future internet

MDPI

Article

A Model Based Framework for IoT-Aware Business Process Management

Paolo Bocciarelli *, Andrea D'Ambrogio and Tommaso Panetti

Department of Enterprise Engineering, University of Rome Tor Vergata, Via del Politecnico 1, 00133 Rome, Italy
* Correspondence: paolo.bocciarelli@uniroma2.it

Abstract: IoT-aware Business Processes (BPs) that exchange data with Internet of Things (IoT) devices, briefly referred to as IoT-aware BPs, are gaining momentum in the BPM field. Introducing IoT technologies from the early stages of the BP development process requires dealing with the complexity and heterogeneity of such technologies at design and analysis time. This paper analyzes widely used IoT frameworks and ontologies to introduce a BPMN extension that improves the expressiveness of relevant BP modeling notations and allows an appropriate representation of IoT devices from both an architectural and a behavioral perspective. In the BP management field, the use of simulation-based approaches is recognized as an effective technology for analyzing BPs. Simulation models need to be parameterized according to relevant properties of the process under study. Unfortunately, such parameters may change during the process operational life, thus making the simulation model invalid with respect to the actual process behavior. To ease the analysis of IoT-aware BPs, this paper introduces a model-driven method for the automated development of digital twins of actual business processes. The proposed method also exploits data retrieved by IoT sensors to automatically reconfigure the simulation model, to make the digital twin continuously coherent and compliant with its actual counterpart.

Keywords: IoT; business process; IoT-aware BPs; simulation; model-driven; digital twin

Citation: Bocciarelli, P.; D'Ambrogio, A.; Panetti, T. A Model Based Framework for IoT-Aware Business Process Management. *Future Internet* **2023**, *15*, 50. https://doi.org/10.3390/fi15020050

Academic Editor: Davide Tosi

Received: 3 January 2023
Revised: 23 January 2023
Accepted: 24 January 2023
Published: 28 January 2023

1. Introduction

The Internet of Things (IoT) paradigm refers to the interoperability of physical devices such as sensors and actuators, appliances, wearable technology, vehicles, and other objects, embedded with software capabilities and network connectivity that allow these items to collect and exchange data among them and from/to a cloud environment [1].

In this context, IoT-aware Business Processes (BPs) describe, from an operational point-of-view, how data collected by sensors can be processed and provided to human and automated process participants, for promptly reacting to changes in the monitored environment.

Due to their complexity, the development of IoT systems and relevant IoT-aware BPs claim the adoption of innovative methodologies that support all involved stakeholders from the early stages of the development process. As an example, at design time, various alternatives have to be evaluated in order to identify the one that better fits with the system requirements. At execution time, the process behavior has to be constantly monitored in order to identify issues (e.g., performance downgrades, failures, etc.) and timely plan and execute appropriate corrective actions. Among the various approaches for planning such actions, the use of simulation is acknowledged to be the most valuable and effective [2].

In this respect, this work proposes a novel framework which aims at supporting the development and simulation-based analysis of IoT-aware BPs. The framework conceptual design has been based on principles introduced in the Model-driven Engineering (MDE) field and makes use of technologies and standards provided by the OMG's Model Driven Architecture (MDA) [3], which have proven to be effective for simulation-based analysis of complex systems [4].

The development of executable simulations requires building and parameterizing a simulation model, according to the structure and relevant properties of the process under study. The simulation model parameters may change during the process operational life, thus making the simulation model invalid with respect to the actual state of the simulated process.

In this respect, the proposed framework exploits the digital twin (DT) paradigm [5] and introduces *model transformations* for the automated development of a DT corresponding to the actual BP under analysis. Data retrieved from IoT sensors are also used to automatically reconfigure the simulation model, to make the DT continuously coherent and compliant with its actual process counterpart.

In this regard, the objective of this paper is to tackle the analysis of IoT-aware BPs from conceptual, methodological and technical perspectives, so as to address the three following Research Questions (RQs):

RQ1 —How can we represent a model of an IoT system, consisting of a network of sensors, that includes enough information to ease and possibly automate the implementation of both the actual system and its DT?

RQ2 —Which steps do we need to undertake and which technologies do we need to introduce to effectively exploit the IoT system model for implementing both the actual system and its DT?

RQ3 —What Architectural Building Blocks (ABBs) and relevant capabilities does the Framework Architecture provide to effectively guide its implementation?

According to this point of view, the contribution proposed by this paper can be summarized as follows:

- An IoT metamodel that provides a flexible and coherent description of entities involved in an IoT system consisting of a *sensors network*, i.e., an interconnected collection of IoT sensors, is introduced. According to the founding principles of MDA, the specification of a metamodel which describes domain entities and their relationships is the first and essential task to carry out. The proposed metamodel is used for deriving the following software artifacts:
  - An *IoT-aware UML profile* for annotating SysML models that provide the *structural view* of the IoT system;
  - An *IoT-aware extension of the Business Process Model and Notation (BPMN)*, the standard notation for BP modeling, for specifying the *operational view* of the IoT system;
- A *methodology* that illustrates how the framework exploits the UML profile and the BPMN extension to effectively support the development and analysis of IoT-aware BPs is introduced, along with appropriate model transformations for automating the generation of both the system and its digital twin;
- An *architecture* that describes the framework in terms of its components (i.e., ABBs) and the capabilities they shall provide is specified.

As regards the modeling of BPs, it is worth noting that several formalisms are available, e.g., Petri-Net [6], Yet Another Workflow Language (YAWL) [7], or Event-Driven Process Chain (EPC) [8]. Different from the various modeling languages that focus on specific levels of abstraction (e.g., business level or technical level), BPMN aims to provide a notation that is easily readable by all the different roles involved in a business process management effort, bridging the gap between the BP design and the process implementation. Over the years, BPMN has gained an ever-growing popularity, until it has become a de facto standard in the BPM field [9]. Therefore, as this work deals with BP modeling from both operational and technical perspectives, BPMN has been considered the most suitable option to achieve the research objectives.

Finally, in order to evaluate the feasibility of the proposed approach, this work also includes an experimentation dealing with a Water Monitoring System (WMS).

The rest of this paper is structured as follows. Section 2 reviews the existing literature and clarifies the novelty of the proposed contribution. Section 3 provides the concepts at

the basis of this work, by briefly outlining the DT paradigm, the Sensor Web Enablement framework, the Model-driven Engineering principles and the PyBPMN/eBPMN approach for supporting the simulation-based analysis of BPs. Section 4 introduces the proposed IoT-aware framework for BP analysis. Section 5 provides details on the framework implementation. Section 6 discusses an example application to monitor the risk of flooding of a basin, and finally Section 7 provides concluding remarks.

2. Related Work

Great effort has been made over the past years in dealing with the modeling and execution of IoT-aware BPs in the business process management (BPM) field. As this topic has been faced from various perspectives, the literature review hereby proposed specifically focuses on the semantic description of IoT systems, the specification of middlewares and architectures for supporting the actual implementation of IoT systems, and finally, the identification of approaches for enacting the IoT-ware BP management.

As regards the effort spent in investigating IoT systems from a conceptual point of view, interesting contributions can be found in [10–13].

In [10], a novel semantic model, namely *Internet of Things in Business Processes Ontology (IoT-BPO)* is introduced. Such a model is designed as an extension of IoT-Lite, a semantic model based on the IoT-A reference model [10] that formalizes the main concepts of the IoT domain.

The Global Sensor Network (GSN), a scalable infrastructure for integrating heterogeneous sensor networks, is described in [11]. GSN introduces a set of abstractions to represent sensors and IoT devices, which are specified as XML-based descriptors.

In [12], the International Telecommunication Union (ITU) introduces an architecture that, similarly to the well-known Open Systems Interconnection (OSI) model, specifies the different layers that an IoT System should be structured on: sensing, access, network, middleware and application.

The Semantic Sensor Network (SSN) Ontology [13] is one of the most prominent contributions dealing with the specification of a conceptual picture that describes entities, actions and relations involved in the IoT domain.

In this respect, this paper introduces an IoT metamodel, which has been mostly inspired by the SSN ontology. As discussed in Section 4, the metamodel is used to derive a SysML profile and a BPMN extension that enable the specification of the *structural* and *operational* views of an IoT system, respectively.

Among the various contributions that specifically address the heterogeneity of sensor networks from an architectural point of view, relevant examples are [14,15].

The Hydra middleware was introduced in [14] as an outcome of an EU project that aimed at building a service-oriented middleware for networked embedded systems.

