Institute of Architecture of Application Systems
University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3389

Extending BPMN for Wireless Sensor Networks

C. Timurhan Sungur

Course of Study:	Computer Science
Examiner:	Frank Leymann
Supervisor:	Oliver Kopp, Patrik Spiess, Nina Oertel, Sema Zor
Commenced:	August 17, 2012
Completed:	February 16, 2013
CR-Classification:	H.4.1, K.1

Abstract

As the *Wireless sensor/actuator networks* (WSNs) started to be deployed in enterprise scenarios, the need of integration of WSN applications with enterprise business processes has emerged. Creation of the WSN applications in enterprises, however, requires much low-level programming, and this causes the loss of focus on the high level goals. Therefore a model-driven approach is needed. *As Business Process Model and Notation* (BPMN) is the de-facto standard of modeling business processes, it is suitable for companies to use BPMN for the model-driven approach. In this work, first we analyze the WSN properties, which distinguish them from traditional IT systems. Thereafter, we analyze the general properties of BPMN which need to be preserved to keep the common understanding that BPMN brings. As a result, we end up with some properties that are relevant with modeling WSN processes in BPMN. From these properties, we derive the requirements for modeling WSNs and propose some extensions to standard BPMN. These extensions include a new type of task, a new type of pool and a new grouping structure to set performance goals of the corresponding WSN. Afterwards, we analyze the current state of art to compare our approach with other possible approaches. Specifically, we compare our approach with standard BPMN, BPM4People extensions with respect to requirements and show that why our proposals provide a more comprehensive and suitable approach.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

Internet of Things (IoT) by definition references two different concepts: “Internet” and “Things”. ‘Things” refers to smart objects which provide a way of communicating with real world [Fri11]. Those things or objects are currently examples of RFID tags, wireless sensors, actuators, mobile phones, etc. Because of non-passive and reactive behavior of WSNs unlike the most RFID tags, they will be an important part of IoT [AIM10]. Technological advances of Wireless Sensor Networks provided easier establishment and operating of Wireless Sensor Networks (WSNs). Low costs resulted in new types of application fields of WSNs [SMZ07]. WSNs provide a way of communication with real world by sensing it and reacting to the values sensed [MP11] [AIM10]. As a more concrete definition: “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.” [TSD+12]

A WSN node can also have different type of components such as location finder, mobilizer, power generator etc [ASSC02a]. These difference between type of hardware on nodes would define their role assignments, e.g., the nodes with temperature sensors would have sensor and temperature roles [CB11]. With variety of sensors provided on wireless sensor nodes, different type of application scenarios can be created. The type of sensors can be listed as [ASSC02a]:

- “temperature,
- humidity,
- vehicular movement,
- lightning condition,
- pressure,
- soil makeup,
- noise levels,
- the presence or absence of certain kinds of objects,
- mechanical stress levels on attached objects, and
- the current characteristics such as speed, direction, and size of an object”.

Some of the application scenarios that can be created using the type of sensors mentioned above can be listed as [ASSC02b]::

- Military applications
  - Monitoring friendly forces, equipment and ammunition

  - Battlefield surveillance
  - Reconnaissance of opposing forces and terrain
  - Targeting
  - Battle damage assessment
  - Nuclear, biological and chemical attack detection and reconnaissance
- Environmental applications
  - Forest fire detection
  - Biocomplexity mapping of the environment
  - Flood detection
  - Precision agriculture
- Health applications
  - Telemonitoring of human physiological data
  - Tracking and monitoring doctors and patients inside a hospital
  - Drug administration in hospitals
- Home applications
  - Home automation
  - Smart environment
- Other commercial applications
  - Environmental control in office buildings
  - Interactive museums
  - Managing inventory control
  - Vehicle tracking and detection

Some of the scenarios that have been listed above can be used in enterprises, e.g., "Environmental control in office buildings", "Managing inventory control", "Vehicle tracking and detection", etc. Some of these industrial applications have been surveyed in the works of Pentikainen et al. [PHM+08] and Edwin et al. [EPKG12]. In enterprise scenarios, usage of WSNs can lower the maintenance costs and can result in more flexible applications because no wired infrastructure is needed. Moreover they can make possible some maintenance applications which are not normally possible with a wired infrastructures [PHM+08]. WSNs are ad-hoc networks with special characteristics. These characteristics can be listed as [PHM+08, ASSC02b]: [GZBD11, TSD+12]

- Dense deployment and large number of nodes
- Nodes are failure-prone

- There is dynamic topology formation which changes frequently
- Limited resources (computation, power, memory)

Because of characteristic properties of WSNs, realization of WSNs is a challenge. During this realization, one should consider the following properties of the network [ASSC02a]:

- Fault-tolerance
- Scalability
- Cost
- Hardware
- Topology
- Environment
- Power Consumption

Among the above constraints, especially, power-constraints play an important role in realization of sensor networks, since the sensor network nodes carry limited or irreplaceable energy units all design decisions should consider limitation on power consumption [TSD+12]. Changing the battery of the nodes increase the operation costs of the WSNs and this change might take the feasibility of establishing a WSN.

The current state of art in the development of WSNs is model-driven development due to too much effort spent during development of low level functionality (see Subsection 1.1.2). The modeling need of WSN enabled business processes is emerged with enterprise usage scenarios of WSNs. As BPMN is a commonly used modeling language for business processes and the created models can be executed with necessary execution details, BPMN would be a good candidate to create models of WSN processes (see Subsection 1.1.1).

makeSense project is an EU project[1] which delivers a tool-chain to create deployable binaries from WSN process models which have been modeled using BPMN. This thesis work is conducted under makeSense and concentrated on BPMN part of the tool chain. Moreover this work is based on the extensions provided by Tranquillini et al. [TSD+12]. In the following section, we will define motivations behind choosing BPMN as the modeling language and behind the model-driven approach of WSN applications. After that, we will explain the methodology of this thesis work and give an outline of the thesis.

1.1 Motivation

In the following subsections, we will describe the motivations behind choosing BPMN and why a model-driven development is needed for WSN applications.

[1] http://www.project-makesense.eu

1.1.1 Reasons for Choosing BPMN

It is easier for business experts to model their business processes in a visual way [KAA97]. A well-known OMG standard is UML and its activity diagrams (ADs) is suitable for that however the aim of UML is not business processes but software systems. Even though business experts can reflect their business processes using UML ADs, they want an executable model which is not provided by ADs.

For such situations BPMN 2.0 is suitable, because BPMN 2.0 has its operational semantics, i.e., there are workflow engines which run BPMN and also BPMN has a subset of elements from BPEL [Org07] and this makes BPMN 2.0 partially convertible to BPEL [SKI08]. Hereafter BPMN 2.0 will be used interchangeably with BPMN.

Because the business processes reflected will be executable business processes partially on WSNs and workflow engines, we opt for BPMN to model such processes. BPMN has the properties that satisfy our requirements, i.e.:

- BPMN 2.0 has a well-known visual representation and is commonly used.
- BPMN 2.0 is executable.
- Direct conversion from BPMN 2.0 to BPEL is possible to some extend.
- There are open-source workflow engines which support BPMN 2.0, e.g., Activiti.
- BPMN 2.0 is extendable in case it is needed.

However, standard BPMN might be an over generalization and an extension might be required to address domain specific properties of WSNs.

1.1.2 Model-driven Development of WSNs

In WSNs, the importance of hardware cannot be undervalued but in order to use this hardware effectively, there is a need of proper software platforms available to software developers [MP11]. Sensing, processing and communication under a resource-constrained environment requires consideration from different layers of network stack, i.e., distributed signal/data processing, medium access control, communication protocols [BCDV09]. These programming challenges including scale and complexity of application, make the development effort complicated, time consuming and error-prone and creates a need for higher level of abstractions [ADBS09]. There are some approaches being proposed to achieve easy development, and provide users higher-level constructs for programming, e.g., node-centric programming and macro-programming. The former represents an abstraction at a level of node, local actions of a node and the latter represents a higher level of abstraction, i.e., at the level of network [GGG05]. Regardless from these advancements, still the weakest link of Wireless Sensor Networks is programming of these networks [MP11]. By raising the level of abstraction for programmers instead of making them to deal with too many low-level details, this problem can be solved [EBSK10].

High coupling between application logic and underlying sensor leaves WSN projects with platform dependent code which is difficult to maintain, modify and re-use. This can be avoided by using abstraction of low-level details and by representing the domain features with the languages that domain experts are familiar to [RDD+11]. These abstractions should not be over generalized nor over specialized [ADBS09].

The programming challenges of WSNs can be overcome by using model-driven approaches. By this way WSNs can be used without dealing with low-level programming details. WSNs have enterprise usage scenarios, these scenarios would make BPMN a suitable modeling language to integrate enterprise scenarios in enterprises. makeSense is an European Union (EU) project where a tool-chain has been propose to model business processes in BPMN. The necessary implementation details are added to these BPMN models afterwards these models are used to create binaries which will be deployed on WSN nodes. To achieve this we need a means to create BPMN models with necessary details that cover all relevant WSN properties.

The general flow of the model-driven code generation is shown in Figure 1.1. A BPMN model references WSN operations, modes, etc. This BPMN model is used to generate binaries in a tool chain which are deployed on the nodes as a next step. After WSN application is set-up using these WSN execution ready nodes, the WSN nodes execute the process modeled previously.

1.2 Methodology

In this thesis work, we define some BPMN extensions, which do not change semantics of standard BPMN and adds the necessary details to make models executable. To create these extensions, we need to analyze the properties that make WSNs unique and we need to analyze important and relevant BPMN properties which might be important during creation of the WSN extensions. These properties are later on used to derive our requirements from which we create our extensions. During extensions we do not change existing BPMN semantics. A summary of the thesis work can be found in the Figure 1.2

The “Literature Review” task in this work can be described in three steps as suggested by Eli et al. [LE06]. As the input phase, we analyzed literature related with WSNs, BPMN, BPM, Cognitive Effectiveness of BPMN etc. Main focus during literature review was to understand distinguishing properties of WSNs from standard business processes and the properties of BPMN properly. After necessary processing, the list of properties and corresponding descriptions can be found in Chapter 2. Another part of the literature review was analyzing the input about the current state of art which can be found in Chapter 5. During literature review, we used backward and forward search methods as suggested by Eli et al. [LE06].

The properties found during “Literature Review” are analyzed in the following tasks, i.e., “Analyze BPMN Properties” and “Analyze WSN Properties”. The analysis can be found in Chapter 3. From this analysis, we elect related properties from different both BPMN and WSN domains (see Section 3.2 and Section 3.1).

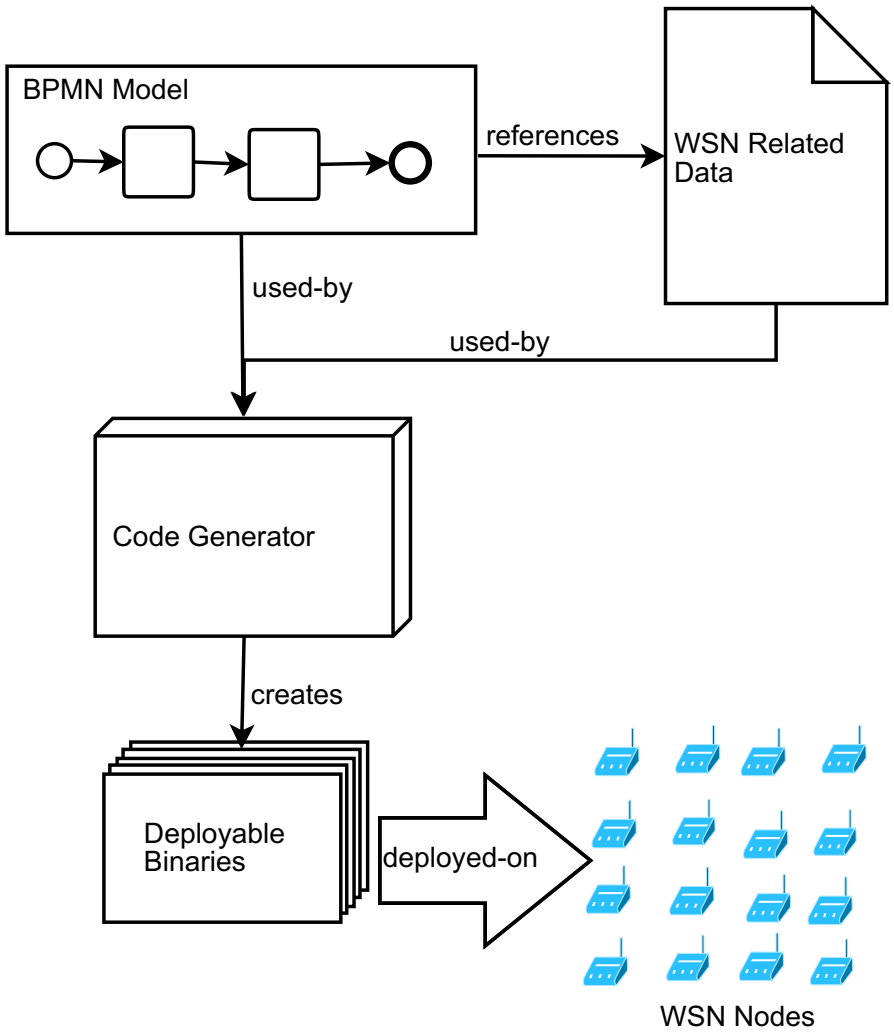


Figure 1.1: The model-driven development of WSNs.

In the Section 3.3, we derive our requirements from the properties that we have found related. By defining our requirements, we conclude the task "Derive Requirements" in the methodology model (Figure 1.2).

In the next task "Create Solution Proposals", we propose our extensions which satisfy the requirements that we have previously defined (see Chapter 4). During creation of these extensions, we use the BPMN extension methodology and we preserve semantics of the existing BPMN properties (see Section 2.2).

As final step task, we evaluate our approach by comparing it with the current state of art. The evaluation can be found in chapter 6. By providing this evaluation, we conclude the task "Evaluate Solution Proposals".

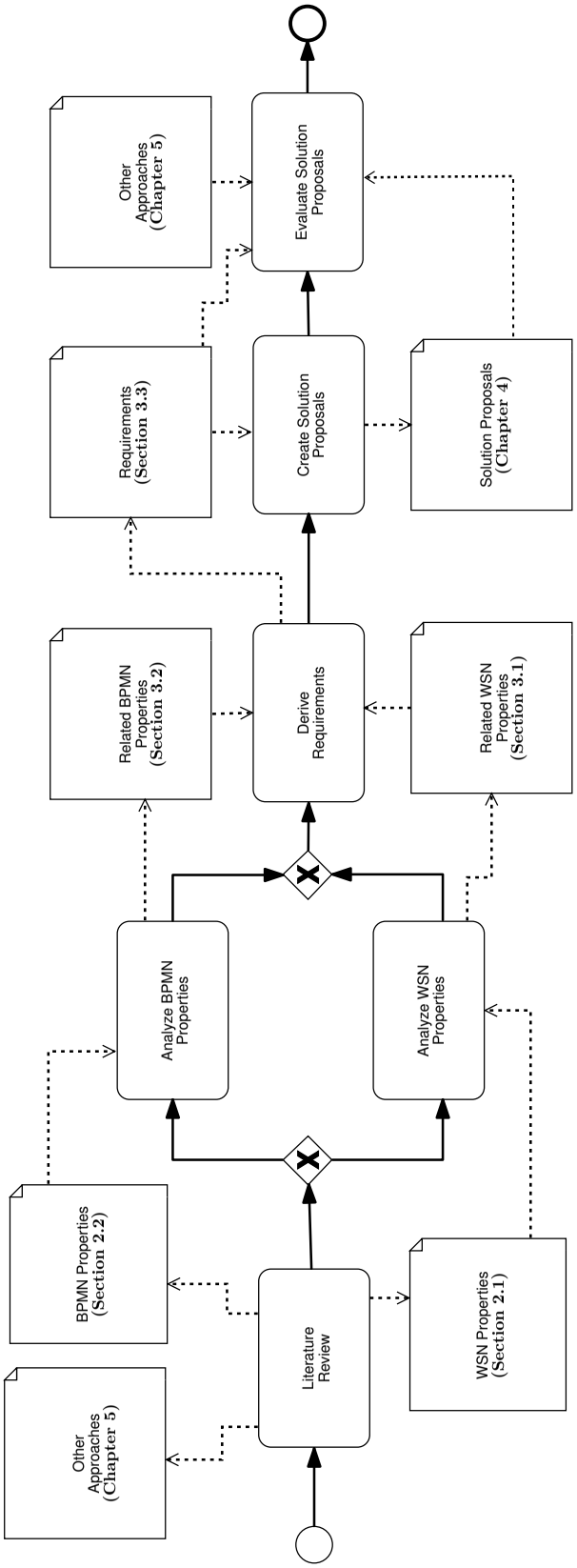


Figure 1.2: The thesis methodology model.

This work is conducted under makeSense project and solutions are based on the extensions provided by Tranquillini et al. [TSD+12]. In the following chapters we will firstly analyze WSN and BPMN properties (see Chapter 2), define our requirements based on these properties (see Chapter 3), propose our extensions (see Chapter 4) and compare our solution proposals with already existing solutions (see Chapter 6). In the following chapter, we describe how we extended a modeler with our extensions (see Chapter 7). In the last chapter, we will give a summary and an outlook about our contribution (see Chapter 8).

2 Background on WSNs and BPMN

In the following sections, we will first introduce properties of WSNs. Other properties can be added to these properties however we present the ones that we found most significant. After presenting WSN properties, we go through some general properties of BPMN and corresponding business management systems.

2.1 WSN Properties

In the following sections, we will introduce some general properties of WSNs. Although all of these properties are important for wireless sensor networks, not all of them are relevant during business process modeling. In Section 3.1, we will give the reasoning of which properties are relevant with business process modeling.

