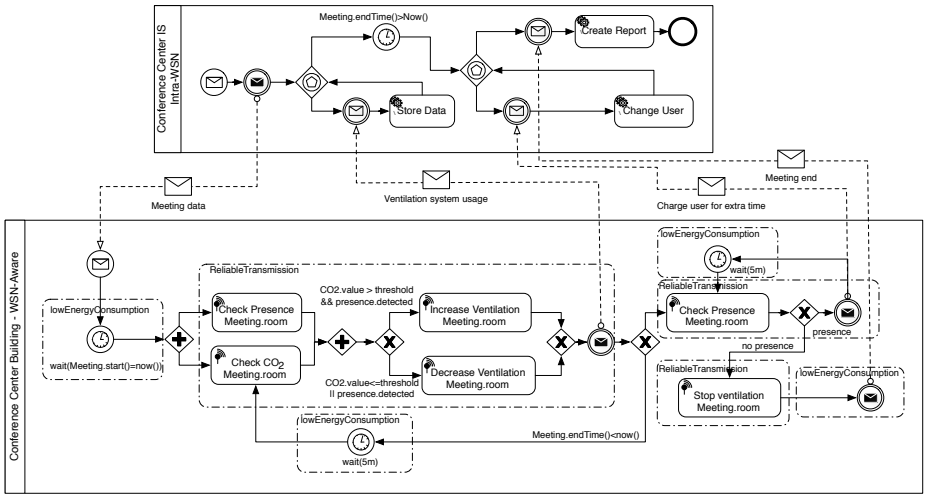

Figure 5. The HVAC process of the convention center. For high-res screen shots, visit `http://goo.gl/4Roc2`; last check 23 March 2012.

Modeling. For the modeling part of the prototype, a well-known web-based BPMN editor called *Signavio Core Components* (`http://code.google.com/p/signavio-core-components`) has been extended. The editor has been modified by adding a start page for scenario selection and a model editor for the *WSN logic constructs*.

The start page is used to select or create a separate workspace for each scenario to enable development for distinct WSN set-ups, each with its on system-description. In each workspace, only operations that can actually be executed inside the corresponding network are enabled, helping the modeler in creating correct executable processes. For instance, in our example scenario there would be the possibility to sense CO_2 and presence but no other environmental parameters as the WSN is only equipped with these sensors.

BPMN extension points have been used to realize WSN Tasks and performance annotations as explained in Section 5 and in Figure 4. To support the modeling, the modeling tool has been extended with these two components.

The *WSN logic construct* composition has been enabled by creating a new meta model inside the tool. By doing so, the modeler is given the possibility to compose *WSN logic construct* blocks by dragging and dropping and nesting them according to predefined composition rules that are checked by the tool. The composition is later translated into an internal format, and the files are used by the compiler to create the binary code for sensors.

Example. In Figure 5 there is a screen shot of the process that models the scenario explained in Section 2. For the sake of clarity, in the intra-WSN process only CO_2 measurement and presence detection are modeled.

A new process instance is started when a new meeting is scheduled. The WSN will be set to *low energy consumption mode* until the actual meeting starts. Throughout the duration of the meeting, the network checks the room conditions, increasing the ventilation when sensor values exceed a given threshold and a human presence is detected. After the scheduled meeting end time, the network checks if someone is still in the room, in which case the information system is informed, charging the user for *extra time*.

Translation and Execution. The WSN-aware part of the process is a standard BPMN model that can be executed by a process engine exterior to the WSN. The intra-WSN part, instead, is translated into executable code. A tool for translation called *model compiler* takes this part of the process and generates code implementing a custom execution engine. The executable program hence behaves similar to a regular BPMN engine interpreting the given BPMN model. The generated program implements a finite state machine, realizing the execution semantics of the translated process model including instance management and message correlation, and of course keeps track of all execution tokens in each process instance as specified in the BPMN 2.0 specification.

For example, an exclusive diverging gateway will be translated into a series of if statements (mapping the conditions on the outgoing flows) in the "main loop" of the program. Each WSN Task is translated using the *WSN logic construct* composition describing sensor logic. This is the most extensive generation step, as these sub-models need to be mapped to an API for instantiating, managing, and using those programming abstractions. The system-description describes the characteristic of each node of the network and it is used as input for the translator. The EoI of a WSN Task is mapped to attribute matching at run-time, e.g. if a WSN Task has been configured to operate on EoI "Floor 1" and the system developer contains information which room ids belong to that floor, this could be mapped to the expression `location='room1.1' or location='room1.2'`.

The two parts of the process can now be executed separately. To make them communicate, the model compiler maps the message flows between intra-WSN and WSN-aware process to communication endpoints that are created automatically on either side, enabling each part to receive and send messages. As the message format and transmission encoding are out of scope of the BPMN specification, a simple message format and an efficient transmission encoding are defined and implemented in both the generated intra-WSN executable and as an extension to a regular BPMN execution engine. In order to support the coordination of multiple instances, each message contains a field that is used for instance correlation and the execution of message start events creates instance IDs that need to be used by either side of a same process instance.