Open Sensor Hub (OSH) [15] is an open source framework which aims at easing the development of smart and scalable sensors networks interconnected by a web platform. The OSH implementation is based on the Software Web Enablement Framework [16], an initiative of the Open Geospatial Consortium to support the implementation of IoT systems. Specifically, SWE includes various standards dealing with the definition of sensor data models, encoding and interface specifications, for enacting the exchange of information among devices in a sensors network.

As discussed in Section 6, this paper introduces a prototype of the architecture for supporting the simulation based analysis of IoT-aware BPs. Such a prototype makes use of OSH as a middleware for enabling the seamless interaction of the BPM layer with the underlying IoT layer.

The deployment of IoT-based solutions in business domains requires addressing issues that go beyond the technological complexity. The network of IoT devices should be appropriately integrated with the BPM infrastructure to enable the business layer able to exploit real-time and historical data collected by the sensors. In this regard, significant contributions can be found in [17–23].

As stated in [17], the integration of the IoT technology in the BP domain constitutes an opportunity for business managers to enhance the business performance and increase business competitiveness. As an IoT-aware BP addresses concepts from both the BP and the IoT domains, this paper proposes a review of relevant approaches that deal with the integration of IoT technology and BPs. According to the proposed analysis, a valuable summary of BP-related concepts is provided by OMG's BPMN specification. Moreover, as regards the conceptual representation of the IoT domain, it is argued that it should consider the definition of various concepts: devices, sensors, actuators and users. In this respect, a valuable strategy for enacting the IoT integration with the BP paradigm is to extend BPMN to enable the representation of IoT concepts and hence integrate them into an IoT-aware BP model. In this respect, such a contribution provides the conceptual guidance to define the strategy upon which we have based our approach, whose founding pillar is constituted by the specification of an IoT-based extension of BPMN that addresses the representation of the aforementioned IoT-related concepts.

In [18], an approach that combines the recording of data collected by IoT sensors with BPM technologies is presented. The authors underline how provisioning of real-time data gathered from sensors to BPM tools might help organizations to achieve cost savings and efficiency. The proposed approach consists of a four-step procedure: IoT devices are connected to the business process management system (BPMS), the BPMN model is extended with data variables for capturing data from sensors, the process model is executed and, finally, context relevant information are provided to process participants. Unlike such an approach, the framework introduced in this work aims to support both the process design and its continuous monitoring and analysis by use of a digital twin. Moreover, the proposed approach has been based on a low code development paradigm, thanks to the adoption of standard and technologies introduced by the MDA effort.

In [19], a novel modeling approach is introduced to provide guidance for integrating IoT resources in a BP model. The proposed method, denoted as the IoT-Aware Process Modeling Method (IAPMM), exploits the IoT-Architecture [24], a notation to represent the IoT concepts in BPMN in terms of annotations associated to BPMN models.

In [20,21], the lack of modeling concepts for representing IoT devices in a business process model is underlined. Specifically, in [20], an IoT domain model for BPs is introduced for both physical entities and device elements. While a physical entity denotes an element in the observed environment, device elements allow the representation of IoT-related concepts. A device element can further be specialized by three subcategories: sensors, actuators, and tags. A sensor is in charge of monitoring the physical entity, whereas an actuator acts on the physical entity to change its state. A tag is used to identify a device. Differently, in [21], a BPMN extension is proposed to deal with the representation of IoT concepts in a BP model. The paper analyzes and compares three potential strategies for extending BPMN to represent physical IoT devices. According to the presented results, two new classes are introduced: the *ParticipantContainer* and the *PhysicalEntity*. Specifically, *ParticipantContainer* is an abstract class that extends the standard *BaseElement* class and is specialized by subclasses that represent different types of entity participating in a collaboration, i.e., the novel *PhysicalEntity* class and the standard *Participant* class.

In [22], a REST-based architectural model is introduced to integrate BPs with IoT resources. Specifically, the proposed architecture provides the core components that enact the development and execution of IoT-aware BPs. To demonstrate the feasibility of the proposed architectural approach, the paper also discussed a prototype implementation based on BPM (Java Business Process Management) [25], an open source workflow engine compliant with BPMN.

The lack of a common architecture which standardizes the communication between the IoT layer and the BPM layer is underlined in [23]. Such a contribution proposes an integrated approach that exploits IoT for BPM and introduces an IoT architecture structured in three layers: the IoT devices layer, the IoT communication middleware and the BP execution layer (also known as BPMS, or BP management software). The proposed

architecture allows sensors to be digitally accessible by humans, machinery and automated systems so to provide the BPMS layer with up-to-date information. Moreover, in order to orchestrate the activities which involve both humans and automated systems, exchanged data are annotated with information about the context and the provenance. Finally, real-time notifications are provided to human participants through mobile devices.

In this respect, the BPMN extension and the architecture proposed in this paper share some aspects with the above-mentioned contributions, specifically with regard to the need of extending the BPMN expressiveness, the layered approach used for the architecture specification, and the identification of most relevant architectural components. Differently, this paper goes beyond previous contributions, thanks to the adoption of the MDA standards and the DT paradigm. Specifically, MDA's standards and technologies have been used for deriving the profile and the BPMN extension at the basis of the conceptual contribution. Moreover, the strength of the proposed methodology lies in the introduced model transformation chains, which aims at:

(i) Easing the development of the architecture's implementation components which depend on the particular IoT devices used in the sensors network, i.e., the data model layer and the required drivers for interacting with IoT sensors are generated by model transformations, as discussed in Section 4.6);

(ii) Keeping the methodology flexible enough to be used in multiple operational contexts (e.g., different BPMN specification tools or various BPMS can be supported, as clarified in Section 4.6).

3. Background

This Section briefly outlines standards and technologies at the basis of the proposed contribution. Specifically, Section 3.1 introduces the digital twin paradigm, Section 3.2 illustrates the OGC Sensor Web Enablement (SWE) standard and the OGC-W3C Semantic Sensor Network Ontology (SSN), while Section 3.4 briefly describes PyBPMN/eBPMN, an integrated approach for BP modeling&simulation (M&S) that focuses on performance and reliability aspects.

3.1. Digital Twins

A *Digital Twin (DT)* [26,27] is a dynamic digital representation of a physical system that can be seen as a virtual instance of the physical system, often referred to as the *Physical Twin (PT)*. In a traditional M&S approach, simulation models of a given system (i.e., the system under study) can be developed at different steps of the system life-cycle for addressing various objectives:

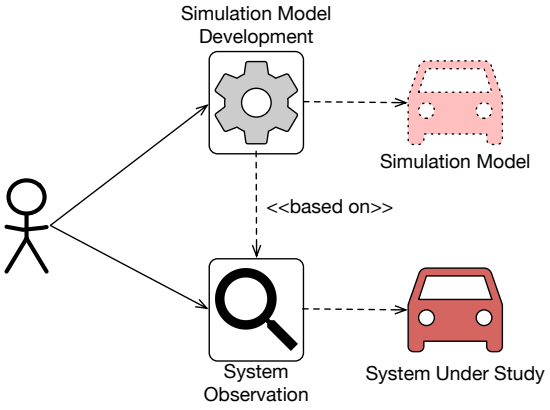
- At *execution time*, an existing system is observed and a corresponding simulation model is specified. A simulation-based analysis is performed to predict system behavior under given conditions and to ultimately obtain guidance to properly address critical situations (see Figure 1a);
- At *design time*, a simulation model can be specified starting from models of the system to be developed. Simulation-based analysis allows developers to compare different design alternatives and evaluate the system behavior before its actual implementation (see Figure 1b).

In order to be coherent with the system under study, a simulation model needs to be *parameterized*. Indeed, the various properties of the real-world environment have to be used for deriving the relevant configuration parameters of the simulation model. One of the most relevant drawbacks of a traditional M&S approach is that the simulation model does not take into account the dynamic evolution of the physical system during its operational life. Changes in the properties of the physical system might invalidate the simulation model with respect to the actual behavior of the system.