2.1.1 Dynamic Addition and Removal of Wireless Sensor Nodes (WP1)

In WSNs, to increase sensing accuracy or to increase the coverage area of the WSN, new nodes can be added even after application has been started. Moreover, nodes might turn off as a result of exhausting their limited energy resources [SMZ07, ASSC02b]. This is why, for WSNs, direct addressing, i.e., addressing single nodes by unique identifiers, is in general not beneficial. An attribute based, indirect addressing can solve this problem [MP11]. For indirect addressing, general common properties of the nodes are used to address them instead of their uniquely identifying properties, e.g., “sensors”, “temperature sensors”, “sensors in room number 5”, etc. In contrast, direct addressing of a node can be the MAC address of this node where a unique identifier of the respective node is used. The target nodes referred to by indirect addressing can either be identified during run-time or during creation of executable code which will be deployed on nodes.

2.1.2 Categories of WSN Operations (WP2)

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, because WSN nodes can be commanded to execute operations 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 either [TSD+12]:

- **Sense** Used to analyze surrounding environment,e.g., sensing temperature, pressure, etc. Usually, this operation creates some output data.
- **Actuate** Used to manipulate surrounding environment (e.g., increase ventilation) or to signal to the user (e.g. by flashing an LED indicator)
- **Intermediary operation**
  - Send/receive data from outer world, e.g., the sensed values.
  - Make decisions, e.g., decide if the temperature is above the threshold.
  - Do computations on received data, e.g., average of CO_2 data.

Both local and command actions may create output values, which are consumed by other nodes [AK04]. Especially, sense actions create data, which is consumed by other nodes later on for some application specific purposes. Successively sensing and using this created data can be called combined operations because they can not be executed independently, i.e., execution of the latter operation depends on the created output data of the former one. Sensing presence data (of people in a space) would be composed of two steps: sensing presence and aggregating it to predict presence more precisely. This implies that a WSN action might be composed of an initial operation which outputs data and another operation which inputs the output data of the initial one.

2.1.3 Limited Operations Available in WSNs (WP3)

WSN applications are applications with no human involvement. After deployment, they operate automatically [ASSC02b]. Usually, WSNs do not receive manual input and none of the tasks in a WSN centric process is executed manually. Changing some WSN nodes can be an example of a manual operation however this would not affect the process flow.

2.1.4 Parallel Execution of the Same Process Logic in one Application (WP4)

Computations in a WSN can be done as local on node, as a group or in a global setting. Some WSN applications can execute the same process logic at different regions of the WSN in one application [MP11]. A simple example is an air-conditioner 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.

In case of Wireless Sensor Network applications, there can be two level of parallelism. The first is at the node level and the second is at process level. Different nodes in the same process might be executing the same operation. Usually, these group of nodes are selected by using attribute based addressing, by this way addition and removal of nodes are more flexible than direct addressing.

2.1.5 Distributed Nature of WSN Applications (WP5)

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 life time of the WSN application [AK04].

2.1.6 Limited Resources and Error-prone Nature of WSN Nodes (WP6)

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 [ASSC02b]. There is a tradeoff between power-consumption and amount of communication in WSNs [ASSC02b]. WSNs provide different routing protocols with different properties such as low-cost maintenance, simplicity, network life time, node life time, etc... [SMZ07, ASSC02b] and currently available programming abstractions of WSNs can hide these details under some level of abstraction. By reducing constraints on power consumption, nodes can can communicate more effectively and reliably, however this would result in shorter running WSN application. 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.

2.1.7 Event-driven Nature of WSNs (WP7)

Communication is the most expensive operation in WSNs [ASSC02b]. 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 [MP11, CB11]. During the execution, event-driven operations communicate less than periodic operations since they go into an idle mode until the specified event happens. Consequently, event-driven operations consume less energy compared to periodic operations [MP11, CB11]. 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 sinks. 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.

2.1.8 Different Type of Nodes (WP8)

In a WSN, there can be nodes with various hardware specifications. The selection of hardware can prolong life time of the WSN, increase data accuracy and decrease failure ratio. With the help of hardware specifications, roles can be assigned to nodes which can be used during the node selection later on [CB11, ASSC02b].

2.1.9 Dense Deployment of Nodes (WP9)

In WSN nodes are densely deployed in order to decrease communication costs [ASSC02b]. This property is important during creating routing algorithms, failure tolerance levels, scalability, etc. of WSN however, they do not add a business value and there is nothing to be represented about them at the level of business modeling. After business analysis if there was a problem at that step, then these properties might be re-configured.

2.1.10 Different Interaction Patterns in WSNs (WP10)

There are three categories of interactions in Wireless Sensor Networks [MP11]:

- Many-to-one,
- One-to-many,
- Many-to-many

Application goal of WSN has a high affect on this interaction pattern. In sense-only applications, mostly, there are a many-to-one interaction pattern and as well some one-to-many interactions for configuration purposes. In sense-and-react applications, many-to-many interaction pattern is a common thing.

2.2 BPMN Properties

BPMN is the de-facto standard for business process modeling [Rec10] and business experts model their business processes mostly using BPMN [SAP09]. BPMN models can be executed on business engines. Therefore it is used by developers and business experts and bridges the gap among them. BPMN uses visual constructs to model business processes which are high level abstractions of actual activities that occur in business processes. The current standard represents a BPMN model which is aimed to be readable, flexible and expandable. In the following subsections, we will give some properties of general business process management concepts and BPMN.

2.2.1 Dimensions of a Business Process Activity (BP1)

The BPMN users wanted their models to be executed and with the version 2.0 of BPMN standard BPMN 2.0 has its operational semantics and can be executed on its own workflow engine. [Ley10]. The life cycle of a business process can be observed in Figure Figure 2.1.

In workflows, typically there are three dimensions to define workflow items [LR00]:

- “What”: What is the work item?, e.g., “Check Customer”
- “With”: With what should this work item be accomplished?,e.g., “a Web Service”

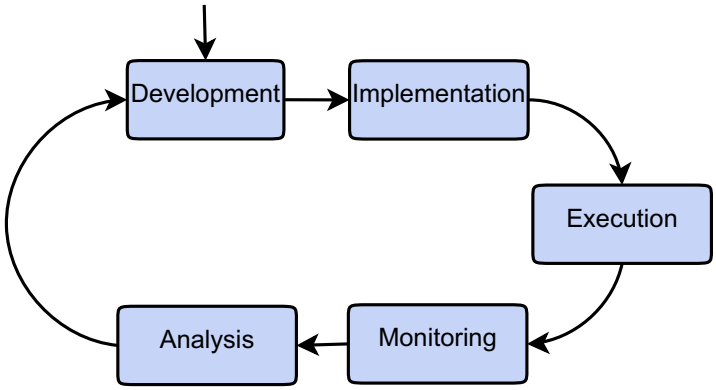


Figure 2.1: Busines Process Life-cycle [Obj11]

- "Who": Who will accomplish this work item?, e.g., "Computing Resources"

2.2.2 Different Levels of Modeling (BP2)

BPMN has 3 levels of modeling, "Descriptive Modeling", "Analytical Modeling" and "Executable Modeling". The first 2 levels of modeling are for documentation purposes. In the first level, there can be errors in the model whereas in the second level there is expected to be no errors. In the 3rd level of modeling, software developers adds execution details to the existing modeling constructs and make the models ready for execution on business process engine [Sil11]. The generated BPMN file is executed on a business process engine and process engine orchestrates defined set of activities.

There are some BPMN vendors who created their execution engines, e.g., Activiti, jBPM, Bonitasoft, etc. Such BPMN vendors provide certain extensions points that they think these extensions will be needed in most of the business cases [Act, jBP12]. There are also some extensions included on some BPMN engines, .e.g, Activiti and jBPM. The list of available BPMN elements including their extensions can be found under [Act, jBP12].

2.2.3 Cognitive Effectiveness of BPMN (BP3)

The cognitive effectiveness of BPMN model is also important because it is used by different group of users with different backgrounds. With high cognitive effectiveness, a better common understanding can be achieved. In [GHA11] BPMN analyzed using principles of the "Physics of Notation" theory and how BPMN provides cognitive effectiveness is explained including its missing points. An example of "Physics of Notation" is "Semiotic Clarity". "Semiotic Clarity" means a 1-1 relation between semantic description and the visual constructs [Moo09].

2.2.4 Extensbility Mechanism of BPMN (BP4)

In certain domain applications, modeling elements of BPMN might not be sufficient. In these situations domain experts can extend BPMN meta-model to achieve a better reflection of their own application domains. BPMN extensions are done by adding new elements and attributes to existing BPMN elements. These new elements and attributes should not contradict with already existing elements and attributes. This approach guarantees interchangeability of existing BPMN constructs and possible interchangeability of extension element and attributes [Obj11].

There are 4 BPMN Elements defined to extend BPMN [Obj11]:

- Extension
  - This is used to bind independent ExtensionDefitinion elements to the model definition.
  - It is contained in a Definition element.
  - Has following attributes.
    - mustUnderstand: It is a Boolean (default: false) and it is used to if this extension must be understood in order to process model correctly.
    - definition: In XML schema, this definition is provided by an external XML Schema file by using its QName
- ExtensionDefinition
  - Defines groups of additional attributes.
  - This is not used when XML Schema interchange is used since <xs:complexType> type already satisfies this requirement.
- ExtensionAttributeDefinition
  - A list of attributes which can be attached to any BPMN elements and this list defines name and type of the new attributes.
  - This is not used when XML Schema interchange is used since <xs:AnyAttribute> and <xs:Any> type already satisfy this requirement.
- ExtensionAttributeValue
  - Value of the newly defined attribute.
  - This is not used when XML Schema interchange is used since <xs:AnyAttribute> and <xs:Any> type already satisfy this requirement.

Meta Object Facility (MOF)	XML Schema
Extension	<extension>
ExtensionDefinition	<xsd:group>
ExtensionAttributeDefinition	<xsd:element> in a <xsd:group»
ExtensionAttributeValue	<extensionElements> of BPMN XML Schema meta-model

Table 2.1: BPMN XML Schema and MOF Meta-model Extension Element Mappings [SCV11]

Listing 2.1 baseElement schema definition [Obj11]

```
<xsd:element name="baseElement" type="tBaseElement"/>
  <xsd:complexType name="tBaseElement" abstract="true">
    <xsd:sequence>
      <xsd:element ref="documentation" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="extensionElements" minOccurs="0" maxOccurs="1" />
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:ID" use="optional"/>
  <xsd:anyAttribute namespace="\#\#other" processContents="lax"/>
</xsd:complexType>
```

MOF meta-model description and XML Schema description of BPMN are not equivalent with respect to extensions. Table 2.1 shows the corresponding mappings between MOF interchange and XML Schema interchange with respect to the extensions. These types are created using <xsd:simpleType> and <xsd:complexType> [SCV11]. BPMN adopters can use the methodology and tool support mentioned [SCV11].

To extend BPMN schema using XML Schema interchange, BPMN adopters need to first define the extensions in an external schema definition, where new elements and attributes are defined. Using <extension> and <import> tag, this externally defined schema is linked to the model and new elements are ready to use under <extensionElements>. As it can be observed from Listing 2.1 and Listing 2.2, additional attributes and also additional elements have *"##other"* namespace definitions which means they should be defined in other namespace but not in *"http://www.omg.org/spec/BPMN/20100524/MODEL"*.

To illustrate an example meta-model extension, the method and tool provided in [SCV11]. We will give a simple example, i.e., a telephone task, which is an extension of a "Manual Task" and has additional following attributes:

Listing 2.2 extensionElements XML Schema [Obj11]

```
<xsd:element name="extensionElements" type="tExtensionElements" />
<xsd:complexType name="tExtensionElements">
       <xsd:sequence>
              <xsd:any namespace="##other" processContents="lax" minOccurs="0"
                  maxOccurs="unbounded" />
       </xsd:sequence>
</xsd:complexType>
```

Listing 2.3 Extension Schema Example: Telephone Task

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
       xmlns="http://www.project-makesense.eu/bpmn/extensions/TelephoneTask"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
       targetNamespace="http://www.project-makesense.eu/bpmn/extensions/TelephoneTask">
       <xsd:import namespace="http://www.omg.org/spec/BPMN/20100524/MODEL"
           schemaLocation="BPMN20.xsd"/>
              <xsd:group id="TelephoneTask" name="TelephoneTask">
                <xsd:sequence>
                       <xsd:element name="nameOfContact" type="xsd:string" minOccurs="1"
                           maxOccurs="1"/>
                       <xsd:element name="numberToDial" type="xsd:string" minOccurs="1"
                           maxOccurs="1"/>
                       <xsd:element name="conversationTopic" type="xsd:string" minOccurs="1"
                           maxOccurs="1"/>
                </xsd:sequence>
       </xsd:group>
</xsd:schema>
```

- numberToDial: String
- nameOfContact: String
- conversationTopic: String

When we apply the method using Eclipse[1] and the plug-in which is provided by [SCV11] and found on "http://code.google.com/p/bpmnx/", we end up with Figure 2.2. To create this UML model we used the rules 3, 4a, 5, 7 found in [SCV11]. after running necessary transformations with the tool provided, the resulting extensions schema can be observed in Listing 2.3. In a concrete model, we need to add an extension and import tags under definitions and we need to link generated extension schema. A more convenient way of integrating our schema can be observed in Listing 2.4. During our extensions, we will not follow the tool support and methodology proposed by [SCV11] because it brings additional modeling effort and does not bring extra advantages.

The extended BPMN model will have a tag as <extension mustUnderstand="true" definition="http://www.project-makesense.eu/bpmn/extensions/TelephoneTask"> and then the extension elements will be stored under <extensionElements> tag as illustrated in Listing 2.5.

This example is only given to show how extensions are defined in BPMN standard. Changing BPMN schema is not the way defined in BPMN standard and in such cases interoperability of created BPMN models would diminish significantly.

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

Listing 2.4 Extension Schema Example: Telephone Task (Modified)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema elementFormDefault="qualified" attributeFormDefault="unqualified"
       xmlns="http://www.project-makesense.eu/bpmn/extensions/TelephoneTask"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:bpmn="http://www.omg.org/spec/BPMN/20100524/MODEL"
       targetNamespace="http://www.project-makesense.eu/bpmn/extensions/TelephoneTask">
       <xsd:import namespace="http://www.omg.org/spec/BPMN/20100524/MODEL"
           schemaLocation="BPMN20.xsd"/>
       <xsd:element name="TelephoneTaskProperties">
              <complexType>
                     <xsd:group ref="TelephoneTask"/>
              </complexType>
       </xsd:element>
       <xsd:group id="TelephoneTask" name="TelephoneTask">
         <xsd:sequence>
                <xsd:element name="nameOfContact" type="xsd:string" minOccurs="1"
                    maxOccurs="1"/>
                <xsd:element name="numberToDial" type="xsd:string" minOccurs="1"
                    maxOccurs="1"/>
                <xsd:element name="conversationTopic" type="xsd:string" minOccurs="1"
                    maxOccurs="1"/>
         </xsd:sequence>
       </xsd:group>
</xsd:schema>
```