7 Discussion and Future Work

Our approach was guided by the core requirement presented in Section 4, i.e., to integrate WSN programming into business process modeling. We address

this requirement by offering unified modeling in one model editor, hiding model artifacts that are not relevant in a given modeling context, splitting work between application developer and system developer, and providing model compilation and execution as a custom engine in the WSN.

The work described in this paper is a first iteration towards integrating WSNs with BPs, combining classical business process modeling with ad-hoc extensions for WSNs that hide low-level network details. This integration allows an application developer to design process logic both inside and outside the sensor network, without requiring intimate knowledge of how to program distributed computations inside a WSN; an intuitive understanding of EoIs and sensing and actuating actions is enough. The system developer instead only focuses on the refinement of WSN Tasks. The described tool-chain takes care of splitting the two logics (intra-WSN and WSN-aware) and of the binary code generation. Endpoints for communication between the business process and the network are created following the model of the process. Yet, there is still space for improvement, space that we are going to cover in our future work.

Patterns. For instance, WSNs are characterized by hardware constraints that an application developer is typically not familiar with. Thus, providing patterns as best practices for modeling WSN logics is a necessity that is not yet implemented but that will be covered in the future, in order to further simplify the creation of WSN applications.

Two different meta models. Our current solution uses a different meta-model for modeling the lowest level of WSN logic, the *WSN logic construct.* WSN Tasks internally follow a logic that is different from BPMN, in order to compose WSN logic constructs. We could therefore not model them via simple sub-processes. A new logic and meta-model is needed, which is however seamlessly integrated in the same model editor. Opting for pure BPMN modeling, also for the *WSN logic construct* would have complicated the modeling as they have strict composition rules that could not be expressed easily in BPMN. Instead, we use a simple box model to compose *WSN logic constructs,* supporting the construction of correct models. We will evaluate to what degree we can use more BPMN and less custom modeling constructs in the future.

Modeling two separate pools. Our current prototype forces the modeler to model at least two pools, one for the intra-WSN part and one for the WSN-aware part. We opted for this solution as it makes communication from and to the WSN explicit and the compilation and creation of communication endpoints easier. Modeling all logic in a single pool would be less burdensome to the modeler, but it requires non-trivial data and control analyses of the process, which for the sake of simplicity we did not implement yet.

Events. Although the current model does not explicitly use special BPMN events for the WSN inside the process model, the approach is strongly based on asynchronous communication under the hood. On the one hand, modeling two explicit pools with asynchronous message exchanges has similar semantics as BPMN message events. On the other hand, for example, the current implementation of receiving a sensor value is realized asynchronously: rather than

querying the sensor directly, the buffered last value of a stream of sensor values is read. This behavior is however hidden behind the WSN logic constructs. Introducing WSN-specific events in the process model would however allow for more flexible control flow logic and is therefore part of the future development of the system.

Multiple processes interacting with one WSN. In our current prototype, the generated intra-WSN logic only supports conversations with multiple instances of one process model. In the future, we intend to support multiple process models by merging the intra-WSN parts of all models to a combined model or by running several of the generated engines concurrently at operating system level and dispatching messages based on model identifiers. In practice, we could single out the WSN-internal logic as an own pool and allow the application developer to define multiple (WSN-aware) processes interacting with this WSN-internal process. This would enable the generation of WSN code that natively supports multiple different WSN-aware processes.

8 Related work

Building commercially relevant applications on resource-constrained, networked embedded systems (the front-end) such as WSNs while integrating them into business processes of an enterprise (the back-end) is a complex, challenging task that has to be repeated for each combination of front-end and back-end. Numerous efforts have been made, aiming also at demonstrating the business benefit.

Approaching the problem bottom-up, i.e., from the WSNs, several solutions have been proposed to simplify programming. Although many programming abstractions have been introduced, most of them aim at simplifying the activities of skilled WSN programmers [9] and cannot be used directly to specify high-level process constructs by domain experts without WSN expertise.

The COBIS project (`www.cobis-online.de`) aimed at integrating heterogeneous WSNs with back-end systems by providing a web service facade to the WSN's functionality. The proof of concept was trialled in an environment, health, and safety application scenario, more specifically by enforcing physical storage rules for hazardous goods managed in an enterprise system [5,10].

The SOCRADES project (`www.socrades.eu`) targeted industrial automation with the goal to almost eliminate the need for any proprietary intermediate layers between embedded services and the business back-end by directly service-enabling devices themselves [1]. The approach was based on the WS-* family of web service standards and only for very resource-constrained and legacy devices a gateway/service-mediator concept was developed to enable those to participate in service orchestrations.

Other proposed solutions for modeling sensor network applications using a process-based design include the Graphic Workflow Execution Language for Sensor Network (GWELS) [6], which enables the design of data-flow as workflow, and an ad-hoc architecture for handling the communication. Similarly, [7] uses a process paradigm for defining WSN applications, easing the configuration for

non-experts of the field. Mash-up composition is also promising; in [11], the authors wrap smart-objects with web services, introducing an architecture and a web-based mash-up tool for composition and execution. These solutions enable the modeling of WSN logic in a model-driven fashion but without deriving the executable logic of the network.