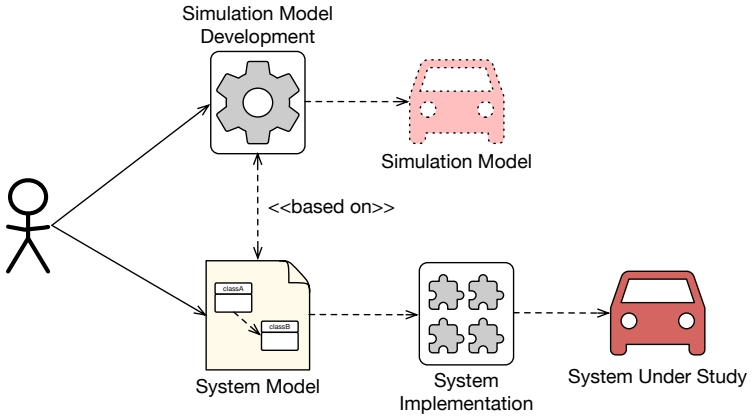
In this respect, the DT paradigm aims at overcoming such issue. DTs are constantly updated according to actual performance, health status and context data of the PT [5], thus keeping the DT continuously coherent with its physical counterpart.

Due to their novelty and relevance, DTs are increasingly being used in several application domains [28].

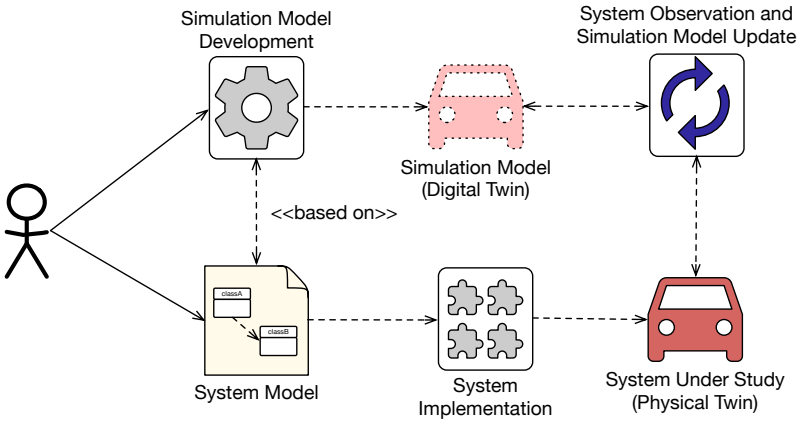
As discussed in Section 4, the framework proposed in this paper introduces an MDA-based approach to ease the development of DT and support the analysis of IoT-aware BPs.



(a) M&S-based Analysis at Design Time



(b) M&S-based Analysis at Execution Time



(c) Digital Twin and Physical Twin

Figure 1. Traditional M&S Approaches vs. Adoption of a Digital Twin.

3.2. SWE Framework and SSN Ontology

A significant challenge in the IoT field comes from the heterogeneity of the physical layer. In an interconnected sensors network, heterogeneous physical entities, provided by different vendors and operating according to specific rules and protocols, must cooperate to achieve the required objectives. An IoT system that aims at collecting and processing the flow of data coming from orchestrated devices must introduce appropriate methods and tools for the seamless interaction with various devices. In this respect, the interoperability should be addressed from two perspectives:

- *Technical* and *syntactical interoperability* should be ensured by introducing standards and technologies that enable an IoT system to interact with various devices, each using different implementation technologies and communication protocols;
- *Semantic interoperability* should also be ensured in order to make sensor data easily understandable by various actors, including IoT system components in charge of automatically processing the flow of data collected by the sensor network.

In this respect, this paper exploits the OGC Sensor Web Enablement standard [16] and the OGC-W3C Semantic Sensor Netwok (SSN) ontology [13], respectively.

The Sensor Web Enablement (SWE) is a framework promoted by the Open Geospatial Consortium (OGC) for supporting the development of the so-called *Sensor Web*, i.e., a collection of sensors and sensors networks provided and controlled by different owners and interconnected by use of the web platform. The SWE initiative has been started in 2003 and has progressively addressed the development of data models, encoding methods and service interface specifications to provide a set of features that includes:

- *Retrieval of real-time or time-series observations*, along with their coverage in standard encodings;
- *Access to sensor settings* that allow the software to automatically process and geo-locate data;
- *Tasking of sensors* to obtain observations of interest;
- *Subscription to and publication of warnings* to be sent by sensors or sensor services based on particular criteria.

Among the various standards currently included in the SWE Framework, this paper specifically exploits the following two:

- *Observations & Measurements Schema (O&M)*, which provides data models and XML Schemas for encoding historical observations, real-time observations and measurements retrieved by sensors;
- *Sensor Model Language (SensorML)*, which introduces models and XML Schemas for describing the observation processing systems;
- *Sensor Observation Service (SOS)*, which specifies an open interface for interacting with sensors and collecting measurement data.

As regards the definition of ontologies for the semantic description of sensor devices, observation processes, and measurement results, one of the most prominent results has been achieved by the joint work of the World Wide Web Consortium (W3C) and the OGC Consortium, which promoted the Semantic Sensor Network (SSN) Ontology [13].

SSN provides a flexible but coherent perspective for the conceptual description of the entities, relations and activities involved in the measurement process. The pillars of the proposed approach can be summarized in the following concepts defined by SSN:

- *System*: an abstraction of physical infrastructures that implement *procedures*;
- *Sensor*: entity (device, human or automated system, e.g., a software) involved in a *procedure*;
- *Procedure*: a sequence of activities or a protocol carried out to make an *observation*;
- *Observation*: action executed to measure, estimate or calculate the actual value of an *observable property* related to a *feature of interest*. An observation produces a *result*;

- *Observable property*: a characteristic of a *feature of interest* which can be subjected to an *observation*;
- *Feature of interest*: an entity in the real world whose properties are subjected to *observations* to measure or estimate their actual values to obtain a *result*;
- *Result*: the result of an *observation*.

As discussed in Section 4.5, the architecture of the proposed framework exploits the Open Sensor Hub (OSH) [15], a middleware for the development of sensor networks based on SWE standards. Moreover, as outlined in Section 4.2, the SSN and the data models specified by the O&M and the SensorML SWE standards have been used to derive the proposed IoT Metamodel.

For an exhaustive description of SSN and SWE standards, interested readers are referred to the official specification and the aforementioned literature contributions.

3.3. Model-Driven Engineering and Model-Driven Architecture

Model-driven development (MDE) is an approach to software systems engineering that addresses the increasing complexity of execution platforms by focusing on the use of formal models [29]. The founding pillars of MDE are constituted by model transformations. An appropriate chain of *model-to-model* and *model-to-text* transformations is specified and executed in order to progressively translate abstract models into more refined models until executable artifacts or models that meet the desired level of abstraction are generated.

One of the most important initiatives driven by MDE is the *Model Driven Architecture (MDA)*, the Object Management Group (OMG) incarnation of MDD principles [3].

The Model-Driven Architecture prescribes that various models have to be specified throughout the software development lifecycle. Such models, which provide different views of the system, are specified from viewpoints focusing on particular system concerns.

The following main standards have been introduced as part of the MDA effort:

- *Meta Object Facility (MOF)*: for specifying technology neutral metamodels (i.e., models used to describe other models) [30];
- *XML Metadata Interchange (XMI)*: for serializing MOF metamodels/models into XML-based schemas/documents [31];
- *Query/View/Transformation (QVT)* and *MOF Model To Text (MOFM2T)*: for specifying *model-to-model* and *model-to-text* transformations, respectively [32,33].

In a general model-driven engineering process, an input model can be used for generating an output model or a textual document (e.g., executable code, text documents, configuration files, etc.)

Specifically, let us consider an input model Model M_A, which is an instance of source metamodel MM_A. Let us assume that such a model has to be mapped to the output model M_B, which, in turn, is an instance of target metamodel MM_B.

Both MM_A and MM_B have to be specified in terms of *MOF Model* constructs.

The *model-to-model* transformation that generates the output model is specified by use of QVT, the declarative language for specifying model-to-model transformations. In order to be handled by a QVT transformation engine, the input and output models have to be serialized as XML-based documents which are obtained by applying the rules specified by the XMI standard.

Finally, the required textual document is generated throughout the execution of an appropriate *model-to-text* transformation specified by use of the *MOFM2T metamodel* standard.

3.4. PyBPMN/eBPMN

In order to automate the building and execution of BP simulation models, this work exploits an approach denoted as *PyBPMN/eBPMN*, which has been introduced and described in previous contributions [34], and that is hereby summarized for the sake of completeness.

Specifically, *PyBPMN (Performability-enabled BPMN)* is a BPMN extension that addresses the specification of performance and reliability properties of BPs [35,36], while *eBPMN* is a Java-based domain-specific simulation framework [37,38].