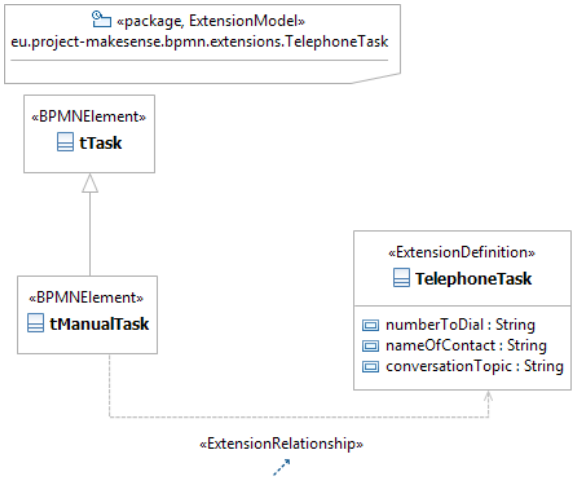


Figure 2.2: Manual Task Extension: Telephone Task

Listing 2.5 A Concrete Example Based on the Created Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
    xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
    xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
    xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
    xmlns:signavio="http://www.signavio.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" exporter="Signavio Process Editor,
    http://www.signavio.com" exporterVersion=""
    expressionLanguage="http://www.w3.org/1999/XPath"
    id="sid-1ecbd0d2-3ab7-400c-b889-224d0df734a8"
    targetNamespace="http://www.signavio.com/bpmn20"
    typeLanguage="http://www.w3.org/2001/XMLSchema"
    xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL
    http://www.omg.org/spec/BPMN/2.0/20100501/BPMN20.xsd">
   <import importType="http://www.w3.org/2001/XMLSchema" location=""
       namespace="http://www.project-makesense.eu/bpmn/extensions/TelephoneTask"/>
   <extension definition="bpmntel:TelephoneTaskProperties" mustUnderstand="true"/>
   <process id="sid-ff1b100c-d527-4a74-af24-b3df4fd3994c" isExecutable="false">
     <manualTask completionQuantity="1" id="sid-28B5FA9F-26C1-4C9E-85EC-5FB1E3663A50"
         isForCompensation="true" name="Make New Order" startQuantity="1">
        <extensionElements>
          <bpmn4tel:TelephoneTaskProperties
              xmlns:bpmn4tel="http://www.project-makesense.eu/bpmn/extensions/TelephoneTask">
             <bpmn4tel:nameOfContact></bpmn4tel:nameOfContact>
             <bpmn4tel:numberToDial>00 49 711 1 2 3</bpmn4tel:numberToDial>
             <bpmn4tel:conversationTopic>New Order</bpmn4tel:conversationTopic>
           </bpmn4tel:TelephoneTaskProperties>
        </extensionElements>
     </manualTask>
   </process>
   <bpmndi:BPMNDiagram id="sid-88456aa5-2408-496a-a617-32e9753b1f0a">
      <bpmndi:BPMNPlane bpmnElement="sid-ff1b100c-d527-4a74-af24-b3df4fd3994c"
          id="sid-7c10069a-9729-41d3-8a62-fc58f563c3bb">
        <bpmndi:BPMNShape bpmnElement="sid-28B5FA9F-26C1-4C9E-85EC-5FB1E3663A50"
            id="sid-28B5FA9F-26C1-4C9E-85EC-5FB1E3663A50_gui">
          <omgdc:Bounds height="80.0" width="100.0" x="75.0" y="210.0"/>
        </bpmndi:BPMNShape>
      </bpmndi:BPMNPlane>
   </bpmndi:BPMNDiagram>
</definitions>
```

2.2.5 Modifiability of Process Models (BP5)

Some of the created models are representations of internal business processes and they are executed by automated execution engines. These models are refined after each execution of the modeled business process [Wes07]. An easier modifiability provided by the business model, will ease these modifications.

2.3 Summary

In this section, we provided background on WSNs, BPMN and general business process management. The properties of WSNs are general identifying properties and their relevance will be evaluated in Chapter 3. It is similar in case of BPMN. Even though some of the BPMN and business process management properties are not relevant for our requirements, they are relevant while creating our extensions and we need to keep them in our minds during this process. The properties of WSNs can be listed as:

- **Dynamic Addition and Removal of Wireless Sensor Nodes (WP1)**: New nodes can be added or existing nodes can be removed from WSNs so there exists a dynamic topology. Nodes can be addressed directly or indirectly, i.e., with their unique identifiers or with logical addresses, which are assigned by the system respectively.
- **Categories of WSN Operations (WP2)**: In WSNs there exits command and local actions, during command actions one node commands another node to execute an action whereas a local action is executed without any communication. This action can be a type of sense, actuate and intermediary action. After execution of command and local actions, there might exist an output data which is returned to a group of nodes and there might exist some operations on the output data such as aggregation.
- **Limited Operations Available in WSNs (WP3)**: The operations executed on WSNs are autonomous, they usually execute without human involvement.
- **Parallel Execution of the Same Process Logic in one Application (WP4)**: The same process logic can be executed at the different places in the same WSN in parallel.
- **Distributed Nature of WSN Applications (WP5)**: There might be multiple nodes which control the execution of the WSN application.
- **Limited Resources and Error-prone Nature of WSN Nodes (WP6)**: Resources on WSN nodes are limited and because of this there are trade-offs such as reliable communication and application life time.
- **Event-driven Nature of WSNs (WP7)**:There are event-driven and periodic actions in WSNs from time perspective. Event-driven actions release the execution flow directly whereas periodic actions wait until the task is completed to release the control.
- **Dense Deployment of Nodes (WP9)**: Nodes are densely deployed in WSNs.

- **Different Interaction Patterns in WSNs (WP10)**: There might be one-to-many, many-to-one and many-to-many interaction patterns in WSNs.

After that we defined general properties of BPMN, please note that these properties can be only a small subset of all BPMN properties:

- **Dimensions of a Business Process Activity (BP1)**: A business process item consists of three dimensions: "What is the work item?", "With what should this work item be accomplished?" and "Who will accomplish this work item?".
- **Different Levels of Modeling (BP2)**: There are three levels of BPMN modeling: "Descriptive Modeling", "Anayltical Modeling" and "Executable Modeling". The first two levels are used for documentation whereas the third level is used for execution.
- **Cognitive Effectiveness of BPMN (BP3)**: BPMN brings a common understanding to the people with different background. Therefore the cognitive effectiveness of BPMN is important. An example of measuring cognitive effectiveness is "Semiotic Clarity" and this is defined in "Physics of Notation" [Moo09].
- **Extensbility Mechanism of BPMN (BP4)**: BPMN is designed to be extensible for domain specific applications. These extensions should be contained under <extensionElements> tag and the extensions schema should be imported with <import> tag. With the <extension> tag an extension is bound to the BPMN model.
- **Modifiability of Process Models (BP5)**: BPMN models are revised iteratively therefore the modifiability of BPMN models is important.

3 Requirements for WSN-specific BPMN

There are some properties that need to be preserved during BPMN modeling, on the other hand there are some properties which are critical for WSNs and these should be represented in BPMN models to integrate WSNs to business processes. Some of those requirements might not appear in some BPMN models because not all the diagrams have the same nature, i.e., some diagrams are for documentary purposes not execution purposes. In the next section, relevance of WSN and BPMN properties will be evaluated afterwards in the next section, requirements for WSN-specific BPMN will be introduced.

3.1 Evaluation of WSN Properties

To evaluate WSN properties, we need to consider their importance during BPMN modeling and also their importance for WSNs. In the next sections, we will make our evaluations and state if they are relevant for business process modeling or not. Relevancy can be for two different purposes in our approach, i.e., for documentation purposes and for execution purposes because BPMN is used for documentation or execution purposes. With a relevancy for documentation, we mean that the inclusion of this property enriches the model expressiveness in case of WSN modeling. With a relevancy for execution, we mean that this property could be used for execution of the corresponding property.

3.1.1 Dynamic Addition and Removal of Wireless Sensor Nodes (WP1)

WSNs have a dynamic topology and the WSN nodes which execute WSN operations might change. We need to have the ability to define the nodes which are responsible for execution of some operations. This makes this property relevant for the execution purposes because during execution we would need a means to address nodes.

3.1.2 Categories of WSN Operations (WP2)

The type of operation executed by a WSN would be affect the creation of the deployable binaries and the expressiveness of the models. Operation categories on a visual level would increase the expressiveness of the models and only with the necessary operation details execution would be possible. Therefore this property is relevant for execution and documentation purposes.

3.1.3 Limited Operations Available in WSNs (WP3)

Not all the operations provided by visual constructs can be executed in a WSN. The operation constrained by the underlying WSN should also constrain the models in a visual way. Moreover, as expected a WSN cannot provide more than it offers. Therefore this property is relevant for both execution and documentation.

3.1.4 Parallel Execution of the Same Process Logic in one Application (WP4)

Parallel execution of the same process logic is a usual thing in case of standard business processes. However, in case of WSNs, concurrent WSN processes can be executed on WSN itself not on a central execution engine. This critical difference makes this property relevant for documentation and execution purposes.

3.1.5 Distributed Nature of WSN Applications (WP5)

As the WSN processes executed on WSNs themselves, process orchestration is not managed by the central process execution engine. This difference makes this property relevant for execution purposes.

3.1.6 Limited Resources and Error-prone Nature of WSN Nodes (WP6)

Limited resources and error-prone nature of WSNs should be considered during BPMN so that no critical failures occur. This property is relevant for execution and documentation purposes because the regions where the performance changes can be documented moreover performances can be set for the execution.

3.1.7 Event-driven Nature of WSNs (WP7)

The difference in timing of operations would bring different expectations. This difference makes this property relevant for documentation purposes.

3.1.8 Different Type of Nodes (WP8)

Different type of nodes does not affect the model because by properties WP1 and WP2 cover this property indirectly. The difference between hardware would result in different addressing or different operations. Therefore this property is irrelevant.

3.1.9 Dense Deployment of Nodes (WP9)

Dense deployment of WSN nodes is not a relevant property during business process modeling of WSNs.

3.1.10 Different Interaction Patterns in WSNs (WP10)

The interaction patterns in WSNs are not relevant with BPMN modeling due to their low-level nature. As a result of different interaction patterns, different application goals can be observed, e.g., sense, react, etc., however this has been covered in another property (see section 2.1.2).

3.2 Evaluation of BPMN Properties

The BPMN Properties are mostly included in extensions, i.e., proposed extensions will not remove the property given. In the following sections, we will analyze relevance of the provided BPMN properties with our requirements.

3.2.1 Dimensions of a Business Process Activity (BP1)

The dimensions of the workflow items are also important for the business processes which include WSNs. This property is relevant to create a valid activity in a workflow engine which executes a business process of WSN, i.e., in our case WSN is the workflow engine.

3.2.2 Different Levels of Modeling (BP2)

The different levels of modeling is also similar in case of modeling the business processes which include WSNs. First a valid business process model is created and afterwards execution details are added however this property is not relevant with requirements. This property will be useful during evaluation of different approaches.

3.2.3 Cognitive Effectiveness of BPMN (BP3)

This property is important to preserve the common understanding of the BPMN. The created extensions should give the same look and feel as the existing BPMN constructs and the suggestions given by Genon et al. [GHA11] should be followed. This property has been taken as a guide during creation of our visual extensions and it is also relevant for our requirements.

Property Name	Relevant
Dynamic Addition and Removal of Wireless Sensor Nodes (WP1)	Yes
Categories of WSN Operations (WP2)	Yes
Limited Operations Available in WSNs (WP3)	Yes
Parallel Execution of the Same Process Logic in one Application (WP4)	Yes
Distributed Nature of WSN Applications (WP5)	Yes
Limited Resources and Error-prone Nature of WSN Nodes (WP6)	Yes
Event-driven Nature of WSNs (WP7)	Yes
Different Type of Nodes (WP8)	No
Dense Deployment of Nodes (WP9)	No
Different Interaction Patterns in WSNs (WP10)	No
Dimensions of a Business Process Activity (BP1)	Yes
Modifiability of Process Models (BP5)	Yes
Cognitive Effectiveness of BPMN (BP3)	Yes
Different Levels of Modeling (BP2)	No
Extensbility Mechanism of BPMN (BP4)	No

Table 3.1: Evaluation of WSN properties.

3.2.4 Extensbility Mechanism of BPMN (BP4)

This property is only informal and is not a relevant property with the requirements. This information contained in this property has been used during creation of our BPMN extensions.

3.2.5 Modifiability of Process Models (BP5)

The base modifiability of the BPMN needs to be preserved or the changes in modifiability should be minimized. Therefore this property is a relevant property.

3.3 Requirements

In this section, we will derive our requirements from the relevant properties (see Table 3.1).

3.3.1 Support for Indirect and Dynamic Addressing of Nodes (R1)

For the purpose of supporting a dynamically changing set of nodes (WP1), we need a means to address nodes indirectly. To increase flexibility of the created WSN applications, designer may want to use indirect addressing. For instance, 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. For instance, in a WSN application, instances of the same process might be executed at different regions of the WSN application, in this case there would be a need of dynamic selection to address the correct region of the WSN.

3.3.2 Support and Restrict User to WSN Operation Categories (R2)

Sense, actuate, and intermediary operations (WP2) behave differently during execution: Sense operations are used to observe the environment, intermediary operations are usually used to do some computation on sensed values, and actuate operations are used to manipulate the environment. Local actions are initiated by the executing nodes themselves, whereas command actions are initiated by some nodes and executed by other nodes. This difference in execution logic demands necessary differentiation between local and command actions at the modeling. The executed operation might create some output data and this data might be an input for a consequent operation. Therefore, we need the ability to define the nodes which receive the created output data and the possible operation they might apply on the output data. 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:

1. The type of operation: {Sense, Actuate, Intermediary Operation}.
2. Location of the operation: {Local, Command}.
3. Target nodes which will execute this operation.
4. An operation definition.
5. Output target nodes, which will receive the output results.
6. An operation definition, which is applied on the output data.

With this 6-tuple, we can define ‘what’, ‘with’ and ‘who’ dimensions of the corresponding workflow item BP1 (Subsection 2.2.1).

3.3.3 Limit Available Operations for WSNs (R3)

During modeling of businesses processes that run on WSNs, modelers should be limited with the operations provided by the underlying WSN to create their business processes (WP3). BPMN provides some constructs which cannot be executed on a WSN, e.g., “Manual Task”. We need a means to limit such BPMN constructs in WSN pools. Otherwise the created BPMN models might be unrealistic moreover, there could be problems while creating the deployable binaries.

3.3.4 Support for Multiple Instances of the Same Process (R4)

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 (WP4). Therefore, explicit separation of WSN processes from standard business processes and support for the visual representation of parallel processes are needed.

3.3.5 Distribution of Execution Logic into WSN (R5)

In a WSN process, there are events and data, which are produced and consumed by some nodes in a WSN (WP5). The produced data and events can be used to make decisions in the business process. This decision mechanism is executed seamlessly in business process execution engines in case of conventional business processes; however, this is not the case for the processes executed in a WSN. As we want to execute the modeled processes completely in a WSN, we need a means to provide 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.

3.3.6 Prioritization of Performance Goals (R6)

In WSNs, there may be application specific conflicting performance goals, e.g., shorter-response time, reliable communication, lower power consumption (WP6). 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. By defining these performance goals, we are expecting a WSN to execute the corresponding operations aligned with the performance goals defined. For instance, in case a fire is detection scenario, reliable message transmission will be needed and response time needs to be short.

3.3.7 Support for Event-driven Actions in Modeling (R7)

WSNs provide both event-driven and periodic operations, the behavior of event-driven WSN actions (WP7) 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 (BP3). 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.

3.3.8 Models should be stable on minor WSN changes (R8)

Business processes run on WSNs should be as flexibly modifiable as standard BPMN process models. 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

Requirement Name	Properties
Support for Indirect and Dynamic Addressing of Nodes (R1)	Dynamic Addition and Removal of Wireless Sensor Nodes (WP1)
Support and Restrict User to WSN Operation Categories (R2)	Categories of WSN Operations (WP2) Dimensions of a Business Process Activity (BP1)
Limit Available Operations for WSNs (R3)	Limited Operations Available in WSNs (WP3)
Support for Multiple Instances of the Same Process (R4)	Parallel Execution of the Same Process Logic in one Application (WP4)
Distribution of Execution Logic into WSN (R5)	Distributed Nature of WSN Applications (WP5)
Prioritization of Performance Goals (R6)	Limited Resources and Error-prone Nature of WSN Nodes (WP6)
Support for Event-driven Actions in Modeling (R7)	Event-driven Nature of WSNs (WP7) Cognitive Effectiveness of BPMN (BP3)
Models should be stable on minor WSN changes (R8)	Modifiability of Process Models (BP5)

Table 3.2: This table shows which requirements are based on which properties.

task, etc. should not affect the model itself. By this way, we will end up with a BPMN process model which conforms to standard BPMN regarding to modifiability of process models.

3.4 Summary

In this section, we evaluated the properties investigated in previous section and created our requirements using the relevant properties. The relevant properties are shown in Table 3.1. Using these properties we created these requirements:

- **Support for Indirect and Dynamic Addressing of Nodes (R1)**: The indirect addressing of nodes and dynamic addressing of nodes need to be supported in BPMN models.
- **Support and Restrict User to WSN Operation Categories (R2)**: The BPMN models should have a means to define: type of the operation (sense, actuate or intermediary operation), location of the operation, operation definition, output nodes for the output data and an operation definition for the output nodes.
- **Limit Available Operations for WSNs (R3)**: The created models should realistic, i.e., the constructs that are not possible in WSNs should not be possible.
- **Support for Multiple Instances of the Same Process (R4)**: The BPMN models should support multiple instances of the same WSN process as it does in standard BPMN processes and similarly WSN processes should be modeled as standard BPMN processes.

- **Distribution of Execution Logic into WSN (R5)**: The modelers should have the ability to define orchestrating nodes of a WSN process.
- **Prioritization of Performance Goals (R6)**: The prioritization of the performance goals should be supported in BPMN models to define application goals, e.g., consume energy.
- **Support for Event-driven Actions in Modeling (R7)**: There should be an explicit separation between event-driven operations and periodic operations.
- **Models should be stable on minor WSN changes (R8)**: Models should be as stable as possible whenever a WSN property changes.

A table of relationships between relevant properties and requirements is shown in Table 3.2. In this table, we have shown which property has been a basis for which requirement.

4 Solution Proposals

To satisfy the requirements presented in Section 3.3, we propose extensions to standard BPMN. These extensions are WSN Task, WSN Pool, and Performance Annotations. These proposals are built on the work by Tranquillini et al. [TSD+12]. We define a WSN Task is an extension of Service Task of standard BPMN, a WSN Pool is an annotated standard BPMN pool and a Performance Annotation is an extension of BPMN groups. In the following sections, we will define these extensions in detail.

4.1 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 an additional antenna marker on the left top corner.

A class diagram of a WSN Task is presented in Figure 4.1. It extends Service Task to provide backward compatibility with standard web services. The instances based on tWSNPerformer are used to address nodes dynamically and indirectly which satisfies the requirement R1. OrchestrationPerformers are used to change the orchestrating nodes of the WSN and by providing that we meet the requirement R5. The outputTarget element stands for the set of nodes to which output data of the executed task is sent. The actionPerformer elements are used to address the nodes which will execute the corresponding task. ActionType and isCommandAction elements are used to define type of the WSN Task. TargetOperation and

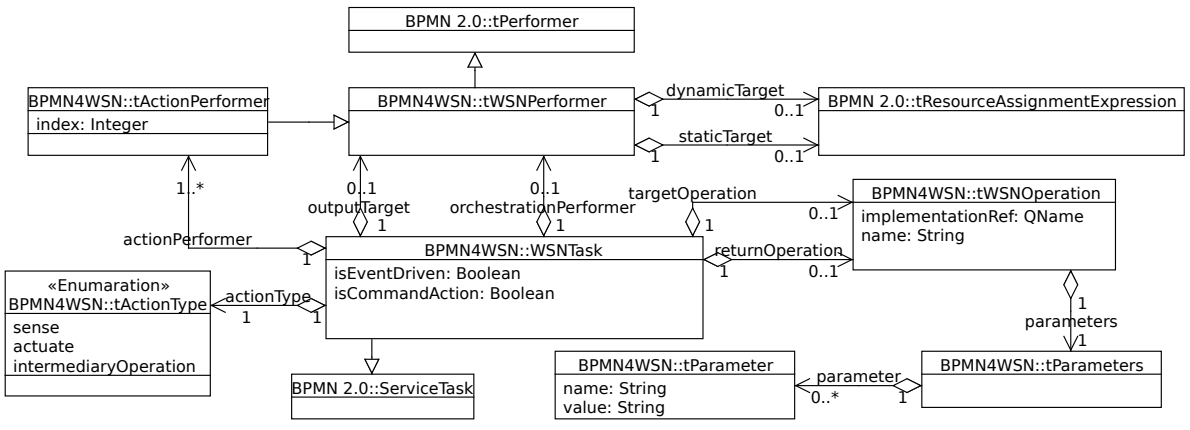


Figure 4.1: WSN Task class diagram.

Listing 4.1 WSN Task XSD definition

