Institute of Architecture of Application Systems

Extending BPMN for Wireless Sensor Networks

C. Timurhan Sungur∗†, Patrik Spiess∗, Nina Oertel∗, and Oliver Kopp†
∗SAP Research, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe Germany
†IAAS, University of Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany
Email: {lastname}@iaas.uni-stuttgart.de, {patrik.spiess, nina.oertel}@sap.com

BibTeX:

```
@inproceedings{BPMN4WSN,
  author    = {C. Timurhan Sungar and Patrik Spiess and
               Nina Oertel and Oliver Kopp},
  title     = {Extending BPMN for Wireless Sensor Networks},
  booktitle = {2013 IEEE International Conference on Business Informatics},
  year      = {2013},
  pages     = {109--116},
  doi       = {10.1109/CBI.2013.24},
  publisher = {IEEE Computer Society}
}
```

© 2013 IEEE Computer Society. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Universität Stuttgart
Germany

Extending BPMN for Wireless Sensor Networks

C. Timurhan Sungur*†, Patrik Spiess*, Nina Oertel*, and Oliver Kopp†
*SAP Research, Vincenz-Priessnitz-Str. 1, 76131 Karlsruhe Germany
†IAAS, University of Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany
Email: {lastname}@iaas.uni-stuttgart.de, {patrik.spiess, nina.oertel}@sap.com

***Abstract*—Wireless sensor/actuator networks (WSNs) are hard to program, in particular so for business domain experts that have a good understanding of how WSNs can best be used to improve business operations. This contributes to hampering WSN adoption by enterprises. As business process modeling languages such as the Business Process Model and Notation (BPMN) are well accessible to domain experts, they can be used as a tool to facilitate WSN programming. In this paper, we explore the properties of WSNs that set them apart from traditional IT systems and use these properties to derive requirements for BPMN extensions that are tailored to the specifics of WSNs. We furthermore propose a set of BPMN extensions that fulfill these requirements and demonstrate that they are better suited for modeling WSN processes than standard BPMN.**

I. Introduction

WSNs are dynamic, ad-hoc networks of tiny computers: the sensor nodes. They have limited capabilities and typically perform sensing and actuation tasks [1], [2]. For example, a node in a WSN might measure temperature values in a room while another node controls the air conditioning according to the sensed values and desired overall room temperature. WSN applications range from environmental monitoring, e.g., wild fire spotting or monitoring volcanic activity, to applications running in an enterprise context [2]–[5], such as monitoring and optimizing energy consumption of buildings or enabling predictive maintenance of assets.

Application logic for WSNs is notoriously difficult to program, often requiring low level programming skills due to the requirements of the restricted node hardware [6]. Although the initial setup of a network is cumbersome and costly, modifying the tasks performed by an already running network requires additional effort. Additionally, it is often domain experts (such as facility managers) that have a thorough understanding of the processes in companies that could be improved by WSN deployments. These experts have the knowledge how to make the best possible use of sensing and actuating equipment, but there is currently no easy possibility to make the networks accessible to this type of user. All these reasons combined slow and sometimes hinder the adoption of WSNs in a broader context.

The `make`*Sense* project has ventured out to bridge the gap between the technical expertise needed to program sensor networks and the domain expertise required to design useful processes. The goal is to facilitate the programming of WSNs by employing graphical business process modeling approaches, such as offered by BPMN, to design WSN applications. These process modeling languages are in widespread use and easy to grasp for domain experts.

The work described in this paper is embedded in a larger framework for model-driven development of logic for WSNs. Figure 1 shows the overall process. Before the described modeling takes place, a technical person chooses the required sensors and actuators and orders and installs them. They come with basic software that allows them to form a mesh network and exposes their remote sensing and actuation function. A discovery mechanism scans the network and creates a system model that is used to perform the actual modeling as described in this paper. After this modeling step, the process model is analyzed and those parts that obtain and aggregate sensor values or drive actuators are used to generate and compile executable code that is installed directly on the sensor nodes. The execution of the generated code directly on sensor nodes is more efficient than interpreting the process part in a process execution engine. Due to their constrained resources, the nodes would not allow offering a complete execution engine. At runtime, the compiled part of the process running on the WSN communicates with the rest of the process via messaging.

Although WSNs are IT systems, they still have properties that make them stand out from systems typically addressed in business process modeling. For example, task execution in a WSN is usually distributed on subsets of nodes and the set of operations a WSN is able to carry out is typically severely restricted compared to multipurpose IT systems. In general, it is possible to model WSN processes with standard BPMN [7],