The PyBPMN extension is based on principles and standards introduced in the model-driven engineering field, specifically by the OMG's Model Driven Architecture (MDA) [3]. Such an extension addresses the non-functional characterization of a BP according to four different dimensions:

- *Workload*, to model the workload submitted to BP tasks;
- *Performance*, to specify the performance properties, i.e., efficiency-related properties such as the service time, associated to single tasks;
- *Reliability*, to specify the reliability properties of the resources that tasks depend on to carry out work requests;
- *Resource management*, to describe the execution platform actually used for executing the BP.

The PyBPMN-based modeling approach does not introduce a separate notation for representing the BP under study. Rather, the functional requirements of the process to be analyzed are initially used for specifying a standard BPMN model. Then, the model is enriched with annotations that capture the non-functional requirements, according to elements (or metaclasses) introduced in the PyBPMN metamodel. As such, a PyBPMN model is still a valid BPMN model.

PyBPMN models can be directly simulated by using eBPMN, a Java-based domain-specific simulation framework. The eBPMN architecture, which has been designed to comply with the execution semantics of the BPMN version 2.0 specification, reflects the structure of the PyBPMN metamodel. Relevant concrete classes have been defined for explicitly handling the non-functional characterization of a BP (e.g., the service demand for a given resource or the probability of a failure/repair event), in order to mimic the realistic behavior of the BP under study. As an example, `EBPMNResource` is the abstract superclass for all the resource classes defined in eBPMN. This class holds a list of references to the eBPMN elements that use the resource. The `EBPMNResource` class can be specialized as *Performer*, i.e., the atomic entity that actually executes the service request, or as *Broker* and *Subsystem*, which provide the implementation of complex patterns of composite resources.

4. IoT-Aware BPM Framework

This section illustrates the proposed framework for the IoT-aware BPM. Specifically, Section 4.1 states the requirements that guided the framework specification, Section 4.2 provides the rationale which has driven the framework development and outlines its conceptual pillars, Section 4.3 introduces *SysML4IoT*, a profile for describing sensor networks, Section 4.4 describes the *IoT-BPMN* extension and, finally, Section 4.5 provides an architectural view of the proposed framework and, finally, Section 4.6 discusses the IoT-aware BPMN framework from an operational perspective.

4.1. Framework Requirements

The proposed framework has been designed to address the following requirements:

- *REQ1*: the framework shall support the specification of BP models by use of the BPMN notation;
- *REQ2*: the framework shall provide appropriate features to enrich the BPMN model with information characterizing the underlying IoT platform;
- *REQ3*: the framework shall support the specification of the IoT platform structure by use of SysML;
- *REQ4*: an appropriate UML profile should be provided to enrich the expressiveness of SysML with regards to the IoT domain;
- *REQ5*: the framework shall be able to support the simulation of the BPMN model;
- *REQ6*: the framework shall include a BP execution engine able to interact with the underlying IoT platform;
- *REQ6*: the framework shall be able to update the simulation model according to the actual state of the system, as derived from data collected by IoT sensors;

Sections 4.2–4.6 clarify how such requirements have been addressed.

4.2. Rationale and Overview

As outlined in Section 1, this paper introduces a conceptual framework for the analysis of IoT-aware BPs, e.g., processes which specifically monitor, exploit and manage data retrieved from a network of distributed IoT sensors.

The strategy at the basis of the proposed framework development is outlined in Figure 2.

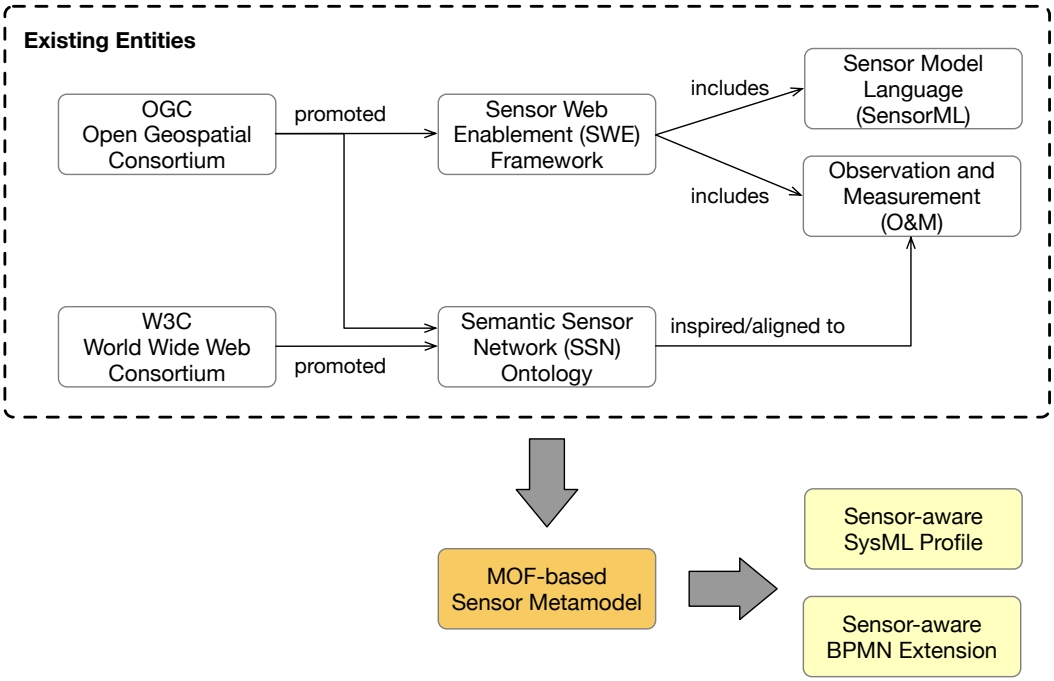


Figure 2. Conceptual strategy for developing the framework for the analysis of IoT-aware BPs.

As a first step, in order to describe from a conceptual point of view the addressed sensors network domain, an MOF-based metamodel has been initially specified, MOF being the key MDA standard that provides an abstract language and a framework for specifying, constructing, and managing technology neutral metamodels, e.g., models used to describe other models [30].

The IoT metamodel describes entities that compose a sensor network system and the relationships among them. As shown in the upper part of Figure 2, in order to identify the metamodel elements, existing ontologies and frameworks dealing with the addressed domain have been analyzed. Specifically, the proposed metamodel is inspired by the following elements:

- *Sensor Web Enablement (SWE) framework*, which has been promoted by the OGC Spatial Consortium and includes various standards, each related to a particular aspect in the general context of sensors network development (e.g., sensors API, data representation, data encoding, etc.). Specifically, the proposed metamodel design is based on the sensor structural model and the data model provided by the SensorML and the O&M standards, respectively;
- *Semantic Sensor Network Ontology (SSN)*, which is one the most relevant ontologies for conceptual description of sensors network systems.

The proposed IoT Metamodel is shown in Figure 3.