```
<xsd:element name="WSNTask" type="tWSNTask"></xsd:element>
<xsd:annotation>
       <xsd:documentation>
              Definition of WSN Task with the element types defined previously.
       </xsd:documentation>
</xsd:annotation>
    <xsd:complexType name="tWSNTask">
       <xsd:sequence>
         <xsd:element ref="actionType" maxOccurs="1" minOccurs="1"></xsd:element>
     <xsd:element ref="isLocalAction" maxOccurs="1" minOccurs="1"></xsd:element>
     <xsd:element ref="targetOperation" maxOccurs="1" minOccurs="0"></xsd:element>
     <xsd:element ref="isEventDriven" maxOccurs="1" minOccurs="1"></xsd:element>
     <xsd:element ref="actionPerformer" maxOccurs="unbounded"
         minOccurs="0"></xsd:element>
     <xsd:element ref="outputTarget" maxOccurs="1" minOccurs="0"></xsd:element>
     <xsd:element ref="returnOperation" maxOccurs="1" minOccurs="0"></xsd:element>
     <xsd:element ref="orchestrationPerformer" maxOccurs="1" minOccurs="0"></xsd:element>
       </xsd:sequence>
   </xsd:complexType>
```

returnOperation are type of tWSNOperation and they are used to reference available WSN operations of underlying WSN. 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 R7. Extension element parameters, which is an instance of tParameters, fulfills the requirement R6 partially and with Performance Annotations (see Section 4.3), we fulfill corresponding requirement completely. Listing 4.1 shows the XSD definition of a WSNTask. In the following sub-sections, we explain the extension elements of a WSN Task in detail.

4.1.1 tWSNOperation

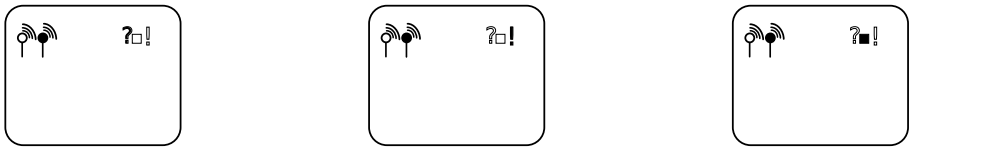
In standard BPMN, the standard operation construct is of type tOperation; however, this construct does not suffice in case of WSN operations. The tWSNOperation type is used to bind a WSN operation to a WSN Task. The corresponding task is expected to execute this referenced WSN operation during the execution. 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. They are used to define the operations that are executed on actionPerformers and outputTargets respectively. Parameters element of type tWSNOperation contains a list of parameters used to pass configuration variables to corresponding operations referenced in WSN Tasks. A parameter is composed of

Listing 4.2 WSNOperation XSD definition

```
<xsd:element name="parameters" type="tParameters"></xsd:element>
<xsd:complexType name="tParameters">
	<xsd:annotation>
		<xsd:documentation>
			Static parameters that will be passed to operations in WSN tasks.
		</xsd:documentation>
	</xsd:annotation>
	<xsd:sequence>
		<xsd:element ref="parameter" maxOccurs="unbounded" minOccurs="0"></xsd:element>
	</xsd:sequence>
</xsd:complexType>

<xsd:element name="parameter" type="tParameter"></xsd:element>
<xsd:complexType name="tParameter">
	<xsd:annotation>
		<xsd:documentation>
			Parameter definition for parameter list. Target operation stands for
			    the operation this parameter refers to.
		</xsd:documentation>
	</xsd:annotation>
	<xsd:attribute name="name" type="xsd:string"></xsd:attribute>
	<xsd:attribute name="value" type="xsd:string"></xsd:attribute>
</xsd:complexType>

<xsd:element name="targetOperation" type="tWSNOperation"></xsd:element>
<xsd:element name="returnOperation" type="tWSNOperation"></xsd:element>
<xsd:element name="WSNOperation" type="tWSNOperation"></xsd:element>
<xsd:complexType name="tWSNOperation">
	<xsd:sequence>
		<xsd:element name="implementationRef" type="xsd:string"></xsd:element>
		<xsd:element name="operationName" type="xsd:string"></xsd:element>
		<xsd:element ref="parameters"></xsd:element>
	</xsd:sequence>
</xsd:complexType>
```



(a) Sense **(b)** Actuate **(c)** Intermediary Operation

Figure 4.2: The WSN Task with different action type representations.

name, value pair to define the parameter. The XSD representation of the tWSNOperation and its instances are shown in Listing 4.2.

Listing 4.3 WSN ActionType XSD definition

```
<xsd:element name="actionType" type="tActionType"></xsd:element>
<xsd:complexType name="tActionType">
   <xsd:annotation>
     <xsd:documentation>
        An action type is either a local action or a command action.
     </xsd:documentation>
   </xsd:annotation>
   <xsd:attribute name="type" type="xsd:string" default="sense"></xsd:attribute>
</xsd:complexType>
```

Listing 4.4 WSN isCommandAction XSD definition

```
<xsd:element name="isCommandAction" type="tIsCommandAction"></xsd:element>
<xsd:simpleType name="tIsCommandAction">
   <xsd:annotation>
     <xsd:documentation>
        An action type is either a local action or a command action.
     </xsd:documentation>
   </xsd:annotation>
   <xsd:restriction base="xsd:boolean"></xsd:restriction>
</xsd:simpleType>
```

4.1.2 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. The XSD representation of the actionType of the WSNTask can be found in Listing 4.3

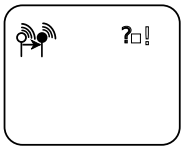
4.1.3 isCommandAction

The extension element isCommandAction is of type Boolean and its default value is false. By setting this element true will define this task as a command action. This would semantically mean that the underlying task execution would involve a command action. The XSD representation of the isCommandAction property of the WSNTask can be found in Listing 4.4. In the following paragraphs, we will describe local and command actions.

LocalAction

This is the default selection for a WSN Task. There is no message exchange is needed to execute the targetOperation as the execution has been initiated and accomplished by the the same nodes.

CommandAction



The command action stands for commanding nodes for executing the action referenced by the targetOperation. There is a communication to initiate a command and multiple commands can be observed in case multiple actionPerformers have been defined. The additional arrow icon represents a command message sent to the nodes which are responsible for execution.

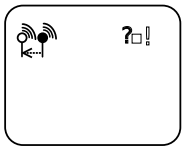
4.1.4 tWSNPerformer

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. 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 used to address WSN nodes before deployment of WSN node binaries. We need to distinguish between these two attributes as it is not possible to derive from a resource assignment expression itself whether it should be evaluated during run-time of the WSN or before deployment time. Technically, both dynamicTarget and staticTarget are of type tResourceAssignmentExpression, which is the BPMN standard type for resource assignments. OutputTarget and OrchestrationPerformer extension elements are instances of the type tWSNPerformer and in the following paragraphs we will explain them. The XSD representation of the tWSNPerformer and its instances can be found in Listing 4.5.

Listing 4.5 WSNPerformer XSD definition

```
<xsd:element name="orchestrationPerformer" type="tWSNPerformer"></xsd:element>
<xsd:element name="outputTarget" type="tWSNPerformer"></xsd:element>
<xsd:element name="WSNPerformer" type="tWSNPerformer"></xsd:element>
<xsd:complexType name="tWSNPerformer">
  <xsd:annotation>
    <xsd:documentation>
      This type is used to define a set of nodes in a WSN. Which has both zero to many static
          attributes and zero or one dynamic attribute.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:complexContent>
    <xsd:extension base="BPMN:tPerformer">
          <xsd:sequence>
            <xsd:element name="staticTarget"
                type="BPMN:tResourceAssignmentExpression"></xsd:element>
            <xsd:element name="dynamicTarget"
                type="BPMN:tResourceAssignmentExpression"></xsd:element>
          </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

outputTarget



The 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. The prerequisite of having a returnOperation is having a valid outputTarget because the nodes which are defined by outputTarget will execute the operation referenced by returnOperation.

orchestrationPerformer

This is an optional element used to change the orchestrating nodes of a WSN process instance. This element is of type tWSNPerformer. If a WSN task is defined with an orchestrationPerformer, the execution of the 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 work and a further reading can be found in [AK04].

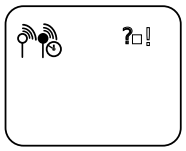
Listing 4.6 WSN isEventDriven XSD definition

```
<xsd:element name="isEventDriven" type="tIsEventDriven"></xsd:element>
<xsd:simpleType name="tIsEventDriven">
  <xsd:annotation>
    <xsd:documentation>
      An annotation in order to differentiate between periodic and event-driven operations.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:restriction base="xsd:boolean"></xsd:restriction>
</xsd:simpleType>
```

actionPerformer

This extension is an instance of tActionPerformer which is a sub-class of tWSNPerformer. Additionally, the tActionPerformer type has an index. The nodes that are supposed to execute the targetOperation are defined by the actionPerformer extension element of the WSN Task. 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. On the other hand, 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.

4.1.5 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. This icon would remind modelers that this task would take time. Whenever, isEventDriven is true, the corresponding task is supposed to block until the expected event happens. Whenever the corresponding event is triggered, the control is released and outgoing sequence flows are activated. The XSD representation of the isEventDriven property can be found in Listing 4.6.

4.2 WSN Pool

WSN Pools are used to define processes which are executed in WSNs. A class diagram of WSN Pool is shown in Figure 4.3. The WSN Pool extends the standard BPMN. 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. In WSN Pools, the use of BPMN constructs is limited as detailed by Stefano et al. [TSD+12]. By use of these WSN pools we provide a clear separation between WSN processes and standard business processes. We can create blueprint of a WSN business

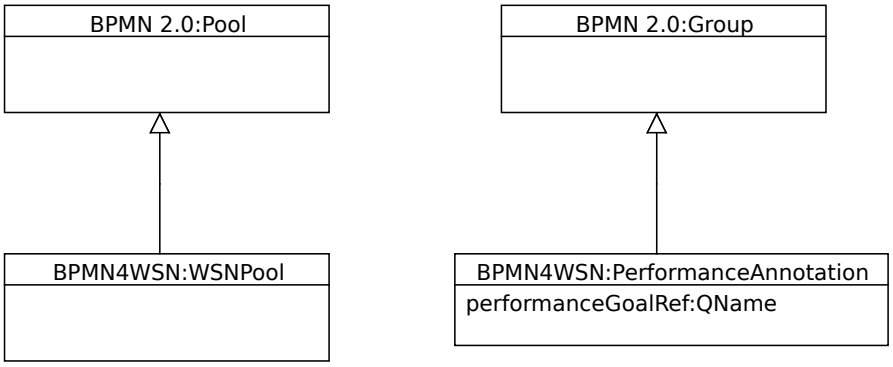


Figure 4.3: WSN Pool and Performance Annotations class diagram.

Listing 4.7 WSNPool XSD definition

```
<xsd:element name="WSNPool" type="tWSNPool"></xsd:element>
<xsd:simpleType name="tWSNPool">
       <xsd:restriction base="xsd:boolean"></xsd:restriction>
</xsd:simpleType>
```

process in WSN pool and multiple instances of it can be executed concurrently. Moreover by restricting use of the unnecessary constructs in WSN pools, we will have a sensible WSN model, i.e., unnecessary BPMN constructs would not exist in WSN pool. By using WSN Pools, we satisfy requirement “Support for Multiple Instances of the Same Process (R4)”. The XSD representation of a WSN Pool can be found in Listing 4.7.

4.3 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 a concrete performance goal. A class diagram of Performance Annotations is shown in Figure 4.3. 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 change the performance of the BPMN constructs that it contains during their execution. In case of 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. By referencing the actual performance goal, we are separating our models from the actual non-functional property definitions and we are providing the possibility of our non-functional model and process model to evolve independently. 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 the requirement “Prioritization of Performance Goals (R6)” completely (It has been partially satisfied by parameters defined

Listing 4.8 Performance Annotation XSD definition

```
<xsd:element name="performanceRef" type="tPerformanceRef"></xsd:element>
<xsd:complexType name="tPerformanceRef">
  <xsd:sequence>
       <xsd:element name="name" type="xsd:string" maxOccurs="1" minOccurs="1"></xsd:element>
  </xsd:sequence>
  <xsd:attribute name="location" type="xsd:QName"></xsd:attribute>
</xsd:complexType>
```

Solution Name	Requirement
WSNTask/tWSNPerformer, tActionPerformer	Support for Indirect and Dynamic Addressing of Nodes (R1)
WSNTask/tWSNOperation	Support and Restrict User to WSN Operation Categories (R2)
WSNTask/actionType	Support and Restrict User to WSN Operation Categories (R2)
WSNTask/isCommandAction	Support and Restrict User to WSN Operation Categories (R2)
WSNTask/actionPerfomer, outputTarget	Support and Restrict User to WSN Operation Categories (R2)
WSN Pool	Limit Available Operations for WSNs (R3) Support for Multiple Instances of the Same Process (R4)
WSNTask/ orchestrationPerformer	Distribution of Execution Logic into WSN (R5)
Performance Annotations	Prioritization of Performance Goals (R6)
WSNTask/ tWSNOperation/parameters	Prioritization of Performance Goals (R6)
WSNTask/isEventDriven	Support for Event-driven Actions in Modeling (R7)
All	Models should be stable on minor WSN changes (R8)

Table 4.1: Solution Requirement Mapping

in tWSNOperation). The XSD representation of a Performance Annotation can be found in Listing 4.8.

The mappings between solution proposals and requirements are shown in Table 4.1. In Figure 4.4, the same business process model as in Figure 6.1 has been drawn with the proposed extensions. XML excerpts of the extended BPMN of the diagram shown in Figure 4.4 is shown in Listing 4.10. This model and BPMN have been created using an extended version of "Signavio Core Components"[1].

[1] http://code.google.com/p/signavio-core-components/

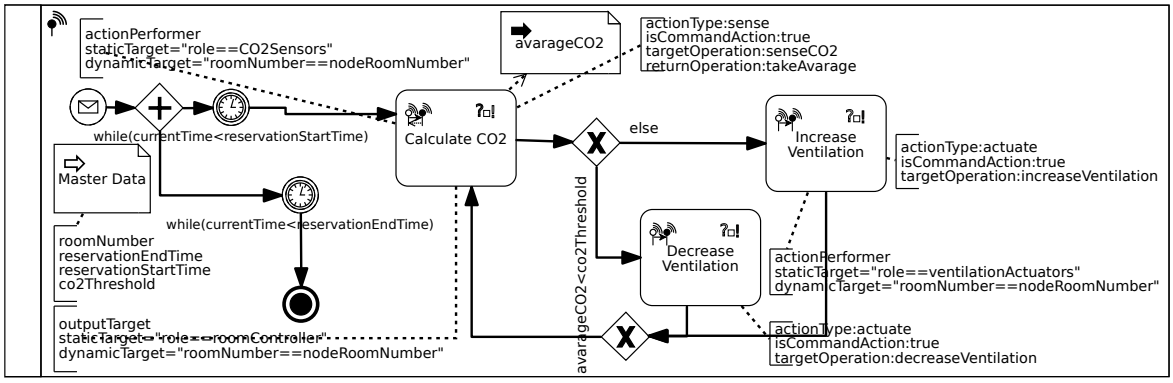


Figure 4.4: A ventilation business process with the extended BPMN.