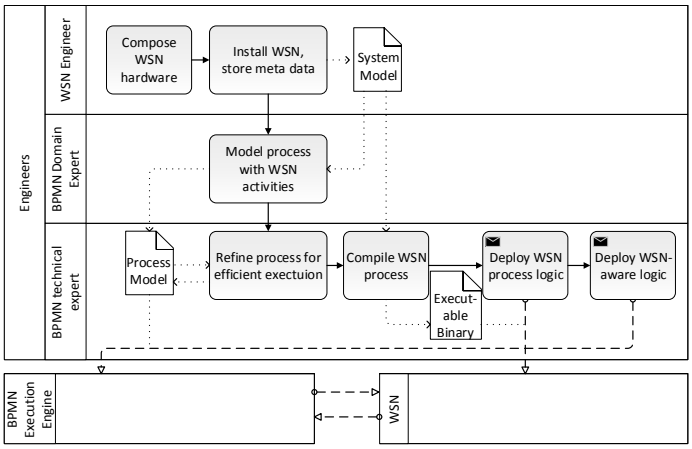

Fig. 1: `make`*Sense* Development Process

[8]. Due to the specific characteristics of WSNs, however, this might require modeling details for execution, which is not the case in our modeling approach that is tailored to the specifics of WSNs.

In this paper, our goal is therefore to propose a set of extensions for BPMN 2.0 [9] that are effective for modeling WSN processes. These extensions have been devised based on an analysis of the specific properties that distinguish WSNs from other IT systems. These properties and the requirements towards BPMN extensions motivated by them are presented in Section II and Section III respectively. The proposed modeling extensions will be described in Section IV. Section V outlines the implementation. This will be followed in Section VI by a survey of related work and a comparison of the proposed extensions to other approaches. Section VII compares the approach with modeling WSN processes using standard BPMN. Finally, Section VIII concludes and provides an outlook on future work.

II. Background on WSNs and BPMN

In Subsection II-A we analyze properties of WSNs relevant for application modeling. This will guide our requirements for WSN-specific BPMN in Section III and our solution proposals in Section IV. Subsection II-B provides a short overview on BPMN and describes its current capabilities to capture WSNs.

A. WSN Properties

WSNs consist of nodes, which are connected via digital radio links. The network itself is not fixed, but ad-hoc, as sensors may join or disconnect. Thus, WSNs are ad-hoc networks with specific properties [5]. In general, these properties make it challenging to model WSNs and to create corresponding applications. The properties of WSNs also motivate the need for extending BPMN to better reflect these properties in the process models. In the following paragraphs, we will introduce and explain the properties that are ultimately relevant for creating WSN specific BPMN models.

Dynamic Addition and Removal of Wireless Sensor Nodes (P1) In WSNs, to increase sensing accuracy or to increase the coverage area of the WSN, new nodes can be added even after the application has been started. Moreover, nodes might turn off as a result of exhausting their limited energy resources [2], [5]. This is why, for WSNs, direct addressing, i.e., addressing single nodes by unique identifiers (e.g., MAC address), is in general not beneficial. An indirect addressing scheme can solve this problem [10]. For indirect addressing, general common properties of the nodes are used to address them instead of their uniquely identifying properties, e.g., "sensors" or "temperature sensors". In the introduced model-driven tool chain, addresses can be resolved during compilation (static resolution) time of the models or during run-time of the application (dynamic resolution).

Categories of WSN Operations (P2) Regarding WSN operations, one can distinguish between local and command actions. A local action is an operation that is actually executed at a specific node on which it is invoked. A command action involves sending a message to remote nodes, triggering them to perform an operation. Command actions enable a more dynamic environment by allowing execution of operations remotely based on run-time decisions, e.g., after a local sense action, based on the results, nodes might be commanded to actuate. Both the command and local actions can be type of a sense, an actuate or an intermediary operation [11]. The sense operation is used to analyze surrounding environment, e.g., sensing pressure. Usually, this operation creates some output data. The actuate operation is used to manipulate surrounding environment, e.g., increase ventilation. The intermediary operation is used to send/receive data from outer world and to make computation in WSNs, e.g., decide whether the temperature is above the threshold.

Both local and command actions may create output values, which are consumed by other nodes [1]. Especially, sense actions create data, which is consumed by other nodes later on for some application specific purposes. Sensing presence data (of people in a space) would be composed of two steps: sensing presence and aggregating it to predict presence more precisely. Successively sensing and using this created data can be called as combined operations, i.e., execution of the one operation depends on the other one.

Parallel Execution of the Same Process Logic in one Application (P3) Some WSN applications can execute the same process logic at different regions of the WSN in one application [10]. A simple example is an air conditioning system of multiple meeting rooms that is operated with a WSN. In this example, there might be multiple meeting rooms running different instances of the same process.

Distributed Nature of WSN Applications (P4) To reduce the power consumption and to remove a central point of failure, WSN applications are executed in a distributed fashion, i.e., by not making computations on a single powerful node. The execution logic is distributed on multiple powerful nodes. Moreover, by having multiple nodes executing operations instead of a single powerful node, the response time of WSNs to events and power consumption of nodes decrease. This makes WSNs more suitable for real-time applications and prolongs the lifetime of the WSN application [1].

Limited Resources and Error-prone Behavior of WSN Nodes (P5) WSN sensor nodes typically have limited resources, among which battery plays a critical role. These limited resources and their deployment environment make nodes prone to failures [5]. There is a strong correlation between power-consumption and amount of communication in WSNs [5]. The degree of power consumption would affect the non-functional properties of the respective WSN application such as response time, total running time of the WSN, etc.