Recently, BPMN has gained interest as method to *program* WSNs. Caracas et al. [12,13] presented studies on the expressiveness of the language and its potential to be compiled into source code for WSN nodes. As results they produce a system that creates WSN applications by compiling BPMN processes. The outcomes highlight that, as it is, BPMN is powerful enough for specifying the high-level behavior (if modeled with correct patterns) more than low-level one. At the same time they prove how a process can be compiled into native source code for WSNs, without losing too much performance compared to hand-written code. These preliminary works show the possibility to compile the BPMN for creating binary code. However, the example shown in this work users a higher-level API, that does not allow one to fine-tune communication in the WSN as it is possible with our approach.

In the past months, extensions of BPMN for modeling smart objects have been proposed as outcome of the IoT-A (`www.iot-a.eu`) project [14,15], an idea that shares some common ground with our approach. The idea is to extend the BPMN language to model Internet of Things (IoT) aspects. However, this approach differs as they propose modeling extensions that affect the language at a high level of abstraction; in fact their goal is to use this language to model IoT services instead of creating the logic from the process.

Approaches like SysML [16] are only remotely related to our approach. This modeling framework, derived from UML, allows the modeling of low-level details of a WSN system. Yet, SysML models are graphical models without a standard serialization, therefore they are not directly usable for process-based integration.

9 Conclusion

In the era of the IoT, collaboration and integration of non-conventional IT devices, such as entertainment and automotive equipment, RFID devices and tags, or WSNs, with Enterprise services is of paramount importance [1]. In this paper, we focused on one relevant representative of this need, i.e., WSNs, which typically still represent isolated and impenetrable realities from a business IT point of view. We proposed a layered approach for developing, deploying and managing WSN applications that natively interact with enterprise information systems, such as a business process engine and the processes running therein. We did not try to crack the whole problem at once, e.g., by aiming at a business-view-only approach to WSN application development, and rather foster current practice, equipping both the application developer (holding the process knowledge) and the system developer (holding the WSN knowledge) with effective languages and instruments to co-develop advanced, process-based WSN applications with non-trivial distributed sensing and actuation logics.

Acknowledgments. This work was supported by the European Commission funded project make*Sense* (www.project-makesense.eu).

References

1. Karnouskos, S., Savio, D., Spiess, P., Guinard, D., Trifa, V., Baecker, O.: Real world service interaction with enterprise systems in dynamic manufacturing environments. In: Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management. Springer (2010)
2. OMG: Business Process Model and Notation (BPMN), Version 2.0. http://www.omg.org/spec/BPMN/2.0 (January 2011)
3. Mottola, L., Picco, G.: Programming wireless sensor networks: Fundamental concepts and state of the art. ACM Computing Surveys (CSUR) **43**(3) (2011) 19
4. Casati, F., Daniel, F., Dantchev, G., Eriksson, J., Finne, N., Karnouskos, S., Montero, P.M., Mottola, L., Oppermann, F., Picco, G., Quartulli, A., Römer, K., Spiess, P., Tranquillini, S., Voigt, T.: Towards business processes orchestrating the physical enterprise with wireless sensor networks. In: ICSE 2012. (June 2012)
5. Spiess, P., Vogt, H., Jutting, H.: Integrating sensor networks with business processes. In: Real-World Sensor Networks Workshop at ACM MobiSys. (2006)
6. Glombitza, N., Lipphardt, M., Werner, C., Fischer, S.: Using graphical process modeling for realizing soa programming paradigms in sensor networks. In: WONS 2009. (2009) 61 –70
7. Amundson, I., Kushwaha, M., Koutsoukos, X., Neema, S., Sztipanovits, J.: Efficient integration of web services in ambient-aware sensor network applications. In: BaseNets 2006. (October 2006)
8. White, S.: Introduction to BPMN. IBM Cooperation (2004)
9. Mottola, L., Picco, G.P.: Programming wireless sensor networks: Fundamental concepts and state of the art. Technical report (2008)
10. Spiess, P., Karnouskos, S.: Maximizing the business value of networked embedded systems through process-level integration into enterprise software. In: ICPCA 2007. (July 2007) 536–541
11. Guinard, D., Trifa, V., Wilde, E.: Architecting a mashable open world wide web of things. Technical Report 663, Institute for Pervasive Computing, ETH Zurich (2010)
12. Caracas, A., Kramp, T.: On the expressiveness of bpmn for modeling wireless sensor networks applications. In: 3rd internation workshop on BPMN. (2011)
13. Caracas, A., Bernauer, A.: Compiling business process models for sensor networks. In: DCOSS, IEEE (2011) 1–8
14. Sperner, K., Meyer, S., Magerkurth, C.: Introducing entity-based concepts to business process modeling. In: 3rd International Workshop and Practitioner Day on BPMN. (2011)
15. Meyer, S., Sperner, K., Magerkurth, C., Pasquier, J.: Towards modeling real-world aware business processes. In: Proceedings of the Second International Workshop on Web of Things. WoT '11, New York, NY, USA, ACM (2011) 8:1–8:6
16. Weilkiens, T.: Systems engineering with SysML/UML: modeling, analysis, design. Morgan Kaufmann (2007)