Listing 4.9 XML excerpts of the extended BPMN model from the Figure 4.4

```
...
<import importType="http://www.w3.org/2001/XMLSchema" location=""
    namespace="http://www.project-makesense.eu/bpmn4wsn"/>
<extension definition="bpmn4wsn:WSNTask" mustUnderstand="true"/>
...
 <process id="sid-20a02571-7300-4298-9c1b-a36cb0856b0d" isClosed="false"
     isExecutable="false" processType="None">
  <extensionElements>
    <bpmn4wsn:WSNPool bpmn4wsn:wsnPool="true"/>
  </extensionElements>
  ...
  <serviceTask completionQuantity="1" id="sid-34182559-A393-41B7-A981-B558987B15CB"
      implementation="webService" isForCompensation="false" name="Calculate CO2"
      startQuantity="1">
    <extensionElements>
     ...
    </extensionElements>
    ...
  </serviceTask>
  ...
```

4.4 Summary

In this chapter we proposed some extensions to BPMN to meet the requirements which have been derived from the relevant BPMN and WSN properties (see Section 3.1 and Section 3.2). These extensions are:

- **WSN Task**: A WSN Task is based on the "Service Task" of standard BPMN. The visual view of a WSN Task is similar to a BPMN task with additional icons on top right corner (see Figure 4.2). A WSN Task corresponds to a task which is executed in a WSN. There are additional properties of WSN Task to its base type and they are:
  - **tWSNOperation**: This type is used to define a WSN operation type. A namespace for the operation defined via implementationRef. In this namespace an operation

Listing 4.10 <extensionElements> of the "Calculate CO2" WSN Task from the extended BPMN model in Figure 4.4

```
<extensionElements>
 <bpmn4wsn:WSNTask>
  <bpmn4wsn:actionType bpmn4wsn:type="sense"/>
  <bpmn4wsn:isCommandAction>true</bpmn4wsn:isCommandAction>
  <bpmn4wsn:targetOperation>
   <bpmn4wsn:implementationRef>
    {http://www.project-makesense.eu/application_domain_model}LocalAction
   </bpmn4wsn:implementationRef>
   <bpmn4wsn:operationName>CO2SensingLocalAction</bpmn4wsn:operationName>
  </bpmn4wsn:targetOperation>
  <bpmn4wsn:isEventDriven>false</bpmn4wsn:isEventDriven>
  <bpmn4wsn:actionPerformer bpmn4wsn:index="0"
      id="sid-f27bd7c1-ee02-4677-8a04-d59ab4d7a942">
   <bpmn4wsn:staticTarget id="sid-4bbd804d-db10-4222-9e1a-8e02e67a6b0d">
    <formalExpression id="sid-30661a3f-d916-408d-b724-5fabee922094">
     node.type==co2Sensor
    </formalExpression>
   </bpmn4wsn:staticTarget>
   <bpmn4wsn:dynamicTarget id="sid-3189609f-29aa-4e4d-b546-bf35ba26cf32">
    <formalExpression id="sid-fdbaf212-bcbf-40f4-8362-50163079b8b0">
     node.currentRoom==MasterData.roomNumber
    </formalExpression>
   </bpmn4wsn:dynamicTarget>
  </bpmn4wsn:actionPerformer>
  <bpmn4wsn:outputTarget id="sid-360064c8-3594-4814-8239-3e3d20ad216d">
   <bpmn4wsn:staticTarget id="sid-4cc7dbca-7f62-41db-8aa7-93ac6db48741">
    <formalExpression id="sid-a44cb4e0-4dc0-4bff-89eb-33118bfdfd8d">
     node.type==roomController
    </formalExpression>
   </bpmn4wsn:staticTarget>
   <bpmn4wsn:dynamicTarget id="sid-1272ed08-9718-46b3-bf58-1ccf6b02f2b9">
    <formalExpression id="sid-88d41cfe-d3c5-468e-9a53-80c518da1904">
     node.currentRoom==MasterData.roomNumber
    </formalExpression>
   </bpmn4wsn:dynamicTarget>
  </bpmn4wsn:outputTarget>
  <bpmn4wsn:returnOperation>
   <bpmn4wsn:implementationRef>
    {http://www.project-makesense.eu/application_domain_model}ConcreteAbstraction
   </bpmn4wsn:implementationRef>
   <bpmn4wsn:operationName>MedianOperator</bpmn4wsn:operationName>
  </bpmn4wsn:returnOperation>
 </bpmn4wsn:WSNTask>
```

defined with its name. There are parameters for each tWSNOperation and these parameters are of type of tParameters. There are two instances of tWSNOperation, a targetOperation and a returnOperation. A targetOperation is the operation executed by the nodes represented by actionPerformer and the latter executed by the group of nodes represented by outputTarget.

- **actionType**: With this property, we can define a WSN operation as sense, actuate, or intermediary operation.
- **isCommandAction**: With this extension property, we define a task as a command action or as a local action. In case a command action is selected an additional arrow is shown, on the WSN Task.
- **tWSNPerformer**: The activity performers in WSNs are defined through tWSNPerformer. There are two expressions defined in the type tWSNPerformer, i.e., staticTarget and dynamicTarget. The staticTarget is evaluated before deployment time whereas the dynamicTarget is evaluated during run-time of the WSN process. The instances that based on tWSNPerformer are:
  - **outputTarget**: The outputTarget represents the group of nodes to which the output data of the targetOperation is sent. An additional return arrow is visible in case the outputTarget is defined.
  - **orchestrationPerformer**: The orchestrationPerformer represents the nodes which orchestrate the WSN process. After it has been defined the following execution flow is orchestrated by the nodes referenced by this orchestrationPerformer. There can exist parallel flows therefor parallel orchestrationPerformers can exist.
  - **actionPerformer**: This is an instance of tActionPerformer. tActionPerformer is a sub-class of tWSNPerformer. Additionally, there is an index variable to represent the ordering of the execution in case multiple command needs to be delegated multiple times.
- **isEventDriven**: This is a variable of Boolean type. True means that this task represents an event-driven task otherwise it represents a periodic task. An additional clock icon is shown on the WSN Task in case it has been set to true.

- **WSN Pool**: WSN processes are created in these pools. By this way we can create WSN processes similar to standard BPMN and we can semantically restrict some constructs in WSN pools. There exists an additional icon on the left top corner of the pool. This construct is inherited from standard BPMN pools.
- **Performance Annotations**: This extension inherits from the group construct of standard BPMN and adds an additional performanceGoalRef property which is used to reference a performance definition. During the execution, the grouped constructs by a "Performance Annotation" are supposed to execute in the performance mode referenced by the performanceGoalRef.

5 Related Work

There are domain specific extensions proposed to BPMN and might be used to model business processes which include WSNs [GZBD11, LKS11, ZSL11, BFV11a]. Although the intended purpose of those extensions are mostly not related with WSNs and do not satisfy our requirements, some might cover some of the requirements as the extensions proposed by Brambill et al. [BFV11a] does.

BPM 4 People[1] is an EU project which aims to include social platforms in business processes. To achieve these goals and model their changes in BPM, they made several extensions in BPMN on top of pools, tasks, gateways and events. With these extensions they provide:

- Actor categorization.
- Visibility of the process status.
- Level of social participation.

The aim of this project is to provide social integration to

- Increase transparency and participation to the decision procedures.
- Exploit weak ties between people and implicit enterprise know-how.
- Involve activities communities in activity execution.

[BFV11c, BFV11a, BFVR12, BFV11b]

The details about BPMN 2.0 extensions provided by BPM 4 People can be found under [BFV11a]. Some related extensions are:

- Community Pools
- Hierarchical Definition of User Roles.
- Publish Task

[1] http://www.bpm4people.org/cms/content/en/home

The extensions on pools are categorization of pools to reflect the corresponding actor categorization, i.e., defines the type of pool and its associated roles. What the performer categorization generally brings is extending the current performers of tasks in BPMN and add dynamic performers to it, i.e., the roles can be assigned after deployment time. This approach is more like a publish/subscribe way of doing things in case of performers, some certain tasks are defined without the knowledge of performers. The performers from external actor pools can participate in business processes by some invitations.

A hierarchical assignment of user roles via lanes have been made possible. A sub-role is assigned to a lane and can perform tasks from super-role and from it's assigned role.

There is an extension based on "Send Task" of standard BPMN which is called Publish, which is used to send messages from a regular pool to social pool. This task has similar operations which can be observed in WSNs, i.e., Broadcast, Multicast, and, Uni-cast. The main difference among these tasks are as their names suggest inviting all users, a group of users or just one user.

With standard BPMN 2.0, BPMN models have their corresponding execution semantics. There are some BPMN vendors who created their execution engines, e.g., Activiti, jBPM, Bonitasoft, etc. Such BPMN vendors provide certain extensions points that they think these extensions will be needed in most of the business cases [Act, jBP12]. There are also some extensions included on some BPMN engines, .e.g, Activiti and jBPM. The list of available BPMN elements including their extensions can be found under [Act, jBP12].

Activiti is an open-source BPM platform which provides a tool set to design and execute business processes and its business engine is based on BPMN. jBPM is a similar tool-set which is offered by jBoss community. They are both run in any java environments and they both provide business experts some extensions that they see it increases usability of BPMN.

In Activiti, there are some assignments to performers of UserTask, which are "candidateUsers" and "candidateGroups". These extensions define potential owners to a task. In jBPM, there are no significant extensions. There is a general additional task name attribute in task and there are "onEntryscript" and "onExitscript" (Which are by assumption the scripts to be run at the beginning and at the end of task).

Standard BPMN can be used with minimal extensions to model WSNs. The extensions possible mentioned are [CK11]:

- A task annotation to differentiate asynchronous WSN tasks.
- For simulation environments some network configuration parameters can be included.

Both of these extensions do not effect the execution semantics of BPMN language and used as meta-data for simulation environments.

Creation of WSN applications are challenging because during development, low-level programming is needed. Application should be scalable and resources are constrained. Web-services is the standard for business process interactions [Obj11] and WSNs do not provide any kind of web-services interfaces and to achieve such a functionality a middle-ware is needed [AIM10]. In case of integration of these WSN application to an enterprise context requires significant

amount of effort and low-level programming; however, what domain experts are interested in are operations offered by WSNs on certain objects not the low level details [TSD+12]. To overcome these programming difficulties, different model-driven code generation techniques with different modeling languages were proposed in the works of [GEPF11, CB11, ADBS09, TSD+12]. Akbal et al.. [ADBS09] use UML to create a model-driven development approach. UML is used to model the architecture of software systems, not business systems [JAC03]. Glombitza el al. [GEPF11] use BPEL to create a model-driven approach to create code from business processes. Only the work of [CB11] and [TSD+12] use the 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 makeSense project as the work of [TSD+12] and solution proposals are built on top of the previous work of [TSD+12].

5.1 Summary

In this chapter, we briefly explained BPM4People extension of standard BPMN which might be a relevant extension to model WSN integrated business processes. The BPM4People is an EU project and it proposes some extensions to standard BPMN to provide social integration. For instance, there are some minor extensions from BPMN execution engines which do not meet our requirements. These extensions are defined to support standard IT systems in a better way.

The low-level modeling during creation of WSN applications is the main obstacle for domain experts. There are multiple model-driven approaches with different modeling and execution languages. There is another approach which uses standard BPMN without any extensions [CK11]. Caracas et al. proposes that standard BPMN can be used to model WSN integrated business processes without any extensions.

6 Evaluation

In this chapter, we discuss benefits of using our extensions and draw-backs of other possible approaches with respect to requirements that we have previously defined (see Chapter 3). The requirements can be satisfied (2), partially satisfied (1) or not satisfied (0). We will evaluate our approach against standard BPMN and BPM4People extensions. BPM4People subsections will appear where they have some related additional extensions otherwise it will be similar to standard BPMN. After a comparison of different approaches with respect to our requirements, we will provide an overall picture of our the results achieved.

6.1 Support for Indirect and Dynamic Addressing of Nodes (R1)

6.1.1 Standard BPMN

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

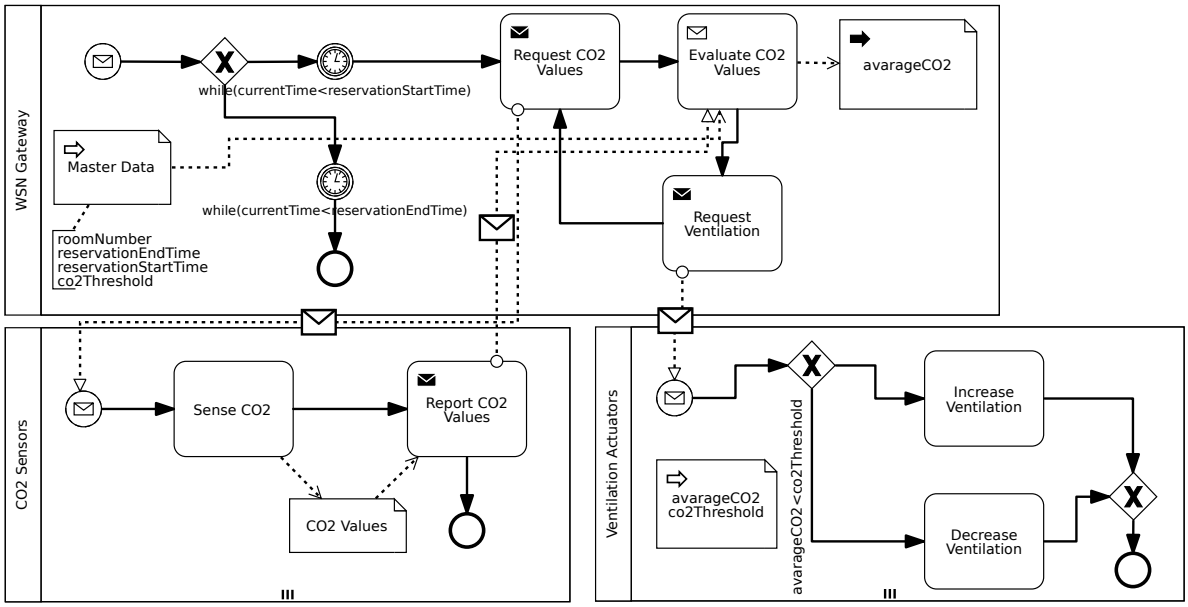


Figure 6.1: A ventilation business process with standard BPMN.

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. The requirement is partially satisfied (1) because one cannot define explicit dynamically evaluated addresses.

6.1.2 BPM4PEOPLE

The hierarchical definition of user roles might be used to address some nodes indirectly, e.g., first sensors in a lane as a super-role, then temperature sensors in another lane as a sub-role and then sensors in "room 1" in another lane as another sub-role, etc. This approach would partially satisfy (1) the requirement because dynamic addressing is not supported.

6.1.3 BPMN4WSN

With the BPMN4WSN extensions, users can define expressions which can address nodes directly, indirectly, statically and dynamically using WSN specific performer elements. This requirement is satisfied (2) by our extensions because dynamic and indirect addressing are provided by our extensions.

6.2 Support and Restrict User to WSN Operation Categories (R2)

6.2.1 Standard BPMN

Standard BPMN does not provide constructs for sense, actuate and intermediary operations because it is tailored to the execution on general multipurpose 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. The requirement is not satisfied (0) by the standard BPMN.

6.2.2 BPM4PEOPLE

The mostly related available operations in BPM4People extensions are the publish tasks. This task stands for sending information from a standard BPMN pool to a social pool. This can be done in three ways uni-cast, multicast and broadcast. These different type of operations do

not fit the operations that we have mentioned in our requirement thus it does not satisfy (0) the requirement as well.

6.2.3 BPMN4WSN

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. By providing these extensions, we satisfy the requirement (2).

6.3 Limit Available Operations for WSNs (R3)

6.3.1 Standard BPMN

In standard BPMN, there can be no restriction on BPMN constructs because there is no explicit separation between standard BPMN pools and WSN pools. This means too much flexibility for modelers and can result in erroneous models. A domain expert can put a "Manual Task" into a WSN business process although it is actually not possible. Thus standard BPMN does not satisfy this requirement (0).

6.3.2 BPM4PEOPLE

There are social pool extensions and one of them could be used instead of a WSN Pool extension to restrict the constructs however this would result in a semantic change of the social pool extension construct because it defines a social pool not a WSN Pool. Moreover, this solution would be only practical in case of non-existence of real social pools. This would partially satisfy our requirement (1).

6.3.3 BPMN4WSN

However, by separating WSN business processes, we provide a means to limit available constructs in those pools. We restrict the construct suggested by Stefano et al. [TSD+12] semantically and this can be easily implemented to modelers for necessary correctness. By providing this extension, we meet this requirement (2).

6.4 Support for Multiple Instances of the Same Process (R4)

6.4.1 Standard BPMN

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. This requirement is partially satisfied (0) by standard BPMN.

6.4.2 BPM4PEOPLE

There are social pool extensions and one of them can be used instead of a WSN Pool extension however this would result in a semantic change of the social pool extension construct because it defines a social pool not a WSN Pool. This solution would not be practical in case of the existence of the real social pools because it would not be possible to distinguish which one is a real social pool and which is a WSN pool. This would partially satisfy our requirement (1).

6.4.3 BPMN4WSN

In case of extensions, we preserve standard BPMN semantics with the WSN Pool. By this extensions element, we satisfy (2) the requirement.

6.5 Distribution of Execution Logic into WSN (R5)

6.5.1 Standard BPMN

There is no explicit decision mechanism in standard BPMN to assign the center of orchestration, as in a WSN, a set of equally powerful nodes collaborate. 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. The orchestration would be done implicitly therefore the requirement is not satisfied (0).

6.5.2 BPMN4WSN

In the proposed extensions, this is done by the orchestrationPerformer element of WSN Tasks thus this requirement has been satisfied.

6.6 Prioritization of Performance Goals (R6)

6.6.1 Standard BPMN

Standard BPMN does not provide a construct to prioritize performance goals. Caracas et al. [CB11] proposed using BPMN the message categories element to define communication protocols between WSN processes. Assuming, different protocols 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. The requirement has been partially satisfied (1) because such explicit constructs in BPMN does not exist.

6.6.2 BPMN4WSN

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. By introducing this element, we satisfy (2) the requirement.

6.7 Support for Event-driven Actions in Modeling (R7)

6.7.1 Standard BPMN

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. [CB11], to distinguish between the synchronous and asynchronous tasks, is to add a textual annotation at the outgoing sequence flows which are 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 requirement is partially satisfied (1) by standard BPMN.

6.7.2 BPMN4WSN

The WSN tasks provide explicit event-driven markings, of which business experts and developers would have a common understanding. By introducing these event-driven markings, we satisfy (1) the requirement.

6.8 Models should be stable on minor WSN changes (R8)

To evaluate the modifiability of a process model created with BPMN4WSN, we will consider the related WSN properties that we presented in Chapter 3 and observe the effects of modifying them. These are the additional modifiable properties on top of standard business processes. We can categorize modifications in two different levels of modeling: At the level of executive modeling (E), i.e., changing properties of existing visual constructs or at the level of analytical modeling (A), i.e., adding and removing visual constructs to reflect the changes. The latter one is a more substantial change than the former one. Another option is: not having any changes in case of some WSN property changes (N). In the following paragraphs, we will evaluate the relevant properties of the WSNs to evaluate the changes on BPMN models.

6.8.1 Dynamic Addition and Removal of Wireless Sensor Nodes (WP1)

Standard BPMN:

These changes in a WSN may cause changes at the level of analytical modeling (A) in case of standard BPMN. If conversation diagrams are used to model the topology of the underlying WSN as proposed by Caracas et al. [CB11], changes in the topology would directly affect these models as well. Moreover, the addition and removal of targets might result in addition and removal of new and existing pools. An example of this can be observed in Figure 6.2. We changed the location of aggregation operation from “WSN Gateway” to “Room Controller”. To apply this change to the existing BPMN model, we need to add a new pool because we created a new WSN node group. After creating this pool, we need to carry the necessary tasks. As you can in Figure 6.2, we also needed to add additional data input/output associations. This would be the case whenever the tasks, which we have carried from “WSN Gateway” to “Room Controller” pool, use some input data from the previous task.