Event-driven Nature of WSNs (P6) Behavior of event-driven operations is quite different from periodic operations because they are halted until an event triggers them whereas periodic operations are executed in certain time intervals and return the result as soon as they are called [7], [10]. During the execution, event-driven operations communicate less than

periodic operations because they go into an idle mode until the specified event happens. Consequently, event-driven operations consume less energy compared to periodic operations [7], [10], because the communication is the most expensive operation in WSNs [5]. An example of event-driven operations in a WSN can be a set of nodes that check if a door is open or not. If the observed door has been opened, they communicate to inform the data sinks, i.e., the nodes or external systems interested in the sensed data. In case of periodic action, the sensors would not wait for event to happen but they would inform the sinks periodically about the status of the door.

B. BPMN

Economic changes and technology advancements force companies to keep their business processes updated and competitive. Business process management (BPM) is a way to drive a company by processes. BPM aims to document and improve processes, and to support them by IT [12]. BPMN is the de-facto standard for modeling processes and to bring them to execution. Note that internally, a workflow engine can use another language to execute BPMN [13]. BPMN is a process modeling and execution language and a well-accepted industry standard. It bridges both the gap in understanding between business experts and developers as well as among business experts, so that a common understanding of the process is achieved [9].

Although standard BPMN can be used to model WSN processes [7], [8], that approach results in a non optimal process models and some details are sacrificed. The main reason is the limited expressiveness of BPMN with regards to characteristics specific to WSNs. An example of a WSN application modeled with standard BPMN is shown in Fig. 2. The scenario is ventilation of a meeting room. The process is started by a message stating the room number, the reservation period and the desired maximum CO_2 value. The process then waits until the meeting begins. It then continuously requests the current CO_2 values of the meeting room and adjusts the ventilation accordingly. The sensing and the ventilation actuators are represented as separate pools as they are different groups of nodes.

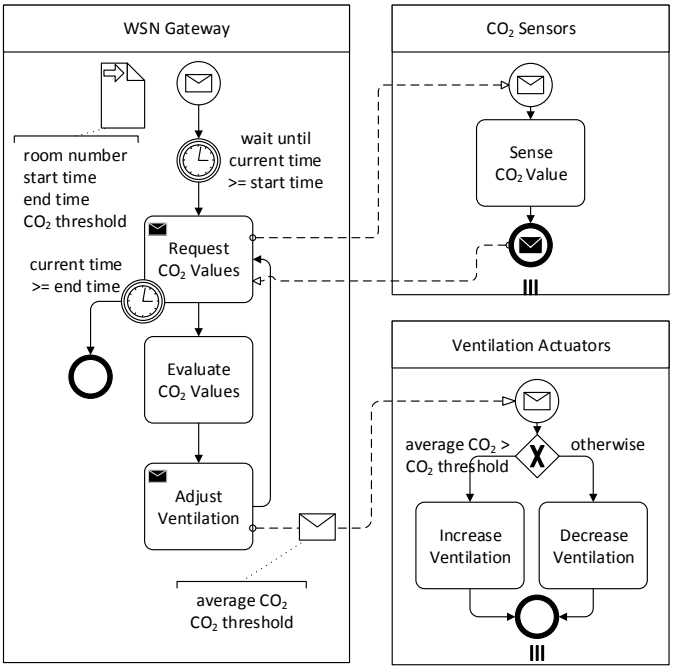

Fig. 2: A ventilation business process with standard BPMN

III. Requirements for WSN-specific BPMN

The properties of WSNs are critical to be correctly modeled and to be used as an execution environment. While reflecting the relevant properties to the user during modeling, we also need to conform to standard BPMN. In the following sections, we state general requirements on BPMN modeling constructs which are specific to WSNs. These requirements are derived from the properties stated in Section II and have been harvested during our work in the `make`*Sense* project.

Support for Indirect and Dynamic Addressing of Nodes (R1) For the purpose of supporting a dynamically changing set of nodes (P1), we need a means to address nodes indirectly. In case of direct addressing, a re-initialization might be required to add a new node to the system. Moreover, indirect addressing provides a method of grouping nodes, e.g., sensors, actuators, temperature sensors, etc. By this way, one can give common attributes to nodes and later on use them to address these nodes. Indirect addressing would provide programmers a type of reference autonomy, i.e., it would avoid coupling between model and individual nodes. Different resolution schemes (static vs. dynamic resolution) in the model-driven tool chain demand their necessary differentiation during modeling (P1).