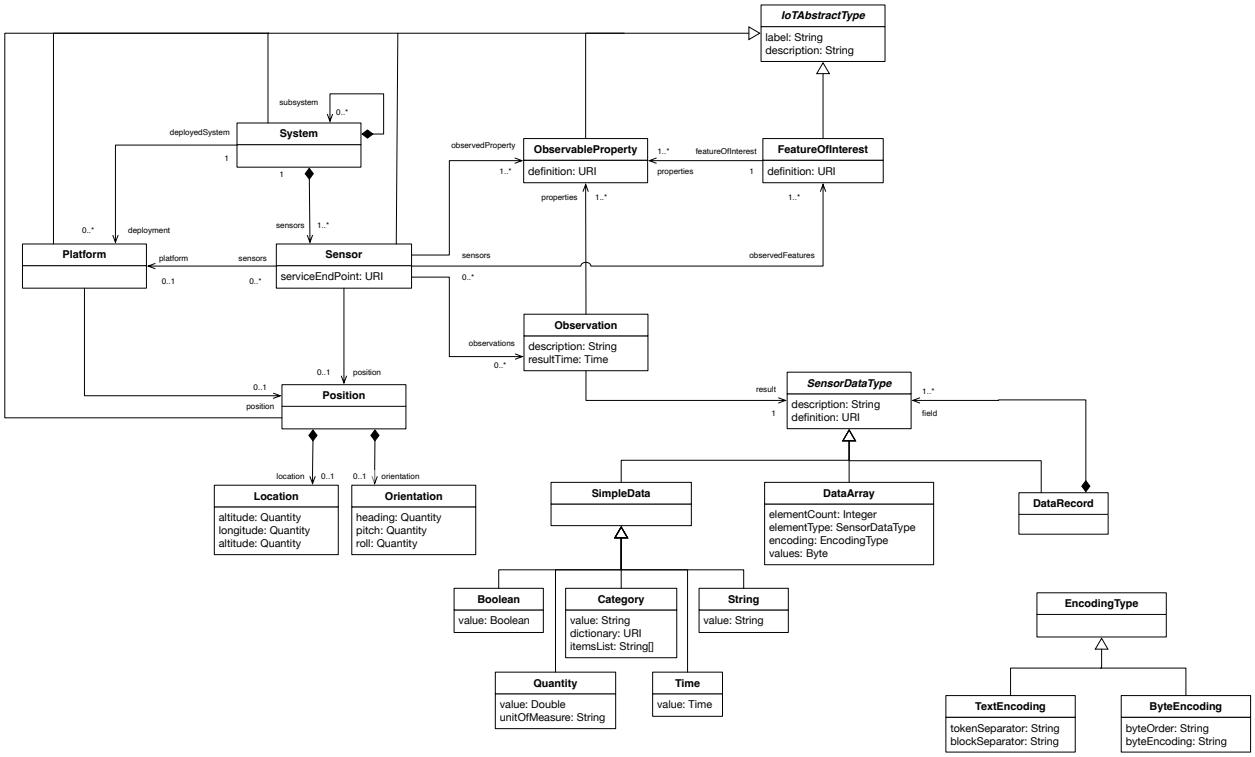


Figure 3. IoT Metamodel.

According to the figure, a sensor network system (represented by the `System` entity) is defined as a collection of `Sensors` (or, in other words, sensing devices). Each sensor provides a *serviceEndPoint* which is used for inquiring the device and retrieving the collected measures.

Sensors might be single, stand-alone devices or they can be structured in a `Platform`, which is a physical component constituting the system deployment. A `Position` can be specified for sensors and platforms, to describe the entity `Location` (in terms of *latitude*, *longitude* and *altitude*) and its `Orientation` (by specifying the entity *heading*, *pitch* and *roll*).

Sensors observe properties (`ObservableProperty` entity), which denote physical entities' characteristics subjected to a measure, and make observations (`Observation` entity). The observable properties of an entity can be grouped into features of interest (*FeatureOfInterest* entity). As an example, let us consider a meteorological sensor for measuring the rainfall and the wind speed. The *weather condition* is the *feature of interest* observed by sensors, while the *rainfall* and *windspeed* are two *observable properties* characterizing the *weather condition* feature of interest.

Properties and features of interest are characterized by a `Definition` URI, which allows one to associate a reference to an external ontology or dictionary that provides the relevant conceptual description.

Any `Observation` (or measure) collected by a sensor includes one `Result`. As results are typed elements, the metamodel includes the following types:

- **SimpleData**: metaclass which is further specified by the following subclasses:
  - *Boolean*, which represents a boolean type. The `value` attribute provides the measured boolean value of the observed property;
  - *Quantity*, which is a type representing a physical quantity whose measure requires the specification of a `value` along with the related unit of measure (`unitOfMeasure` attribute);
  - *Category*, which is a type representing a property whose actual value is defined in terms of a limited and fixed set of possible values. It is characterized by the following attributes: `value`, which is the actual value of the measured property,

`dictionary`, which refers to the external resource defining the set of allowable values and `itemList`, which directly provides the allowable values;
  - *String*, which represents a sequence of characters;
  - *Time*, which denotes an observable property representing a time and/or a date value.
- **DataArray**, which is a complex type specified in terms of a sequence of same type elements. It is characterized by the following attributes:
  - *elementCount*, which denotes the array length;
  - *elementType*, that is the type of each array's item. As it is defined as a `SensorDataType`, an array contains elements type by any possible sensor datatype;
  - *encoding*, that describes how elements are encoded in the array. Two encodings are provided: `TextEncodings`, according to which the array elements are encoded as a text-based list of tokens, or `ByteEncoding` for items encoded as a row sequence of bytes;
  - *values*, which contains the encoded data representing the actual array value;
- **DataRecord**, which is a complex type representing a collection of elements. Unlike the DataArray type, `fields` in a DataRecord are not supposed to be of the same type.

Finally, it should be noted that any result is associated with a textual *Description*, and a *Definition* URI that specifies the external dictionary/ontology used for its formal definition.

As discussed in Section 1, in order to effectively support the simulation-based analysis of BPs dealing with a sensor network, the proposed IoT-aware methodology addresses the system representation from two modeling perspectives: the *structural view*, specified by a SysML model, and the *operational view*, provided in terms of an IoT-aware BP.

In this respect, as outlined in Figure 2, two artifacts have been implemented starting from the conceptual representations provided by the IoT Metamodel: the *IoT4SysML* profile and the *IoT-BPMN* extension. The former is a UML profile which allows the annotation of a SysML model representing the structural view of a sensor network systems. The latter is a BPMN metamodel extension which allows the specification of IoT-aware BPs providing the operational view of the sensor network systems. Such artifacts are described in Sections 4.3 and 4.4, respectively.

4.3. IoT4SysML: An IoT-Aware UML Profile for SysML

The proposed IoT-aware profile for annotating SysML models providing the *structural view* of sensor networks is shown in Figure 4.

The profile includes the following stereotypes, which have been introduced according to relevant concepts/relationships specified in the IoT Metamodel:

- **System**, which extends both the `Package` and the `Class` metaclass and identifies the *Sensors Network System*;
- **Platform**: which extends the `Package` metaclass and denotes a SysML Block element representing a *Platform*. It provides the `position` property for allowing the specification of the Platform's location and orientation;
- **Sensor**, which identifies SysML Blocks representing *Sensors* within the addressed sensor network. It provides the following attributes:
  - `serviceURI`, that specifies an URI endpoint for inquiring the sensor and retrieving the collected measures;
  - `position`, that describes the sensor's position in terms of its location and orientation;
- **FeatureOfInterest**, that denotes a physical object observed by a sensor. It includes the `definition` attribute for referring to an external resource (i.e., a dictionary or an ontology) which formally defines the feature;
- **ObservableProperty**, which denotes a SyML Block `Property` representing an *observable property*. The formal definition of the property can be provided by use of the `definition` attribute;

- **Observes**, which extends the `Association` metaclass. It allows the proper specification of the relationships among sensors and observed entities;
- **SensorData**, which extends the `Property` metaclass and identifies a SysML Block attribute representing data measured by the sensor. As SensorData represents physical and measurable quantities, the `uom` attribute allows the specification of the related *unit of measure*;
- **CategorySensorData**, which extends the `Property` metaclass and identifies a SyML Block attribute representing a *Category Data*, as specified by the IoT Metamodel. As the profile does not make any assumption on the annotated SysML model, the CategorySensorData includes the two following attributes which might be possibly used for further specifying the Category Data:
  - `Dictionary`, which specifies a reference to an external resource;
  - `ItemList`, which includes the list of allowed values;
- **SensorDataRecord**, which extends the `DataType` metaclass and identifies a *DataRecord* type;
- **SensorDataArray**, which extends the `DataType` metaclass and identifies an *Array* type;

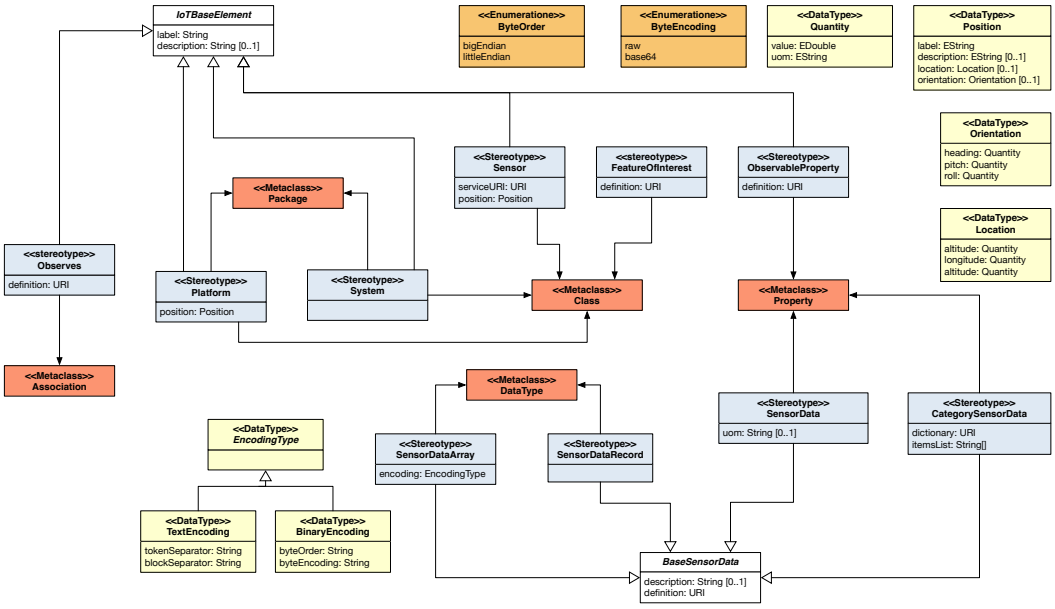


Figure 4. SysML profile for IoT.