BPMN4WSN:

On the other hand in the extensions we proposed, in case developers have used indirect addressing, changes would not affect at all otherwise users would need change properties at the executable modeling (E).

6.8.2 Categories of WSN Operations (WP2)

Standard BPMN:

Categories of WSN operations are not supported in the standard BPMN, so there won’t be any effects (N) of changing WSN related operation categories, e.g., changing a task from a sense task to an actuate task.

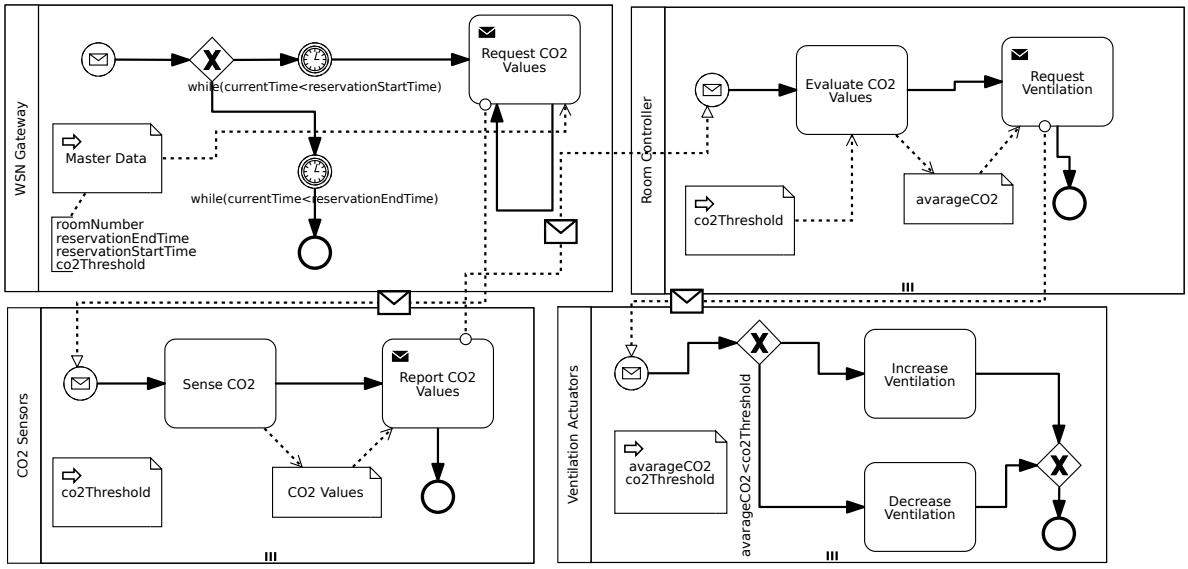


Figure 6.2: The aggregation is now done at the "Room Controller" pool.

BPMN4WSN:

In case of BPMN4WSN extensions, the changes are at level of executable modeling (E) because only properties of the existing constructs need to be changed.

6.8.3 Limited Operations Available in WSNs (WP3)

Standard BPMN:

Because there is no explicit way of defining orchestrating nodes in WSN, this property is irrelevant for the standard BPMN. This would mean no change (N).

BPMN4WSN:

In case of extensions proposed, the changes are at the level of executable modeling (E) because only properties of the existing constructs need to be changed.

6.8.4 Limited Resources and Error-prone Nature of WSN Nodes (WP6)

Standard BPMN:

Standard BPMN does not support performance goals. The communication protocol can be set between different pools and changing this would be at the level executable modeling (E) because only properties of the existing constructs need to be changed.

Property	Standard BPMN	BPM4PEOPLE	BPMN4WSN
Dynamic Addition and Removal of Wireless Sensor Nodes (WP1)	A	A	E
Categories of WSN Operations (WP2)	N	N	E
Limited Operations Available in WSNs (WP3)	N	N	E
Limited Resources and Error-prone Nature of WSN Nodes (WP6)	E	E	E
Event-driven Nature of WSNs (WP7)	E	E	E

Table 6.1: Evaluation of the requirement "Models should be stable on minor WSN changes (R8)".

BPMN4WSN:

On the other hand, when we use BPMN4WSN extensions, we can either change the referenced definitions of performance annotations or the parameters that have been defined in WSN tasks. Thus, in the worst case, it would be at the level of executable modeling (E) because only properties of the existing constructs need to be changed.

6.8.5 Event-driven Nature of WSNs (WP7)

Standard BPMN:

If the standard BPMN was used, the successful completion outgoing sequence flow would be changed. This would result in a change which is at the level of executable modeling (E).

BPMN4WSN:

In case of proposed extensions, the isEventDriven of the corresponding WSN task would be changed. This would result in a change which is at executable modeling (E) because only properties of the existing constructs need to be changed.

6.8.6 Conclusions

A summary of the results for the evaluation of the properties can be found in Table 6.1. In our extensions, we introduce new properties which can be reflected at our business process models therefore the changes on these properties would have effects on the business process models. This result is an expected result and another way is not possible. By introducing

Requirement	Standard BPMN	BPM4PEOPLE	BPMN4WSN
Support for Indirect and Dynamic Addressing of Nodes (R1)	1	1	2
Support and Restrict User to WSN Operation Categories (R2)	0	0	2
Limit Available Operations for WSNs (R3)	0	1	2
Support for Multiple Instances of the Same Process (R4)	1	1	2
Distribution of Execution Logic into WSN (R5)	0	0	2
Prioritization of Performance Goals (R6)	1	1	2
Support for Event-driven Actions in Modeling (R7)	1	1	2
Models should be stable on minor WSN changes (R8)	1	1	1
Sum	**5**	**6**	**15**

Table 6.2: Evaluation of different approaches with requirements.

our extensions, we carry modifiability of a property from the level of analytical modeling to the level of executable modeling which means that one of our solution proposals break the connection between a visual construct (Pool) and carry it to a property of a visual construct (see tWSNPerformer). By doing so we decrease the degree of modifications in case of changes in the property "Support for Indirect and Dynamic Addressing of Nodes (R1)". The two properties which changed from no modifications needed to modifications at a executable modeling are expected changes because we start representing those two properties in our process models. However still this means a decrease in modifiability. In conclusion, we say that we partially satisfy this requirement as standard BPMN does because there is no way to measure the change that we provided with our extensions.

6.9 Interpretation of the Comparison

The overall list of the requirements and their satisfactions with respect to different approaches can be observed in Table 6.2. Standard BPMN does not satisfy any requirements completely however it satisfies 5 requirements partially. BPM4PEOPLE is similar to standard BPMN except at the requirements "Support for Multiple Instances of the Same Process (R4)" and "Limit Available Operations for WSNs (R3)" which it partially satisfies by use of social pool extensions. By having social pools, BPM4PEOPLE social pools provide a better suitability

than standard BPMN because one can capture a whole process in one pool. However it fails whenever there are real social pools and moreover we redefine the definition of social pools in case we use them to represent WSNs which is not a good approach. With the changes that we introduce, we concluded that standard BPMN and BPMN4WSN satisfy the requirement "Models should be stable on minor WSN changes (R8)" partially. This means we did not bring any significant change in the modifiability of standard BPMN by introducing a set of new properties which cover WSN properties comprehensively. Other than R8, we have all requirements satisfied by BPMN4WSN extensions.

6.10 Classification of BPMN4WSN Extensions

In the work of Kopp et al. [KGK+11], a classification for BPEL extensions have been done. This classification can be similarly applied to BPMN4WSN extensions. In the following paragraphs, we will classify BPMN4WSN extensions with respect criterion defined in [KGK+11].

Standard BPMN defines the method how the extensions should be defined and we followed this methodology during definition of BPMN4WSN extensions. We preserve the existing look and feel of standard BPMN and its existing semantics. Thus BPMN4WSN extensions are conformant with standard BPMN.

By introducing new modeling constructs, we extend the modeling (M) of BPMN. BPMN4WSN extensions propose a new type of pool which is converted to binaries and deployed on the WSN nodes. WSNs are the execution engines because the execute WSN processes on WSNs. However in case of collaborations, the other process participants need to communicate with these WSNs and this is not done via Web-services. Therefore there is a need of update on the navigator for communicating with WSN process participants. Apart from this a new navigator is created from scratch for each WSN because for each WSN process new binaries are created. In summary navigator needs to be updated and this means: BPMN4WSN is a run-time (R) extension.

The purpose of BPMN4WSN extensions are functionality (C1.3), maintainability (C1.4), robustness (C1.7) and performance (C1.5). With BPMN4WSN extensions, we add new functionality to our models, i.e., new models can model business processes that include WSNs comprehensively. During our BPMN4WSN extensions, we aimed to keep all properties at the level of executable modeling thus we reduce the modifications on visual level. Less modifications on a visual level would result easier maintainability. Robustness and performance purposes have been provided by Performance Annotations and parameters defined in tWSNOperation.

As subject criteria, control flow (C2.1), service binding (C2.6) and service invocations (C2.7) have been affected. Control flow has been affected by event-driven and periodic (see Subsection 4.1.5) task definitions. Service binding and invocations have been affected because WSN processes are not executed in a service-oriented fashion.

IT infrastructure (C3.1) and Organization (C3.3) dimensions have been affected because we defined a new operation (see Subsection 4.1.1) type and a new performer (see Subsection 4.1.4) type with our extensions.

BPMN4WSN extensions are used during Modeling (C4.1), IT refinement (C4.2), Deployment (C4.4) and Execution (C4.5) phases of the BPM life-cycle. New visual constructs are used to model WSNs. Execution details are added to WSNs and with the help of these variables deployable binaries are created. During execution these some extension attributes are used.

In summary, BPMN4WSN extensions conform with standard BPMN. They extend modeling and run-time mechanism of standard BPMN. They have been created with the purposes of functionality, maintainability, robustness and performance. BPMN4WSN affects control flow, service binding and invocation. IT infrastructure and Organization have been affected by BPMN4WSN extensions. BPMN4WSN extensions are used during Modeling, IT refinement, Deployment and Execution phases of the BPM life-cycle.

6.11 Summary

In this chapter, we compared our extensions with other possible approaches. These approaches are “Standard BPMN” and “BPM4People” extensions.

- **Standard BPMN**: Standard BPMN fails to satisfy most of the requirements. It partially satisfies the R1, R4, R6, R7 and R8. It does not satisfy the requirements R2, R3 and R5.
- **BPM4People**: R1 can be partially satisfied differently by hierarchical role assignments. Moreover, these extensions partially satisfy R3 and R4 because it provides a special type of pool in addition to the standard BPMN pool. The satisfaction of the other requirements are similar to standard BPMN because these extensions do not provide any suitable construct for satisfying the others.
- **BPMN4WSN**: Our extensions fully satisfy almost all the requirements. Only R8 is partially satisfied as standard BPMN does. The partial satisfaction of R8 is an expected thing because we add new properties and such affects on the modifiability of models is an expected.

In the work of Kopp et al. [KGK+11], a classification for BPEL extensions have been done. This classification can be similarly applied to BPMN4WSN extensions: BPMN4WSN extensions conform with standard BPMN. They extend modeling (M) and run-time (R) mechanism of standard BPMN. They have been created with the purposes of functionality (C1.3), maintainability (C1.4), robustness (C1.7) and performance (C1.5). BPMN4WSN affects control flow (C2.1), service binding (C2.6) and invocation (C2.7). IT infrastructure (C3.1) and Organization (C3.3) have been affected by BPMN4WSN extensions. BPMN4WSN extensions are used during Modeling (C4.1), IT refinement (C4.2), Deployment (C4.4) and Execution (C4.5) phases of the BPM life-cycle.

7 Architecture and Implementation

In this chapter, we will give information about the prototypical implementation of the extensions that we proposed. These extensions are implemented on an existing browser-based BPMN modeling tool which is called "Signavio Core Components" (SCC). SCC is based on Oryx [DOW08] editor, which is detailed in the work of Zouaoui [Zou12]. In the following sections, first, we will describe the architecture of "Signavio Core Components" (SCC) and how we placed our extensions and in the following section, we will give details of our implementation.

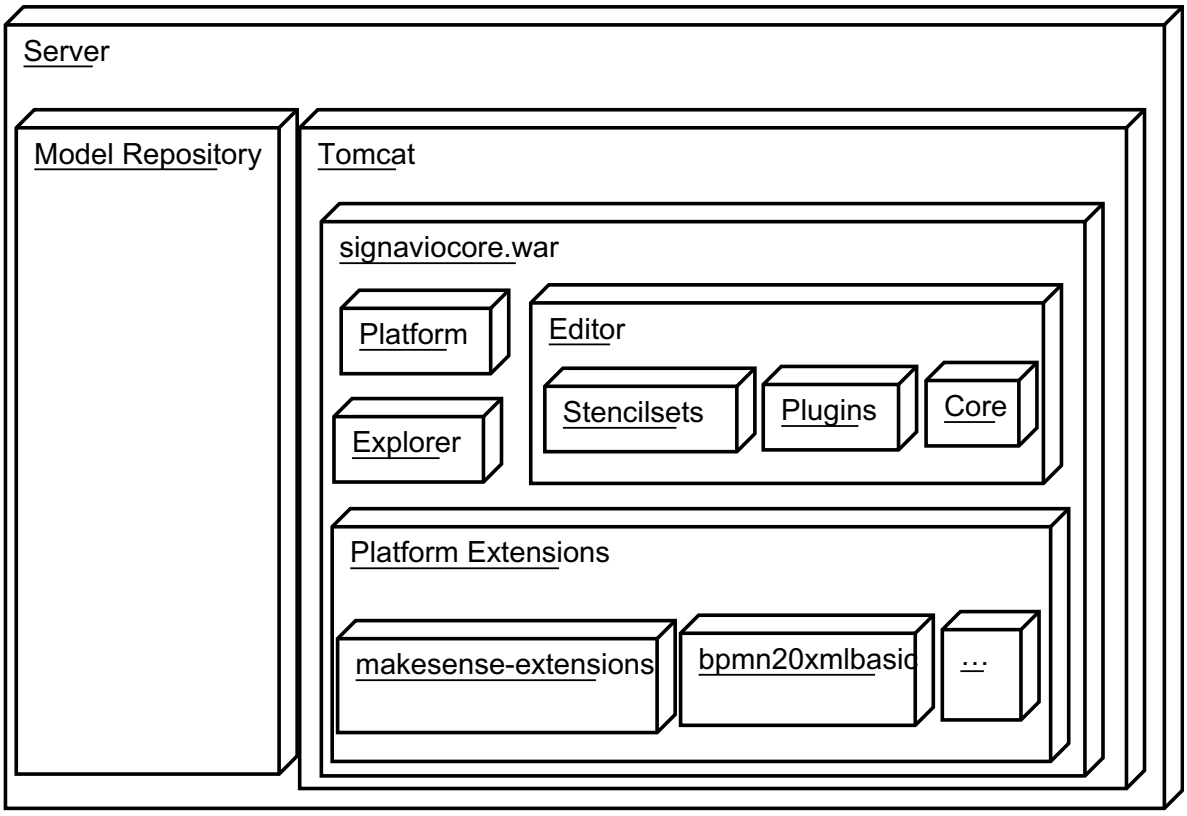


Figure 7.1: Deployment diagram of "Signavio Core Components".

7.1 Architecture

From an architectural perspective, the architecture of SCC is similar to the Oryx which can be found in the work of Zouaoui [Zou12]. A deployment model of the SCC can be found in Figure 7.1. The model repository contains BPMN files of the models and corresponding model meta-data. The location of the repository defined in the build file used during the run time. The build is done using ant[1]. The war file contains different components of the application. The SCC back-end is developed in Java and the user interface is developed in JavaScript and is accessible with modern browsers. The application depends on JavaScript and in case of a no JavaScript support, application cannot be run on a browser. For the communication between server and browser Servlets and Ajax calls have been used.

7.1.1 Signavio Core Components

In the following sub-sections, we will describe each node contained in Figure 7.1.

Explorer

Explorer is a package with JavaScript files and icons. It shows the available modeling tools and navigation in the repository. Users can delete and create new files using the explorer. Nested folders are possible and only valid modeling files will be shown. The design of the explorer.js is made in an object-oriented way. It uses the stensilsets to get the available modeling tools and presents the user available options. There were no updates on the editor during our implementation. Only the name of the model is updated indirectly through stencilset because we updated the type of new models as “BPMN model with BPMN4Extensions”. Stencilsets are the model meta-data in JSON format. They contain nodes, edges and rules for containment, connection, morphing, etc.

Editor

Editor composed of core JavaScript files and plug-in files. Users can customize the editor through the plug-in files. The addition of new buttons are also done through the plug-ins, one needs to offer a new plug-in to facade with necessary onClick function, a text, an icon which will be used to create the button in the toolbar. The buttons can be grouped in the toolbar. Additional plug-in JavaScript files are added to plugins.xml. By doing so they are included during the creation of editor HTML of application and added to deployable war file.

The editor is based on latest public licensed version (v2.0.2) of ExtJS framework and Prototype framework.

[1] http://ant.apache.org/

Editor was developed using Gang of Four (GoF) patterns in an object-oriented way. Some examples of the used GoF patterns are: composite, command and observer patterns. The necessary changes can be done through adding new plug-ins. At first editor HTML is requested from the back-end and it returns an HTML page with links to JavaScript files. After JavaScript files are loaded, the application starts by requesting plugins.xml, and continues with requesting model file. From the model file it checks for the stencilset which is used to create the model. Thereafter editor requests the corresponding stencilset file of the model. The stencilset file can contain nodes, edges and rules. Edges are the connecting objects of the nodes. The rules define which nodes can contain which nodes and which nodes can be connected with which edges and with which nodes. From the stencilset the palette is created. From this palette, users can add the nodes and edges, which have been defined in the stencilsets of the editor, aligned with the rules defined in the stencilset. Every node and edge can have properties, these properties are edited through properties panel. All of these information is saved in the repository when the user clicks save button. Editor has most of the definitions in English and German and by adding necessary fields, new languages can be defined similarly.