Support and Restrict User to WSN Operation Categories (R2) Sense, actuate, and intermediary operations (P2) behave differently during execution. They represent the type of the WSN action (actionType). Local or command actions define if the operation is locally executed or remotely executed (isCommandAction). This difference in execution logic demands necessary differentiation between local and command actions at the modeling level. The executed operation (targetOperation) on some nodes (actionPerformer) might create some output data and this data might be returned to another group of nodes (outputTarget). Returned data might be an input data for another operation (returnOperation). The nodes have to be specified following the result of R1 and thus allow for a resolution to multiple target nodes. In summary, we need a 6-tuple to define a WSN operation comprehensively, i.e., actionType, isCommandAction, actionPerformer, outputTarget, targetOperation, returnOperation.

Support for Multiple Instances of the Same Process (R3) From an execution point of view, there is a significant difference between execution of a WSN business process and a conventional business process: WSN processes are usually executed inside of a WSN itself, whereas standard BPMN processes are executed inside of a business process engine. Moreover, similar to standard business processes, in a WSN multiple instances

of the same process can exist concurrently (P3). Therefore, explicit separation of WSN processes from standard business processes and support for the visual representation of parallel processes are needed.

Distribution of Execution Logic into WSN (R4)

The produced data and events in WSNs can be used to make decisions in a business process. This decision mechanism is executed seamlessly in business process execution engines in case of conventional business processes unlike the processes executed in a WSN. As the modeled processes might be completely executed in a WSN, we might need a means to explicitly define information about which nodes are expected to orchestrate the executing WSN process. These orchestrating nodes would be responsible for holding information about the current state of the process instance, coordination operations executed in the process instance, etc. The execution of a business process might be orchestrated by multiple nodes in parallel (P4).

Prioritization of Performance Goals (R5) In WSNs, there may be application specific conflicting performance goals, e.g., shorter response time, reliable communication, lower power consumption (P5). To achieve more reactive behavior, preserve resources and in general adapt application specific needs, there is a need to define these non-functional properties during BPMN modeling. For instance, in case of a fire detection scenario, reliable message transmission will be needed and response time needs to be short.

Support for Event-driven Actions in Modeling (R6) WSNs provide both event-driven and periodic operations, the behavior of event-driven WSN actions (P6) is different from behavior of periodic actions. Representation of different behaviors with the same visual construct would decrease the understandability of the BPMN process models [14]. For the purpose of avoiding this reduction in understandability, we need a means to provide necessary distinguishability between an event-driven task and a periodic task.

Models should be stable on minor WSN changes (R7) The changes in the deployment of a specific network on which the process runs such as its topology, changing the orchestrating nodes, changing the implementation details of a WSN task, etc. should not affect the model itself.

IV. Solution Proposals

To satisfy the requirements presented in Section III, we propose extensions to standard BPMN. These extensions are WSN Task, WSN Pool, and Performance Annotations. We call the set of all extensions "BPMN4WSN". Our proposals are built on the work by Tranquillini et al. [11]. Although this work is effective, it relies on a separate non-BPMN diagram type: Each WSN task refers to an instance of that diagram type. In this work, we make the approach more compatible to standard BPMN and manage to include all extended information extension elements at the designated places in the BPMN schema and reuse data types of the BPMN standard. In addition, we integrate all visual elements in a standard BPMN diagram.

We define a WSN Task as an extension of a standard BPMN Service Task, a WSN Pool is an annotated standard BPMN pool and a Performance Annotation is an extension of BPMN groups. In the following sections, we define these extensions in detail.

A. WSN Task

A WSN Task corresponds to a task executed in a WSN process. The visual representation of a WSN Task is the same with a standard BPMN task except its additional icons.

A class diagram of a WSN Task is presented in Fig. 3. The instances based on tWSNPerformer are used to address nodes dynamically and indirectly which satisfies the requirement R1. The orchestrationPerformer is used to change the orchestrating nodes of a WSN and it satisfies the requirement R4. The actionType and isCommandAction elements are used to define type and location of the WSN Task respectively. The targetOperation and returnOperation are type of tWSNOperation and tWSNOperation is used to bind WSN operations with WSN Tasks. The targetOperation is the initial operation which is executed by nodes defined by actionPerformer elements. The outputTarget element stands for the set of nodes to which output data of the targetOperation is sent and nodes defined by outputTarget would execute the operation referenced by returnOperation. By introducing this 6-tuple, i.e., {actionType, isCommandAction, actionPerformer, outputTarget, targetOperation, returnOperation}, we meet requirement R2. The isEventDriven extension is used to mark a WSN Task as either periodic or event-driven and use of it satisfies the requirement R6. Extension element parameters, which is an instance of tParameters, fulfills the requirement R5 partially and with Performance Annotations (see Subsection IV-C), we fulfill corresponding requirement completely. In the following subsections, we explain the extension elements of a WSN Task in detail.

tWSNOperation In standard BPMN, the standard operation construct is of type tOperation; however, this construct does not suffice in case of WSN operations because a WSN operation might take additional parameters and the required inMessageRef element of tOperation is not applicable for WSN operations. The tWSNOperation type is used to bind a WSN operation to a WSN Task. To define the namespace for available operations, implementationRef of type tWSNOperation is used. Type of the implementationRef is a qualified name. In defined namespace, the operationName can be used to select a unique operation. The operations can take parameters which are defined by parameters extension element of the type tWSNOperation. The first operation whose input parameters are the same as the provided parameters extension element will be selected. The targetOperation and returnOperation elements are instances of tWSNOperation. The targetOperation is the initial operation executed and returnOperation is the operation based on the output data of the targetOperation. Parameters element of type tWSNOperation contains a list of parameters used to pass compile time configuration variables to corresponding