4.4. IoT-BPMN: A BPMN Extension for IoT-Aware BPs

This section introduces *IoT-BPMN*, an extension of the BPMN metamodel, in other words the notation that the proposed methodology exploits to provision the *behavioral view* of the IoT System.

As mentioned in Section 1, the proposed extension has been designed and developed according to principles and standards provided by MDA. Specifically, the extension is strongly based on a *metamodeling-based* approach, according to which the existing BPMN metamodel has been analyzed and new metaclasses are introduced as extension of existing metaclasses, without altering the structure of the original BPMN metamodel.

The extension process is illustrated in Figure 5. Specifically, the BPMN metamodel is provided in terms of MOF metaclasses. According to the MDA methodology, a BPMN model is an instance of the corresponding metamodel. The XMI [31] standard provides the rules for

- Serializing an MOF metamodel to an XMI Schema;
- Serializing a model to an XMI document which is validated by the relevant XMI schema.

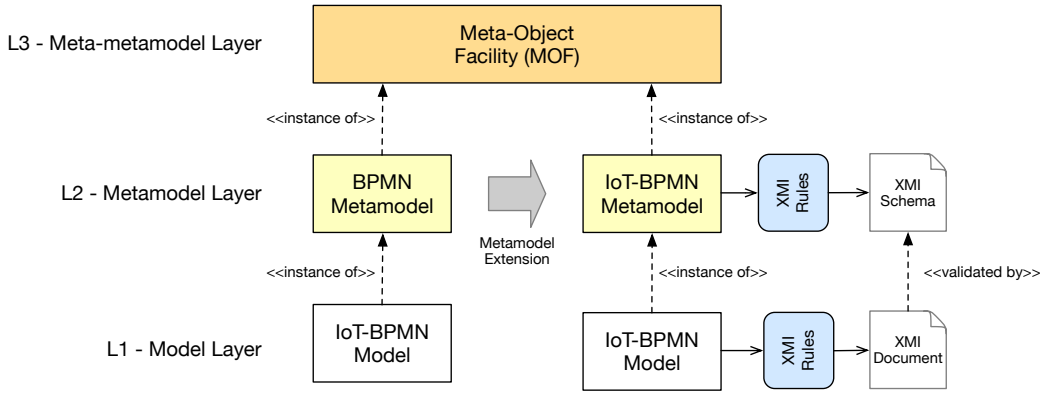


Figure 5. MDA-based Extension Process.

The IoT-BPMN has been defined at layer M2 (metamodel layer) by extending existing BPMN metaclasses.

The IoT-BPMN metamodel is easily obtained by adding metaclasses that extend the original ones, without applying any modifications to the BPMN metamodel. This also has the advantage of maintaining a complete backward compatibility with BPMN, so that a BPMN model conforms to both the BPMN metamodel and the IoT-BPMN metamodel.

The second step of the extension process is the serialization of the IoT-BPMN metamodel by using the XMI Schema Production Rules, in order to obtain the IoT-BPMN XMI Schema.

Finally, the XMI Document Production Rules are instead used to derive an IoT-BPMN document (i.e., a XML document) from the corresponding IoT-BPMN model.

The BPMN metamodel extension which addresses the IoT-related concepts is shown in Figure 6 and consists of the following metaclasses.

- **SensorTasK**, which represents an IoT device. It provides the `serviceEndPoint` parameter that allows the BP engine to inquire the sensor for collecting the measured data;
- **Observation**, which models a single observation made by the sensor. It includes the following attributes:
  - *Description*, for specifying a free description of the measure;
  - ResultTime, which describes when the observation has been made;
  - *Result*, which is a reference to the class providing the actual measured value;
- **ObservableProperty**, which is the subject of an observation. Its `definition` property refers to an external dictionary/ontology where the relevant conceptual definition is provided;
- **FeatureOfInterest**, which describes a feature of interest for the addressed environment. It is specified as a subclass of the *BPMN2::Artifact* metaclass, so that it can be associated to any other BPMN element;
- **SensorDataType**, which represents the abstract type characterizing an observation result. According to the IoT metamodel introduced in Section 4.2, it is further specialized by the concrete classes `SimpleData`, `DataArray` and `DataRecord`.

Finally, it should be emphasized that the above-mentioned metaclasses focus on the IoT-related extension of the BPMN. As the proposed methodology specifically addresses the simulation-based analysis of BPs, the IoT-BPMN extension also includes the metaclasses provided by the PyBPMN metamodel extension [35], briefly summarized in Section 3.4, in order to provide a complete extension that allows one to capture both performance/reliability and IoT-related properties of an IoT system.

The next section illustrates the proposed architecture for enabling the analysis of IoT-aware BPs.

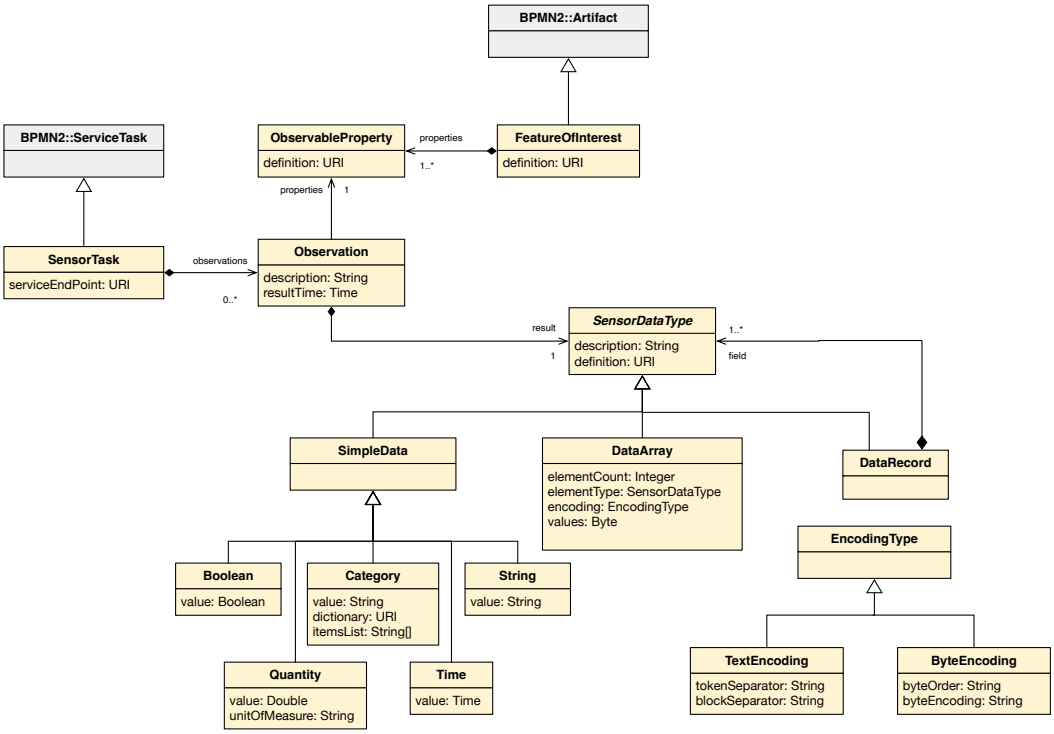


Figure 6. IoT-BPMN Metamodel Extension.

4.5. Architecture for the IoT-Aware BPM

This section introduces the proposed architecture for enacting the simulation-based analysis of IoT-aware BPs. As pointed out in Section 1 and further explained in Section 4.6, the architecture aims at enabling the effective adoption of the digital twin paradigm. Moreover, its design exploits MDA-based standards and technologies to ease the development of the required executable simulation models.