Platform

This module is responsible for handling browser requests. Editor and explorer requests are handled by this module. Platform contains the core back-end packages of the SCC. The repository management is done by this component. The packages in this module is developed by Signavio[2] and as extensions some packages from Oryx were taken.

Platform Extensions

This module includes the packages used to provide additional functionality which is not provided by SCC core libraries, i.e., “Platform” module. Some libraries for conversion from model files to BPMN files exist.

7.1.2 Extension Mechanism of SCC

One can define a new modeling elements or can extend an existing one. To create a new set of modeling elements with a completely new visual elements, one needs to create a new stencilset and define stencils, icons, views, properties, etc. of the created stencilsets. After creating this, stencilsets.json needs to be updated and the new stencilset should be referenced here. Explorer would check stencilsets.json file and show the available options to the user. When user creates a new model with the newly defined stencilset, stencils from the stencilset are loaded to the editor.

[2] www.signavio.com/

Another option is to extend existing set of modeling elements. This can achieved by defining an extension stencilset and adding this extension to extensions.json. These extensions will be added after the main stencilset has been loaded.

These two methods are limited by the existing types defined in SCC. The main types can be listed as:

- String
- Text
- Choice
- Complex

Additional program logic can be defined in platform-extensions and afterwards these files can be put into application as a library. This can be similarly achieved for the editor part by defining new files in plugins folder of the editor and then linking them via plugins.xml.

7.1.3 Application Flow

An activity diagram of the modeling can be observed in Figure 7.2. There are two system actors who accomplish these activities. Some activities on the browser are triggered by the user. The flow activities start whenever user opens the application web-site, e,g., default on the local machine `localhost:8080/signavio/p/explorer`. "/p" is the pattern for using the Servlets instead of resources residing on the server. Browser sends a request for the explorer and it is handled by the explorer handler. Afterwards the user can create a new model or open or delete an existing model. If user opens or creates a new model, first the editor HTML is returned from the server. This HTML page includes core and plugin JavaScript files of the editor. The application resided on the browser first requests the model file, hereafter based on the model file, the corresponding stencilset is requested. The requested stencilset is used to create the palette. Afterwards extensions.json is requested to check if there are some stencilsets associated with the stencilset of the model. The stencilsets are identified by a URI. All the extensions associated with the current stencilset is requested and added to palette. Modeler can edit the model and save it to make the changes permanent.

7.1.4 Updated SCC with the Extensions

Our extensions had two parts as SCC, i.e., a user interface part typed in JavaScript and a back-end part typed in Java. The JavaScript files of our extensions resided in the plugins part of the editor. The Java files were contained in the "platform-extensions" component and it is converted to a .jar lib. We extended standard BPMN by introducing some new properties to the existing visual constructs and introducing some new visual constructs. We added these new properties and visual constructs to an extensions stencilset.

An updated activity diagram is shown in Figure 7.3. This is how BPMN modeling is done in makeSene project. At first some well-defined configuration files are uploaded to the system and

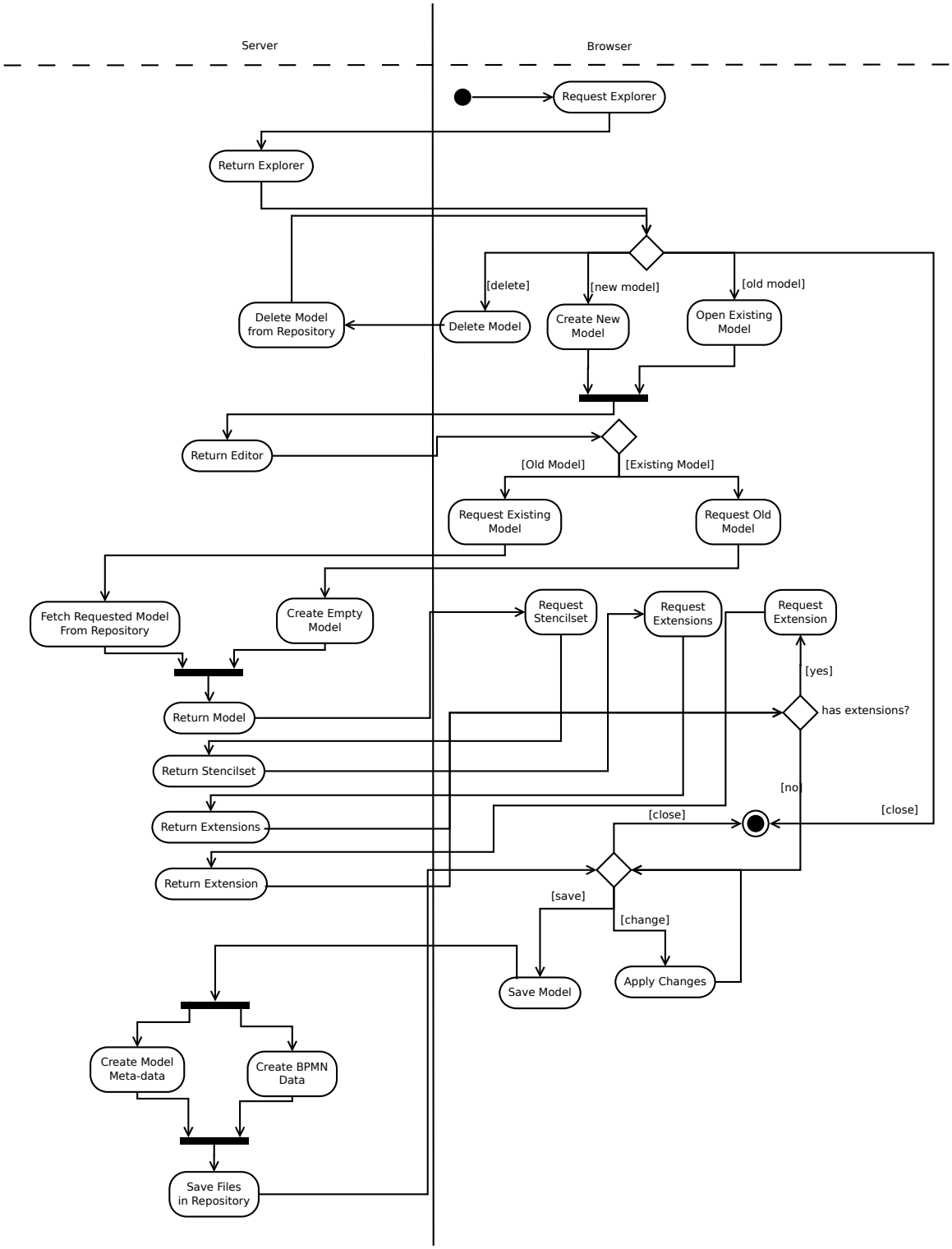


Figure 7.2: Activity diagram of modeling with "Signavio Core Components".

SCC application is started by selecting one of the configuration files. These configuration files provide information related to the WSN, e.g., operations provided by the WSN, the message types that are used to communicate with outside world, etc. User selects a setting that s/he wants to design for and during BPMN modeling the information in the configuration files are made available to the related BPMN constructs. By providing these configuration files, users will create realistic models and the models will be easier to create because these configurations represent the capabilities of the underlying WSN.

7.2 Implementation

During implementation, we firstly created our XSD files for the extensions and then converted them to Java classes using JAXB reference implementation found in http://jaxb.java.net/. We used Eclipse Juno EE[3] version and for JavaScript debugging we used Firefox Firebug plugin[4]. The JAXB classes are used for marshalling and unmarshalling from Java objects to BPMN XML file. To provide the information available in configurations, we created our extension stencilset dynamically. The dynamic creation is achieved by JSP and tag libraries implementation provided by Apache. In the WSN extensions of the SCC, there were 4.468 lines of code including automatically generated JAXB classes. Additionally, there were 2 JavaScript plug-ins which were in total 530 lines of code. We have conducted the code analysis using Sonar[5]. Moreover, there were changes and additions in application logic of the existing component. In the following subsections, we will describe each solution proposal one by one.

7.2.1 WSN Task

As described a WSN operation defined with an operationRef, an operation name and parameters. In the SCC editor, we have 4 properties to define a target operation and return operation. The TargetOperationRef and TargetOperationName are used to define targetOperation of the WSNTask. Similarly, ReturnOperationRef and ReturnOperationName are the used to define returnOperation of the WSNTask. The operation names and namespace are fetched from configuration files automatically and offered to the user for selection.

actionType

This is a choice item and each choice has a visual mapping. The choices are sense, actuate and intermediary operation as detailed in Section 4.1.2. Whenever user changes action type from one to another, the icon on the Task changes accordingly. In stencilsets of the modeling constructs, one can define visual references to the properties. We used this referencing method of the SCC and implemented our type of actions similarly.

[3] http://www.eclipse.org/juno/
[4] https://addons.mozilla.org/en-US/firefox/addon/firebug/
[5] http://www.sonarsource.org/

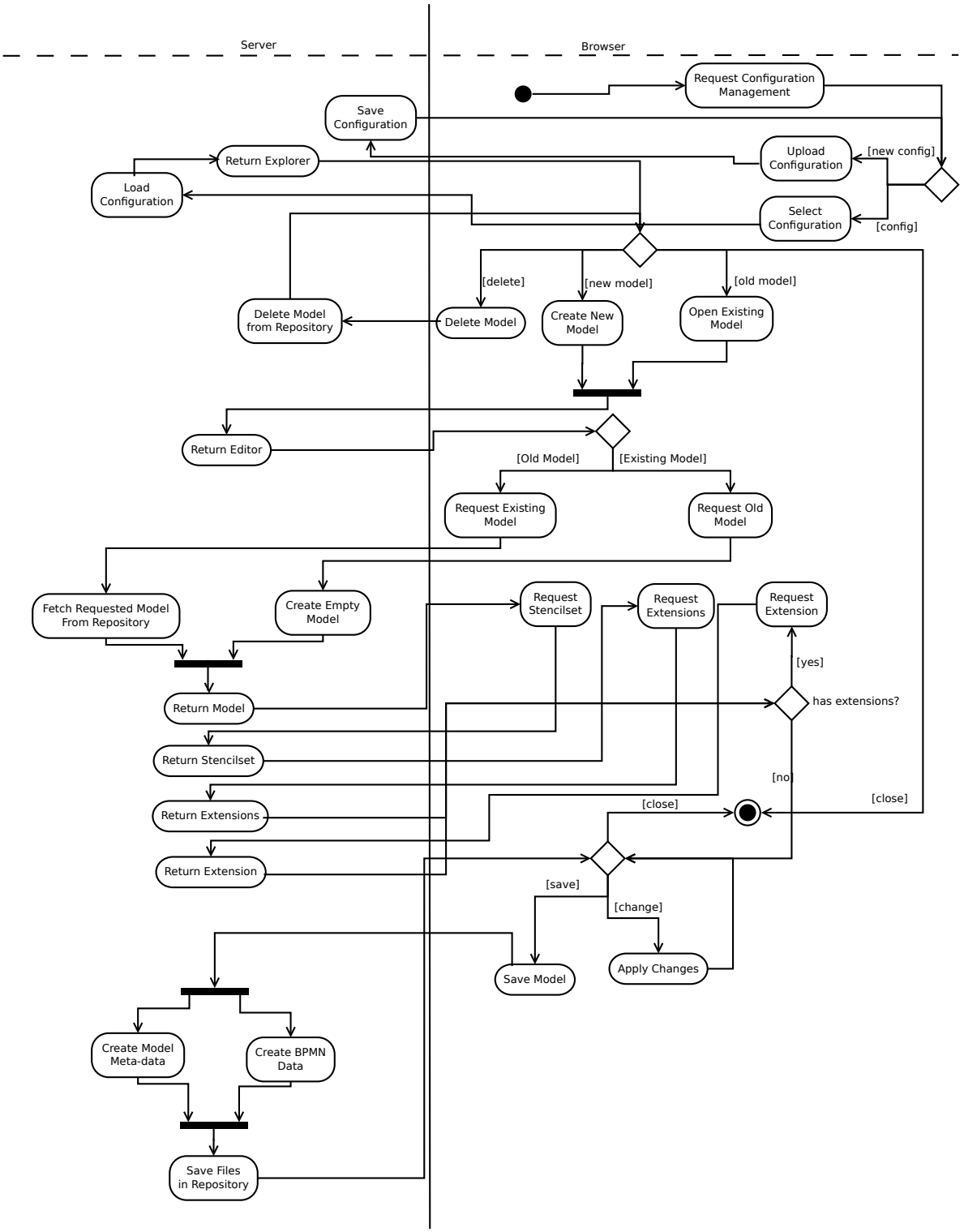


Figure 7.3: Activity diagram of modeling WSN processes in makeSense.

isCommandAction

This is a Boolean box and and if it is true a new arrow is visible on the WSN Task. Arrows on the nodes were not well supported and additional updates on the JavaScript code was needed, i.e., markers on the SVG objects.

tWSNPerformer

To define tWSNPerformer type and its instances properly, we defined a new type in SCC editor because the existing types did not suffice our needs. To define this, we implemented a new editor similar to "Complex Type" editor but it was more of an ad-hoc type of solution. Whenever user wants to add a new WSN Performer, s/he clicks on the WSN Performers property button and it triggers a new window. In this window, user can add zero-to many actionPerformers, zero or one orhestrationPerformer and outputTarget. Each defined WSN Performer can have its static target and dynamic target. These targets are type of tResourceAssignmentExpression and necessary fields have been inherited to define them. These fields can be summarized as expression language, evaluates to type, and expression. In case an outputTarget has been defined, a return arrow visible on the WSN Task.

isEventDriven

This is a property of type of Boolean which had been already defined in SCC previously. This property has a visual mapping and in case it is true, i.e., it is ticked, a clock is visible on the WSNTask.

7.2.2 WSN Pool

We have created a new pool as WSN Pool in the palette. This different pool has an antenna on the left top corner. To define this pool, we have added it to the extension stencilset. However just adding the pool was not enough because the automatic addition of a lane to the pool was hard coded and updates on that was needed to have a similar pool behaviour.

7.2.3 Performance Annotations

The grouping in SCC was buggy, the necessary rules were needed to be updated to define a proper extension of grouping. WSNPools and Performance Annotations were grouped in the palette under WSN Extensions group.

All the extensions are inherited from already existing constructs of BPMN unlike the work of Zouaoui et al. [Zou12]. The containment, connection and morphing rules of these constructs are the same as parent constructs in the stencilset definitions. A WSNPool has the same rules as a standard BPMN pool, a WSNTask has the same rules as a standard BPMN task and finally

a Performance Annotation has the same rules as a group construct of the standard BPMN. Therefore, we reused the rules and roles defined in the stencilset of the parent constructs in our extension stencilset.

7.3 Summary

This chapter provides information about a modeling tool on which we implemented BPMN4WSN extensions. Our extensions were implemented on "Signavio Core Components" (SCC). This is web-based editor and contained in a Servlet container such as Apache Tomcat[6] or JBoss[7]. It provides a web interface which has been developed in JavaScript and the back-end has been developed in Java. The server browser communication are done with Servlets and with Ajax calls on the browser. There exists stencilsets which define models in editors.

Editor has an extension mechanism through stencilsets. An extension stencilset can be created to extend an existing one. Our extensions were defined as an extension stencilset to existing BPMN stencilset. However the standard types defined in SCC were not enough and we needed to define our new type. For changes in the application logic of the editor, editor has been developed with a plug-in mechanism. To add the new type tWSNPerformer, we added new 2 new plug-ins, one for the new type and one for editor for the new type.

[6] http://tomcat.apache.org/
[7] http://www.jboss.org/

8 Summary and Outlook

In this chapter, we will give a general summary or the thesis work and provide information about the place of our approach in model-driven chain.

8.1 Summary

This master thesis focused on modeling WSNs with BPMN and proposes some extensions so that the extended BPMN supports business processes, which include WSNs, comprehensively. BPMN modeling is a step in a model-driven methodology. This methodology includes the creation of binaries from BPMN models which are deployed on WSN nodes later on for execution of BPMN process. This model-driven way of creating binaries from business process models is the current state of art. However our meta-model extensions also support the standard web-services for modeling. A summary of the master thesis is shown in Figure 8.1.

During our literature review, we researched the WSN properties and BPMN properties (see chapter2). First we checked general WSN properties and later on evaluated their relevance in our BPMN modeling concept. The BPMN properties that we found were used while creating the extensions or creating the requirements. The look and feel and existing semantics of the BPMN have been preserved.

After determining the properties, we evaluated them for their relevance with BPMN modeling requirements (see Section 3.1 and Section 3.2). The relevant properties were used to create requirements (see Section 3.3). These requirements were used to create our solution proposals and to compare it with other approaches.

Based on our requirements, we created our BPMN4WSN extensions (see Chapter 4). Based on the related work (see Chapter 5), we selected other approaches which can be used to model WSNs.

We compared our approach with other proposed approaches and detailed which requirements are met by which approaches (see chapter 6). With BPMN4WSN extensions, we almost completely satisfy every requirement whereas standard BPMN and BPM4People extensions fail to fulfill most of the requirements.