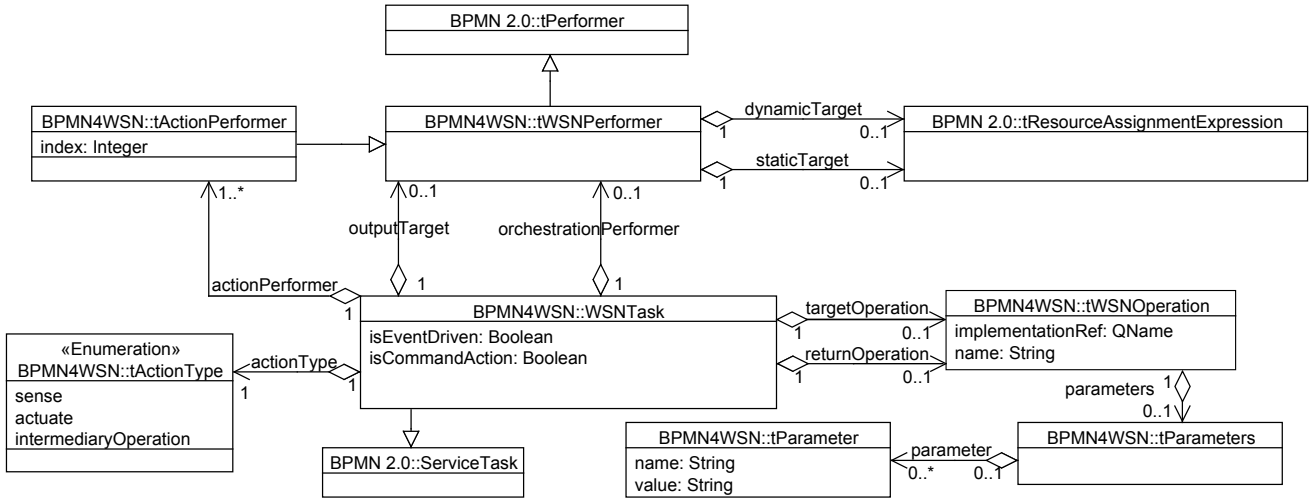

Fig. 3: WSN Task class diagram

operations. A parameter is composed of name and value pair. An example would be the interval at which sensing should occur: "period = 30" for sensing every 30 seconds.

actionType The actionType element is used to define a WSN operation as a sense (?), an actuate (!) or an intermediary operation (■) where sense is the default value. The icon of the selected type is filled with black (?□!, ?□!, ?■!).

isCommandAction The extension element isCommandAction is of type Boolean and its default value is false. Setting this element true would define this task as a command action. LocalAction is the default selection for a WSN Task. Whenever a WSN Task set as a command task an additional arrow icon is visible which represents a command message sent to the nodes which are responsible for the execution ( ).

tWSNPerformer In standard BPMN, tPerformer is the class defining the resource that will perform the activity. The tWSNPerformer extends tPerformer and used to associate WSN nodes with a WSN Task. Therefore, we add a dynamicTarget element and a staticTarget element to distinguish between dynamic and static resolution schemes. The dynamicTarget refers to an expression suited to find performer nodes during run-time of the WSN process. A staticTarget refers to an expression, which is evaluated and then used to address the WSN nodes the corresponding code should be deployed. Technically, both dynamicTarget and staticTarget are of type tResourceAssignmentExpression, which is the BPMN standard type for resource assignments. The outputTarget and orchestrationPerformer extension elements are instances of the type tWSNPerformer.

outputTarget represents the set of nodes to which the output data should be sent. If the returnOperation has been defined, the targeted nodes are expected to execute the referenced operation. The outputTarget is an optional element and in case it is defined, an additional return arrow is added to the WSN task visual element ( ).

orchestrationPerformer is an optional element used to change the orchestrating nodes of a WSN process instance. If a WSN task is defined with an orchestrationPerformer, the execution of the WSN task and the execution constructs on the outgoing sequence flows are orchestrated by the defined orchestration performers until a new definition occurs. Multiple orchestrationPerformers might exist as parallel execution flows are possible. In case of merging execution flows, the set of orchestrating nodes is the union of both execution flows. The coordination techniques of these orchestrating nodes are out of scope of this paper and a further reading can be found in [1].

tActionPerformer The tActionPerformer extends tWSNPerformer. It has an additional index element to give an order if multiple actionPerformer have been defined. The actionPerformer is an instance of tActionPerformer type.

actionPerformer represents the nodes that are supposed to execute the targetOperation. There can be more than one actionPerformer defined in a WSN Task. In case of local action, only the actionPerformer with the lowest index will be used. In case of a command action, all defined actionPerformers will be used. The index attribute of the each actionPerformer will be used to determine the order of execution. Orchestrating nodes initiate the first command message to the actionPerformer with the lowest index and it will be propagated to the last actionPerformer which is responsible for the execution of the targetOperation.

isEventDriven This is an element of Boolean value and used to mark a WSN Task event-driven. The default value is false, which means the task is a periodic task. To represent behavioral difference between these tasks at a visual level, we use a clock icon for event-driven tasks ( ).