The proposed architecture, which is shown in Figure 7, consists of the following components:

- **BP Modeling and Management**, which provides a visual environment to specify the IoT-aware BP and monitor its execution;
- **BP Execution Engine**, which actually executes the BP. Process tasks that specifically require an interaction with the sensors network invoke services provided by the Broker, which acts as the intermediate layer;
- **Transformation Engine**, which implements the *model-to-text* transformations introduced in Section 4.6 to generate an executable process description tailored for the adopted BP execution engine and a Java implementation of the broker components, according to the *structural view* of the IoT System. It is structured in the following layers:
  - The *M2T Transformation Layer*, which provides the Acceleo-based implementation of the aforementioned transformations, being Acceleo [39] an implementation of MOFM2T [33], the MDA standard for the specification of *model-to-text* transformations;
  - *MOF Metamodel Layer*, which provides an implementation of the IoT-BPMN metamodel based on MOF [30], the MDA standard for the specification of technology-neutral metamodels;
  - The *Eclipse Modeling Framework (EMF)*, which provides the extensible environment for the specification and execution of the model transformations.
- **Broker**, which constitutes the intermediate layer for hiding the technological complexity of the underlying IoT infrastructure, thus providing transparency of the business layer from implementation-related details of the IoT-layer. It includes the following components:

  - The *Open Sensor Hub*, which is the core of the Broker. OSH implements the various standard parts of the SWE effort and provides a service-based interface for interacting with the IoT devices. The OSH implementation natively provides a set of *OSH Basic Drivers* for enabling the data exchange with generic devices;
  - *App-specific Sensor Drivers* and *Sensor Data Layer*, which are the components whose implementation is supported by the `IoTBPMN-to-Sensor` *model-to-text* transformation. Specifically, they provide the implementation of device-specific drivers and data models, respectively. As such components' implementation is strictly tied to the set of concrete sensors, the information needed for their implementation is specified by the use of stereotypes provided by the IoT4SysML profile;
  - Data Retrieval Layer, which is responsible of updating the BP simulation model (i.e., the digital twin) with the data collected by sensors defining the actual state of the real environment (i.e., the physical twin).
- **Simulation Engine**, which allows the execution of the simulation model. It provides an implementation of the eBPMN [37] framework for executing simulation models specified by use of PyBPMN [35];
- **BP Execution Engine**, which is in charge of executing the BP model.

The next section introduces the methodology that clarifies how the IoTSyML profile, IoT-BPMN extension and the architecture discussed herein can be actually exploited to enact the analysis of IoT-aware BPS.

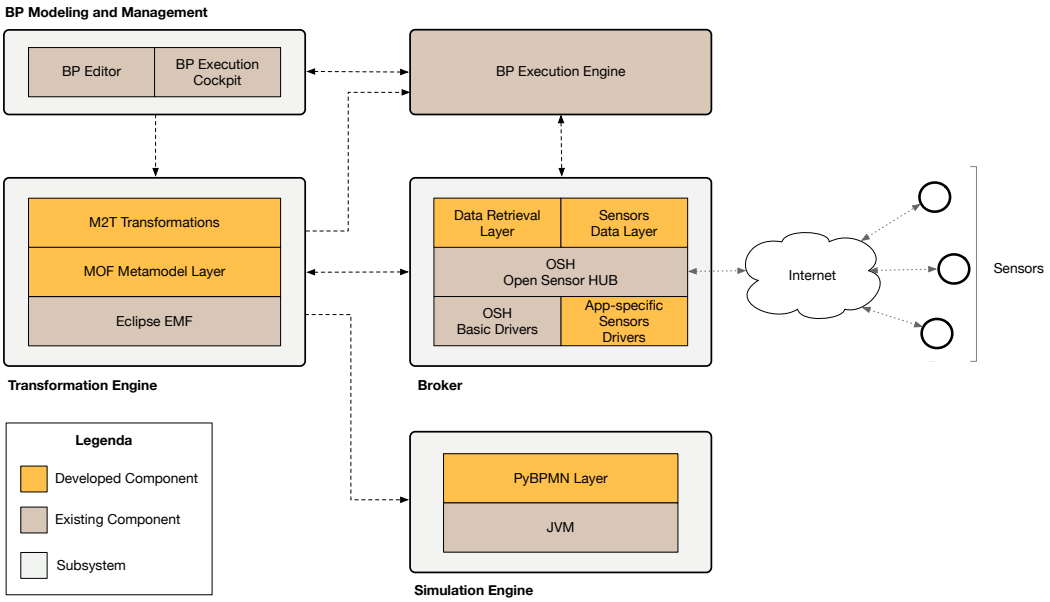


Figure 7. IoT-aware BP Analysis Architecture.

4.6. Methodology for IoT-Aware BPM

The proposed methodology, which is shown in Figure 8, consists of the following steps:

1. Initially, an IoT-BPMN model that provides the *operational view* of the sensors system is specified. The proposed BPMN extension allows systems developers to characterize the process model with information that describe the sensors networks, e.g., service endpoints for Service Task interacting with the underlying IoT platforms, datatype structure for the expected measures, etc.;
2. A SysML model annotated with the IoT4SysML profile is specified for providing the *structural view* of the sensors network;
3. In order to allow the simulation-based analysis of the IoT system, the IoT-BPMN model and the XML-based description of the static simulation parameters characterizing the scenario under study are given as input to the `IoTBPMN-to-eBPMN` *model-to-text*

transformation. In this respect, as explained in Section 4.4, the IoT-BPMN extension includes the metaclasses provided by the PyBPMN metamodel. As such, the IoT-BPMN also describes the performance and reliability properties that characterize the BP (e.g., the activity service demand, the IoT devices MTTF, etc.). The eBPMN code implementing the *digital twin* of the actual IoT-aware BP is thus generated. The eBPMN code execution allows analysts to evaluate the behavior of the IoT system and assess whether or not all the functional and non-functional requirements are satisfied;

4. The `SySML-to-Sensors` *model-to-text* transformation is executed for generating:
   - The sensors driver used for making the PoT Broker component able to actually interact with concrete IoT devices;
   - The required classes that implement the Data Model storing measures collected by sensors.
5. In order to allow the actual execution of the business process, the `IoTBPMN-to-Engine` *model-to-model* transformation is executed for generating a process model compliant with the adopted execution engine. It should be underlined that even though such a transformation is strictly tied to the adopted BP execution engine, the adoption of an MDA-based approach allows keeping the methodology valid regardless the engine adopted in the concrete case: any BP execution environment will be virtually supported by introducing an appropriate model transformation able to generate the required input model;
6. Finally, the IoT-aware BP is executed. The information included in the IoT-BPMN model (e.g., the sensor service end-points), allows the execution engine to interact with the PoT Broker for inquiring the underlying sensors network and collecting the measured data;
7. At execution time, the actual configuration of the sensors system (i.e., the physical twin) may change. In order to keep the digital twin aligned with its physical counterpart, the Data Retrieval Layer is responsible of collecting the parameters which describe the actual system configuration and updating the eBPMN implementation accordingly.

It should be underlined that the methodology does not make any assumption about the availability of an IoT-BPMN modeling tool. Indeed, as shown in Figure 9, the IoT-BPMN model can be obtained throughout the following steps: first, a BPMN model is specified according to the IoT System functional requirements. Then, performance and reliability properties, as well as the description of the IoT devices, are used for enriching the BPMN model with relevant text annotations. Finally, a model-to-model transformation is executed to generate the corresponding PyBPMN and IoT-BPMN models. In this respect, in order to be automatically processed by a *model-to-text* transformation, text annotations should follow the EBNF [40] formal syntax illustrated in [41] and conform to the PyBPM and the IoT metamodels.

In order to show how the proposed methodology can be used in practice, a prototype of the framework has been developed and used in a scenario dealing with a Water Alert System. In this respect, the next section provides relevant implementation details, while Section 6 describes the corresponding experimentation.