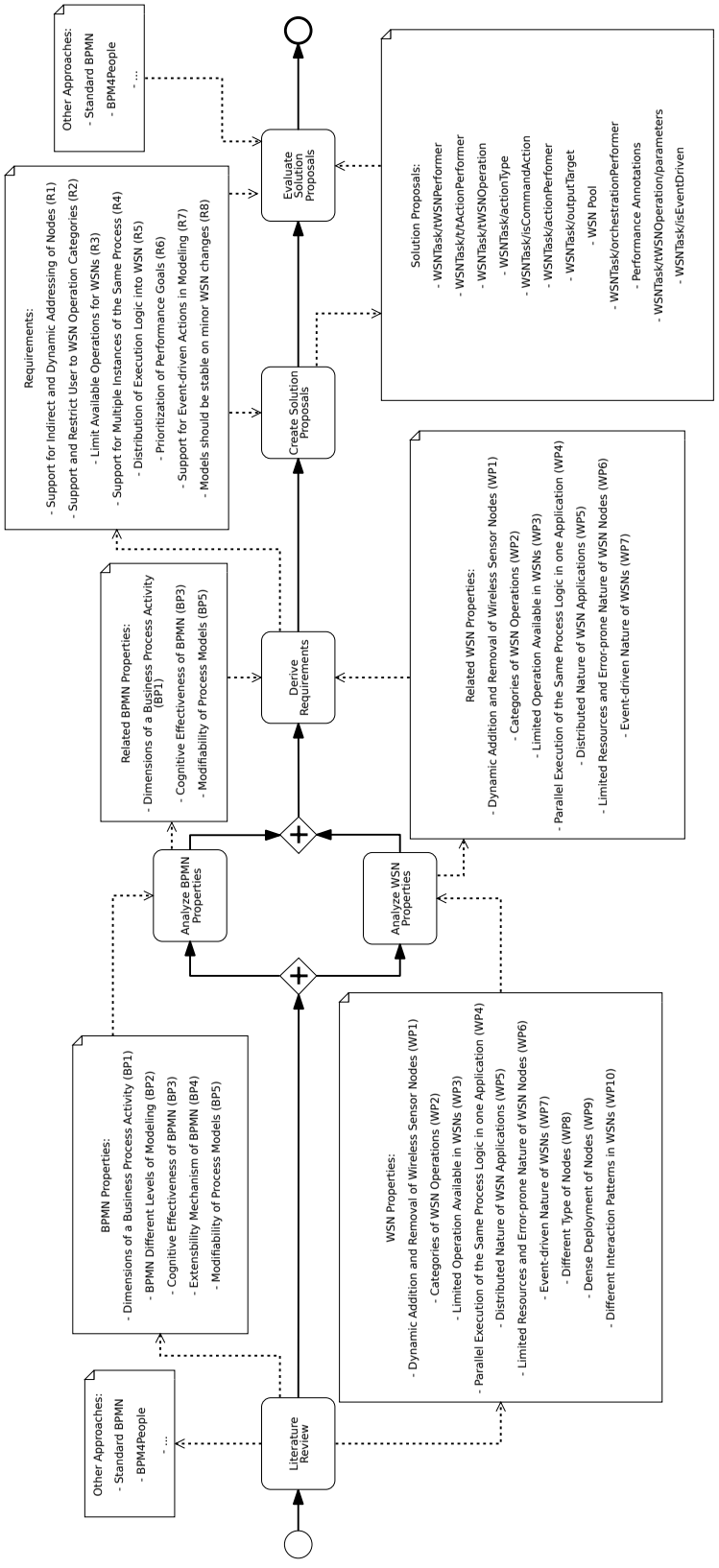


Figure 8.1: Thesis methodology model with results.

8.2 Outlook

This thesis work has concentrated on the BPMN modeling side of the model-driven chain as shown in Figure 8.2. These models have the capability to reference the WSN related data. As a further step, current model compiler needs to be updated and support for the extended BPMN needs to be added.

Later on a comparison of the different approaches can be done. The effort from creating models to generating binaries can be analyzed and the run-time statistics of different approaches can be analyzed.

During execution of collaborations, communicating with external processes would be needed. These external participants would be possibly executing their business processes on business process execution engines. These business execution engines need to support communication

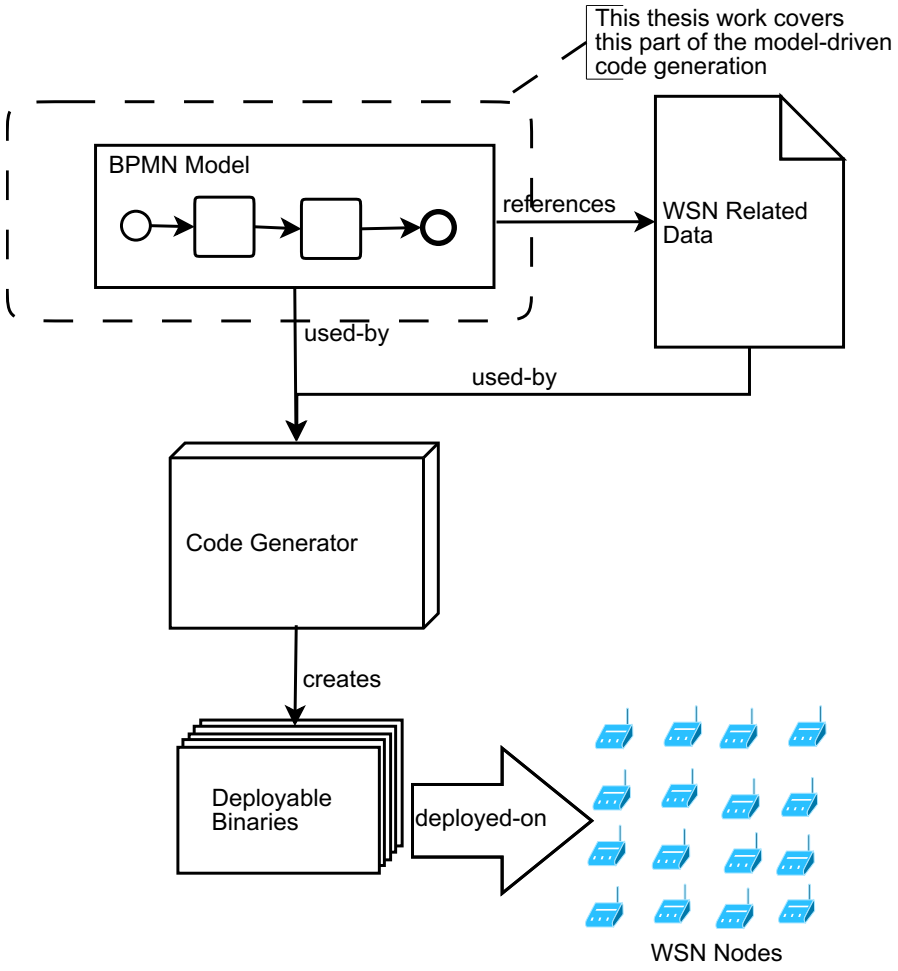


Figure 8.2: The place of this thesis work in the model-driven chain.

with WSNs. Therefore collaborations including WSNs need to be further analyzed and implemented.

Bibliography

[Act] Activiti 5.9 User Guide. URL http://activiti.org/userguide/index.html#bpmnCustomExtensions. (Cited on pages 23 and 52)

[ADBS09] B. Akbal-Delibas, P. Boonma, J. Suzuki. Extensible and Precise Modeling for Wireless Sensor Networks. In *UNISCON*, pp. 551–562. 2009. (Cited on pages 14, 15 and 53)

[AIM10] L. Atzori, A. Iera, G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010. doi:10.1016/j.comnet.2010.05.010. (Cited on pages 11 and 52)

[AK04] I. Akyildiz, I. Kasimoglu. Wireless sensor and actor networks: research challenges. *Ad Hoc Networks*, 2(4):351–367, 2004. doi:10.1016/j.adhoc.2004.04.003. (Cited on pages 20, 21 and 44)

[ASSC02a] I. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102 – 114, 2002. doi:10.1109/MCOM.2002.1024422. (Cited on pages 11 and 13)

[ASSC02b] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002. (Cited on pages 11, 12, 19, 20, 21 and 22)

[BCDV09] C. Buratti, A. Conti, D. Dardari, R. Verdone. An Overview on Wireless Sensor Networks Technology and Evolution. *Sensors*, 9(9):6869–6896, 2009. doi:10.3390/s90906869. (Cited on page 14)

[BFV11a] M. Brambilla, P. Fraternali, C. Vaca. BPMN and Design Patterns for Engineering Social BPM Solutions. In F. Daniel, K. Barkaoui, S. Dustdar, editors, *Business Process Management Workshops (1)*, volume 99 of *Lecture Notes in Business Information Processing*, pp. 219–230. Springer, 2011. (Cited on page 51)

[BFV11b] M. Brambilla, P. Fraternali, C. Vaca. A model-driven approach to social BPM applications. *Social BPM*, pp. 95–112, 2011. (Cited on page 51)

[BFV11c] M. Brambilla, P. Fraternali, C. Vaca. A Notation for Supporting Social Business Process Modeling. In *BPMN*, pp. 88–102. 2011. (Cited on page 51)

[BFVR12] M. Brambilla, P. Fraternali, C. K. Vaca Ruiz. Combining social web and BPM for improving enterprise performances: the BPM4People approach to social BPM. In *Proceedings of the 21st international conference companion on World Wide*

Web, WWW '12 Companion, pp. 223–226. ACM, New York, NY, USA, 2012. doi:10.1145/2187980.2188014. (Cited on page 51)

[CB11] A. Caracas, A. Bernauer. Compiling business process models for sensor networks. In *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*, pp. 1 –8. 2011. doi:10.1109/DCOSS.2011.5982159. (Cited on pages 11, 21, 53, 59 and 60)

[CK11] A. Caracas, T. Kramp. On the Expressiveness of BPMN for Modeling Wireless Sensor Networks Applications. In *BPMN*, pp. 16–30. 2011. (Cited on pages 52 and 53)

[DOW08] G. Decker, H. Overdick, M. Weske. Oryx – An Open Modeling Platform for the BPM Community. In *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, volume 5240 of *Lecture Notes in Computer Science*, pp. 382–385. Springer, 2008. doi:10.1007/978-3-540-85758-7_29. (Cited on page 67)

[EBSK10] E. M. K. K. Er. Barjinder Singh Kaler. Challenges in Wireless Sensor Networks. In *ISCET*. 2010. (Cited on page 14)

[EPKG12] E. B. R. Edwin Prem Kumar Gilbert, Baskaran Kaliaperumal. Research Issues in Wireless Sensor Network Applications: A Survey. *International Journal of Information and Electronics Engineering*, 2:702–706, 2012. (Cited on page 12)

[Fri11] P. Friess. *Internet of Things - Global Technological and Societal Trends From Smart Environments and Spaces to Green ICT*. River Publishers, 2011. (Cited on page 11)

[GEPF11] N. Glombitza, S. Ebers, D. Pfisterer, S. Fischer. Using BPEL to Realize Business Processes for an Internet of Things. In H. Frey, X. Li, S. Ruehrup, editors, *Ad-hoc, Mobile, and Wireless Networks*, volume 6811 of *Lecture Notes in Computer Science*, pp. 294–307. Springer Berlin Heidelberg, 2011. (Cited on page 53)

[GGG05] R. Gummadi, O. Gnawali, R. Govindan. Macro-programming Wireless Sensor Networks Using Kairos, 2005. (Cited on page 14)

[GHA11] N. Genon, P. Heymans, D. Amyot. Analysing the Cognitive Effectiveness of the BPMN 2.0 Visual Notation. In *Software Language Engineering*. 2011. (Cited on pages 23 and 33)

[GZBD11] F. Gao, M. Zaremba, S. Bhiri, W. Derguerch. Extending BPMN 2.0 with Sensor and Smart Device Business Functions. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on*, pp. 297 –302. 2011. doi:10.1109/WETICE.2011.50. (Cited on pages 12 and 51)

[JAC03] Z. JACKOWSKI. Business Modeling with UML: A Business Process Centred Architecture, 2003. (Cited on page 53)

[jBP12] jBPM User Guide, 2012. URL http://docs.jboss.org/jbpm/v5.3/userguide/ch.core-bpmn.html#d0e2587. (Cited on pages 23 and 52)

[KAA97] J. D. Kiper, B. Auernheimer, C. K. Ames. Visual Depiction of Decision Statements: What is Best forProgrammers and Non-Programmers? *Empirical Softw. Engg.*, 2(4):361–379, 1997. doi:10.1023/A:1009797801907. (Cited on page 14)

[KGK+11] O. Kopp, K. Görlach, D. Karastoyanova, F. Leymann, M. Reiter, D. Schumm, M. Sonntag, S. Strauch, T. Unger, M. Wieland, R. Khalaf. A Classification of BPEL Extensions. *Journal of Systems Integration*, 2(4):2–28, 2011. (Cited on pages 64 and 66)

[LE06] Y. Levy, T. Ellis. A systems approach to conduct an effective literature review in support of information systems research. *Informing Science International Journal of an Emerging Transdiscipline*, 9:181–212, 2006. (Cited on page 15)

[Ley10] F. Leymann. BPEL vs. BPMN 2.0: Should You Care? In *2nd International Workshop on BPMN*, Lecture Notes in Business Information Processing. Springer, 2010. (Cited on page 22)

[LKS11] A. Lodhi, V. Köppen, G. Saake. An Extension of BPMN Meta-model for Evaluation of Business Processes. *J. Riga Technical University*, 43:27–34, 2011. (Cited on page 51)

[LR00] F. Leymann, D. Roller. *Production workflow: concepts and techniques*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000. (Cited on page 22)

[Moo09] D. Moody. The "Physics" of "Notations": Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *Software Engineering, IEEE Transactions on*, 35(6):756 –779, 2009. doi:10.1109/TSE.2009.67. (Cited on pages 23 and 30)

[MP11] L. Mottola, G. P. Picco. Programming wireless sensor networks: Fundamental concepts and state of the art. *ACM Comput. Surv.*, 43(3):19:1–19:51, 2011. doi:10.1145/1922649.1922656. (Cited on pages 11, 14, 19, 20, 21 and 22)

[Obj11] Object Management Group. Business Process Model and Notation Version (BPMN) Version 2.0 Specification. Technical report, Object Management Group (OMG), 2011. (Cited on pages 8, 9, 23, 24, 25 and 52)

[Org07] OASIS. *Web Services Business Process Execution Language Version 2.0 – OASIS Standard*, 2007. (Cited on page 14)

[PHM+08] V. Pentikäinen, T. Heikkilä, K. Määttä, P. Tukeva, M. Korkalainen, P. Saavalainen, P. Kilpeläinen. Industrial and non-consumer applications of wireless sensor networks. pp. 69830K–69830K–13, 2008. doi:10.1117/12.786886. (Cited on page 12)

[RDD+11] T. Rodrigues, P. Dantas, F. C. Delicato, P. F. Pires, L. Pirmez, T. Batista, C. Miceli, A. Zomaya. Model-Driven Development of Wireless Sensor Network Applications. *Embedded and Ubiquitous Computing, IEEE/IFIP International Conference on*, 0:11–18, 2011. doi:http://doi.ieeecomputersociety.org/10.1109/EUC.2011.50. (Cited on page 15)

[Rec10] J. C. Recker. Opportunities and constraints : the current struggle with BPMN. *Business Process Management Journal*, 16(1):181–201, 2010. (Cited on page 22)

[SAP09] BPM Technology Taxonomy: A Guided Tour to the Application of BPM. Technical Report 1, SAP, Accenture, 2009. This paper provides a survey of the practices and technology related to business process management (BPM). Basic concepts are explained, the transformational effect on the enterprise is examined, and the value that BPM can create is analyzed. The paper then presents a survey of the vast array of technology that is related to BPM and sorts out how and when such technology is used. (Cited on page 22)

[SCV11] L. J. R. Stroppi, O. Chiotti, P. D. Villarreal. Extending BPMN 2.0: Method and Tool Support. In *BPMN*, pp. 59–73. 2011. (Cited on pages 8, 25 and 26)

[Sil11] B. Silver. *Bpmn Method and Style, 2nd Edition, with Bpmn Implementer's Guide: A Structured Approach for Business Process Modeling and Implementation Using Bpmn 2.0*. Cody-Cassidy Press, 2011. (Cited on page 23)

[SKI08] S. Stein, S. Kühne, K. Ivanov. Business to IT Transformations Revisited. In *MDE4BPM 2008*, volume 17 of *Lecture Notes in Business Information Processing*, pp. 178–187. Springer, 2008. doi:10.1007/978-3-642-00328-8_18. (Cited on page 14)

[SMZ07] K. Sohraby, D. Minoli, T. Znati. *Wireless Sensor Networks: Technology, Protocols, and Applications*. Wiley-Interscience, 2007. (Cited on pages 11, 19 and 21)

[TSD+12] S. Tranquillini, P. Spieß, F. Daniel, S. Karnouskos, F. Casati, N. Oertel, L. Mottola, F. J. Oppermann, G. P. Picco, K. Römer, T. Voigt. Process-based design and integration of wireless sensor network applications. In *Proceedings of the 10th international conference on Business Process Management*, BPM'12, pp. 134–149. Springer, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-32885-5_10. (Cited on pages 11, 12, 13, 18, 20, 39, 45, 53 and 57)

[Wes07] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer New York, Inc., Secaucus, NJ, USA, 2007. (Cited on page 29)

[Zou12] M. Zouaoui. *Ein Modellierungswerkzeug für Produktionsprozesse auf Basis einer BPMN-Erweiterung*. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2012. (Cited on pages 67, 68 and 74)

[ZSL11] S. Zor, D. Schumm, F. Leymann. A Proposal of BPMN Extensions for the Manufacturing Domain. In *Proceedings of the 44th CIRP International Conference on Manufacturing Systems*. 2011. (Cited on page 51)

All links were last followed on February 14, 2013.

Declaration

I declare herewith that I am the author of this thesis. I did not use any other sources than those named and all those passages quoted from other works either literally or in the general sense are marked as such. Neither the complete thesis nor essential parts of it have been used up to now in any other examination procedure. I have not published this thesis up to now either in part or as a whole. The electronically submitted version is identical to all the other versions which have been submitted.

(C. Timurhan Sungur)