B. WSN Pool

WSN Pools are used to define processes which are executed in WSNs. The WSN Pool extends the standard BPMN pool. To provide a visual differentiation between a standard BPMN pool and a WSN pool, we provide a WSN icon on the top left corner of the pools. By use of these WSN pools we provide a

clear separation between WSN processes and standard business processes; therefore, we satisfy requirement R3.

C. Performance Annotations

BPMN allows to associate activities with common properties to a group. We inherit from this grouping construct and add a performanceGoalRef, which references to a concrete performance goal. Performance goals define the performance priority of an application. Performance annotations group BPMN constructs and these constructs will be later transformed into executable code for execution. The performance annotations modify the execution characteristics of the contained BPMN constructs during their execution. In case of a fire detection scenario, the tasks executed after a fire has been detected would sacrifice energy resources and try to deliver messages reliably. Actual performance goal definitions are just referenced and definitions are implementation specific. WSN Task parameters are used to define task specific properties; whereas, performance annotations define performance goals of the whole WSN. By using performance annotations, we fulfill requirement R5 completely.

D. Extensions Applied

The mappings between solution proposals and requirements are shown in Table I. Fig. 2 presented a ventilation process. This process has been remodeled using the presented extensions and is presented in Fig. 4. In Fig. 4, the CO_2 calculation is accomplished through the WSN Task "Calculate CO_2 Average" whereas in Fig. 2 this calculation is distributed among multiple tasks: "Request CO_2 2 Value" and "Evaluate CO_2 Values" in the WSN Gateway pool and tasks in the "CO_2 Sensors" pool. In order to create an equivalent "Calculate CO_2 Average" WSN Task, one needs to define WSN extensions properly. That means, a command sense WSN task with targetOperation, returnOperation, outputTarget and actionPerformer should be defined. The defined returnOperation is an aggregation equivalent to the "Evaluate CO_2 Values" task in Fig. 2. By using outputTarget and actionPerformer, the need of extra pool disappeared. Similarly, there is no need of an additional "Adjust Ventilation" as in Fig. 2 due to use of WSN extension elements.

V. Implementation

In a proof-of-concept implementation, we realized a Java prototype that translates the WSN pool into executable code for the sensor network. Despite we do not demonstrate it in this paper, the WSN pool can connect to other pools and therefore easily integrate with other process participants (such as BPMN processes executing in standard engines or other systems). The compiler that translates the process model into executable code also generates the necessary communication endpoints and instance handling and correlation logic, allowing for both stand-alone and integrated operation of the WSN. An upcoming report on project `make`*Sense* will elaborate on the complete project scope.

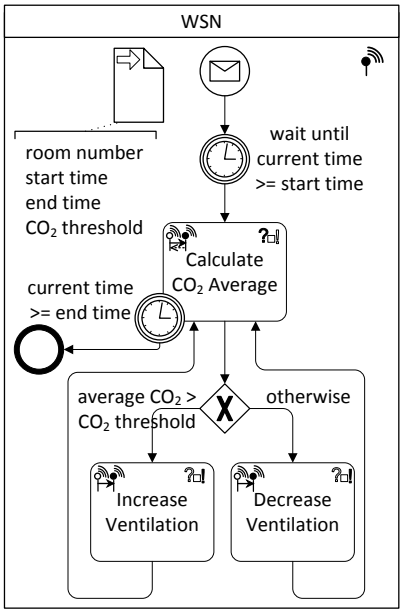

Fig. 4: The ventilation process with BPMN4WSN

VI. Related Work

Tailoring existing modeling languages towards specific needs is a common approach. For instance, there are at least 62 BPEL extensions including modeling and run-time extensions [15]. None of them tackles Wireless Sensor Networks. Zor et al. [16] propose a BPMN extension for the manufacturing domain to explicitly handle products and resources. The inclusion of security aspects, such as access control or intrusion detection, into BPMN is described by Rodriguez et al. [17] and by Bruckner et al. [18]. Management of application topologies is enabled by BPMN4TOSCA [19]. However, these BPMN extensions also do not address the specific requirements of WSNs. Discussions on business process transformations usually regard the business-IT-gap and the transformation of higher level processes into lower level, more technical, business processes. For instance, Stein et al. [20] survey on transformations to BPEL. WSNs, however, have not yet been in the focus of research or practice. Creation of WSN applications are challenging because during development, low-level programming is done. Application should be scalable and resources are constrained. To overcome these programming difficulties, different model-driven code generation techniques with different modeling languages were proposed [6], [8], [11], [21]. Only the work by Caracas and Kramp [8] and Tranquillini et al. [11] use BPMN to model and generate code from these models. The former one uses standard BPMN to create WSN models and the latter one proposes an extension to standard BPMN. This research is conducted under the `make`*Sense* project as the work of [11] and solution proposals are built on top of the previous work of [11].

TABLE I: Solution/Requirement Mapping

Solution Name	Requirement
WSNTask/tWSNPerformer, tActionPerformer	Support for Indirect and Dynamic Addressing of Nodes (R1)
WSNTask/actionType, targetOperation, returnOperation, isCommandAction, actionPerfomer, outputTarget	Support and Restrict User to WSN Operation Categories (R2)
WSN Pool	Support for Multiple Instances of the Same Process (R3)
WSNTask/orchestrationPerformer	Distribution of Execution Logic into WSN (R4)
WSNTask/tWSNOperation/parameters Performance Annotations	Prioritization of Performance Goals (R5)
WSNTask/isEventDriven	Support for Event-driven Actions in Modeling (R6)
All	Models should be stable on minor WSN changes (R7)

VII. Discussion

In this section, we compare benefits of using our extensions and drawbacks of standard BPMN.

Support for Indirect and Dynamic Addressing of Nodes (R1) With standard BPMN, we can address single WSN nodes, group of WSN nodes, or all nodes in WSN. This can be done by using standard BPMN pools where the names of the pools represent corresponding nodes. If more than one node is present, one should use a multiple instance pool. The data contained in the message construct of BPMN may be used to select the target group of nodes. However, this does not provide an explicit dynamic way of selecting WSN nodes which would be responsible for the execution of the corresponding tasks. Pool names can be used to address nodes statically as these names can be considered as attributes. However, this approach would have its draw-backs in case multiple attributes define a group of nodes. With BPMN4WSN, users can define expressions which can address nodes directly, indirectly, statically and dynamically using WSN specific performer elements.

Support and Restrict User to WSN Operation Categories (R2) Standard BPMN does not provide constructs for sense, actuate, and intermediary operations because it is tailored to the execution on general multi-purpose IT systems. Moreover, there is no means of distinguishing local and command actions. Although the initial operation and operations on the output data are combined, standard BPMN would represent it as two distinct tasks and this way of representing would break the combined nature of these operations. Moreover, the created output data would be available for other activities during the life of parent process with respect to the BPMN standard, which would require communication overhead, a waste of resources in such a resource constrained environment. On the other hand, by using the extension we proposed, one can save resources and moreover can increase the understandability of BPMN models because the different actions have different visual mappings which increase the clarity of models.

Support for Multiple Instances of the Same Process (R3) Pools are used to encapsulate processes in case of the existence of multiple participants in standard BPMN. However to model the processes executed on WSNs, pools are used to address nodes with common properties. The tasks defined in these pools are executed on the nodes addressed by the surrounding pool. The actual process executed on a WSN would be a combination of the pools which address WSN entities. This way of modeling would introduce another level of abstraction and would cause a semantic change. This semantic change can cause confusions and diminish common understanding of BPMN models. However, in a soon to be published user study, we could prove that with minimal training domain experts were enabled to understand the new execution semantic of WSN actions. In case of the remaining extensions, we preserve standard BPMN semantics with the WSN Pool and satisfy this requirement.

Distribution of Execution Logic into WSN (R4) In case of standard BPMN, modelers and developers do not have the ability of distributing the execution logic into the WSN, which would limit the executability of WSN models. In the BPMN4WSN, this is done by the orchestrationPerformer element of WSN Tasks.

Prioritization of Performance Goals (R5) Standard BPMN does not provide a construct to prioritize performance goals. Caracas et al. [8] proposed using the categories element of the BPMN message construct to define communication protocols between WSN processes. Different protocols might run with different performance goals, each performance goal will only affect the behavior of the communicating parties, not the behavior of the whole WSN application. In our approach, we give some static parameters to each task, to configure its behavior during compile time. Moreover, we can add some performance goals for the whole WSN during execution of the preselected execution blocks.

Support for Event-driven Actions in Modeling (R6) Standard BPMN does not provide a differentiation between event-driven and periodic tasks. Modeling WSNs with standard BPMN would create models that are less clear and less understandable than models with explicit event-driven task markings. The solution proposed by Caracas et al. [8] distinguishes between synchronous and asynchronous tasks by adding a textual annotation at the successful outgoing sequence flows taken at the successful completion of the task. This can improve visual separation, however, business experts and developers may want to annotate other outgoing sequence flows

for documentation purposes, which might lead to confusion. Additionally, these annotations would belong to the outgoing sequence flow elements, not to the task itself. The WSN tasks provide explicit event-driven markings, of which business experts and developers would have a common understanding.

Models should be stable on minor WSN changes (R7) In case of standard BPMN, dynamic topology changes would affect the models and might require addition or removal of new or existing pools. In contrast, for the models created using BPMN4WSN extensions, there is no need to remodel for minor changes in a WSN set-up. That means, if the WSN topology changes due to dynamic addition and removal of WSN nodes, the models would not need to be visually changed.

VIII. Conclusions and Outlook

This publication introduced an extension for BPMN in order to make it more suitable for creating application logic for scenarios that involve sensing and aggregating values in a physical environment and control of actuators with the help of WSNs. After describing the relevant properties of WSNs from a modeling perspective as a complex, distributed execution environment, a set of requirements was derived for WSN-specific BPMN. Hence, we introduced BPMN extensions, including a WSN task, WSN pool and performance annotations, that fulfill these requirements. In a critical discussion, we compared our approach to modeling WSN logic with standard BPMN and were able to demonstrate the advantages of our approach. Being embedded in an end-to-end tool chain, the extended process models described in this paper can be used to generate executable code for the sensor nodes, replacing the need for an execution engine.

Future work includes evaluating the increase in programming efficiency of our approach for the software developer with a user study with users from different backgrounds, such as experienced BPMN modelers, WSN experts, and technical domain experts for the scenarios where WSNs are deployed. Furthermore, the current tool chain can only generate code for sensors based on the Contiki operating system. Other platforms may be added in the future. Finally, the approach will be tested on more scenarios where WSNs are deployed, especially in the areas of logistics and predictive maintenance.

Acknowledgment

This work is supported by the European Commission through the `make`*Sense* Project (http://www.project-makesense.eu/).

References

[1] I. Akyildiz and I. Kasimoglu, "Wireless sensor and actor networks: research challenges," *Ad Hoc Networks*, vol. 2, no. 4, pp. 351–367, Oct. 2004.

[2] K. Sohraby, D. Minoli, and T. Znati, *Wireless Sensor Networks: Technology, Protocols, and Applications*. Wiley-Interscience, 2007.

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, pp. 2787–2805, 2010.

[4] C. Buratti, A. Conti, D. Dardari, and R. Verdone, "An overview on wireless sensor networks technology and evolution," *Sensors*, vol. 9, pp. 6869–6896, 2009.

[5] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002.

[6] B. Akbal-Delibas, P. Boonma, and J. Suzuki, "Extensible and precise modeling for wireless sensor networks," in *Information Systems: Modeling, Development, and Integration*, 2009.

[7] A. Caracas and T. Kramp, "On the expressiveness of BPMN for modeling wireless sensor networks applications," in *Business Process Model and Notation – Third International Workshop*, 2011.

[8] A. Caracas and A. Bernauer, "Compiling business process models for sensor networks," in *Distributed Computing in Sensor Systems*, 2011.

[9] Object Management Group (OMG), *Business Process Model and Notation (BPMN) Version 2.0*, 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/

[10] L. Mottola and G. P. Picco, "Programming wireless sensor networks: Fundamental concepts and state of the art," *ACM Comput. Surv.*, vol. 43, pp. 19:1–19:51, 2011.

[11] S. Tranquillini *et al.*, "Process-based design and integration of wireless sensor network applications," in *Business Process Management*, 2012.

[12] J. Becker, C. Mathas, and A. Winkelmann, *Geschäftsprozessmanagement*. Springer, 2009, ch. Bedeutung des Geschäftsprozessmanagements.

[13] F. Leymann, "BPEL vs. BPMN 2.0: Should you care?" in *Business Process Modeling Notation*, 2010.

[14] N. Genon, P. Heymans, and D. Amyot, "Analysing the cognitive effectiveness of the BPMN 2.0 visual notation," in *Software Language Engineering*, 2011.

[15] O. Kopp *et al.*, "A Classification of BPEL Extensions," *Journal of Systems Integration*, vol. 2, pp. 2–28, 2011.

[16] S. Zor, F. Leymann, and D. Schumm, "A proposal of BPMN extensions for the manufacturing domain," in *CIRP Conference on Manufacturing Systems*, 2011.

[17] A. Rodríguez, E. Fernández-Medina, and M. Piattini, "A BPMN extension for the modeling of security requirements in business processes," *IEICE Transactions on Information and Systems*, vol. 90, pp. 745–752, 2007.

[18] A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel, "SecureBPMN: Modeling and enforcing access control requirements in business processes," in *ACM symposium on access control models and technologies*, 2012.

[19] O. Kopp, T. Binz, U. Breitenbcher, and F. Leymann, "BPMN4TOSCA: A domain-specific language to model management plans for composite applications," in *Business Process Model and Notation*, 2012.

[20] S. Stein, S. Kühne, and K. Ivanov, "Business to IT transformations revisited," in *Model-Driven Engineering for Business Process Management*, 2008.

[21] N. Glombitza, S. Ebers, D. Pfisterer, and S. Fischer, "Using BPEL to realize business processes for an internet of things," in *Ad-hoc, Mobile, and Wireless Networks*, 2011.

All links were last followed on 2013-05-29.