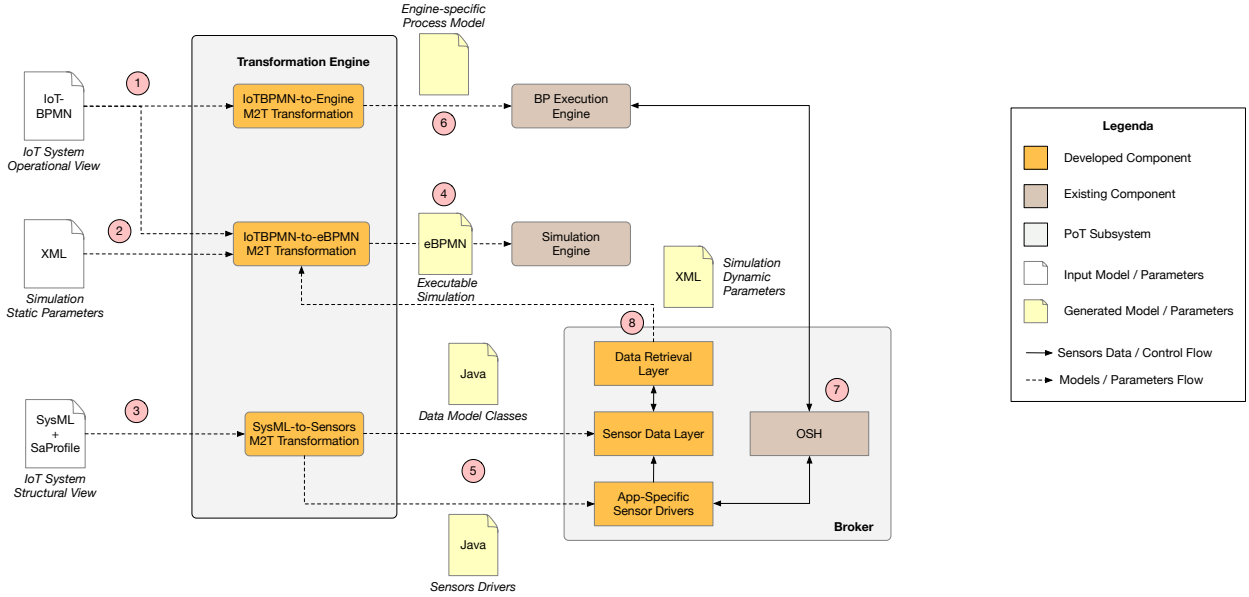


Figure 8. Methodology for the IoT-aware BPM.

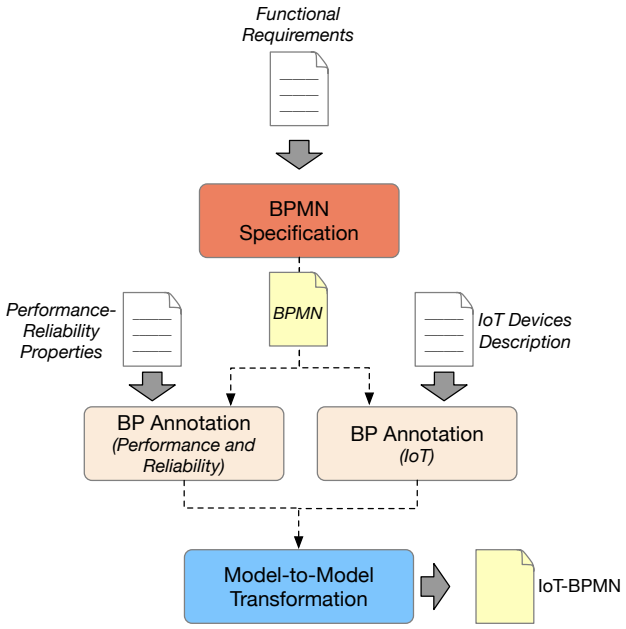


Figure 9. Specification of PyBPMN and IoT-BPMN.

5. Implementation of the IoT-Aware Framework Prototype

As discussed in Section 4.5, the framework architecture includes five main components providing the capabilities to (i) specify a BP model (BP Modeling and Management component), (ii) run the BP (Execution Engine), (iii) execute the required model transformations (Model Transformation Engine), (iv) communicate with the underlying sensors network (Broker), and (v) simulate the BP by executing the relevant simulation model which plays the role of DT (Simulation Engine).

This section describes how the framework prototype has been developed. As illustrated in Figure 7, part of the framework capabilities are provided by existing open source components. The core technologies at the basis of prototype implementation are the Java programming language and the Eclipse Modeling Project [42], an Eclipse component that provides concrete tools to enable the actual adoption of model-driven software engineering approaches.

The core of the Eclipse Modeling Project is the Eclipse Modeling Framework (EMF) [43], which constitutes the backbone of the proposed framework implementation. It provides:

- *Ecore*, the Eclipse reference implementation of OMG's Essential-MOF (EMOF) [30], the OMG standard for the specification of neutral metamodels;
- A visual environment to create, manage and store *metamodels* serialized as XML Metadata Interchange (XMI) data [31].

According to the framework architecture, the prototype provides various *model-to-model* and *model-to-text* transformations. In this respect, the following technologies of the Eclipse Modeling Project have been used:

- *QVT Operational (QVTo)*, which is an implementation of the QVT Operational Mappings language issued by OMG. QVTo provides a transformation engine to execute *model-to-model* transformations on Ecore-based models. Specifically, QVTo allows the transformation of an *input model*, which conforms to the relevant *source metamodel*, to an *output model*, which is in turn an instance of the *target metamodel*. Both the *source* and *target* metamodels have to be specified by use of of Ecore [44] and serialized as XML schemas according to XMI rules [44];
- *Acceleo*, which is an Eclipse plugin that implements the OMG MOFM2T standard. Acceleo provides a *model-to-text* transformation engine which takes as input an XMI model that conforms to a given Ecore-based source metamodel and yields as output a text document. The latter is obtained by use of a *template-based approach*, where fixed text is completed with the information retrieved from the input model [39].

The aforementioned technologies and standards are at the basis of the prototype framework, which consists of a set of components described hereinafter.

The implementation of both the *BP Modeling and Management* component and *BP Execution Engine* makes use of the open-source Camunda Platform, version 7 [45], which provides a BPMN visual editor and a BP execution engine.

As regards the *Transformation Engine*, an EMF- and Java-based implementation of the following transformations have been developed:

- **BPMN-to-IoTBPMN Model-to-Model Transformation**, which takes as input the annotated BPMN model (serialized as an XML file) and yields as output an XMI-based IoT-BPMN model. Specifically, the transformation maps the annotation elements that describe IoT devices, as well as the performance and reliability properties of the BP under study (e.g., the sensor url endpoint, the device MTTF, etc.), to the corresponding IoT-BPMN elements. As an example, the following annotations is used to generate an output model which includes a *SensorTask* element where the *serviceEndPoint* attribute assumes the value `http://utv.testserviceurl.it/sensor1`. In order to be handled by the model transformation engine, the annotations must conform to the EBNF (Extended Backus-Naur Form) formal syntax, as illustrated in [41].

```
<<Sensor>> {
serviceEndPoint = http://utv.testserviceurl.it/sensor1 ,
}
```

- **IoTBPMN-to-eBPMN Model-to-Text Transformation**, which takes as input the IoT-BPMN model and yields as output the eBPMN code that constitutes the Java-based implementation of BP digital twin. It should be underlined that this transformation takes into consideration the performance- and reliability-related properties of the IoT-BPMN model. Therefore, the framework prototype uses the transformation implementation introduced in [41];
- **SysML-to-Sensor Model-To-Text Transformation** which constitutes the core of the proposed approach. This transformation takes as input the annotated SySML model, which represents the structural view of the IoT system, and generates the Java classes implementing the application-specific OSH sensor drivers, along with the relevant data model. In this respect, in order to generate the app-specific sensor drivers, the

transformation analyzes the SysML model and, for each class stereotyped as «*Sensor*», yields as output the following classes:

- *Sensor Configuration class*, which provides the attributes to store the parameters that characterize the sensor configuration. If the SysML model specifies the sensor location (i.e., throughout the *position* attribute provided by «*Sensor*» or «*Platform*» stereotypes, the transformation includes such information in the sensor configuration;
- *Sensor Class*, which constitutes the actual implementation of the sensor driver. According to the OSH architecture, it implements the *ISensorModule* interface. The transformation generates the class skeleton which includes the code to handle the sensor identifiers. The skeleton also includes the appropriate specification and initialization of the methods inherited from the *ISensorModule* interface for accessing the measured sensor data. A manual refinement is thus required for the complete implementation of the driver;
- *Sensor Output Class*, which provides methods to allow access to measured data. In this respect, the sensor output data structure and the encoding to be adopted are generated according to the structure of the SyML model elements. Specifically, the transformation takes into account those elements stereotyped as «*ObservableProperties*», «*SensorData*», «*SensorDataArray*» and «*CategorySensorData*».

Finally, in order to obtain the implementation of the sensor data model, the transformation generates one separated class for each SysML model element stereotyped as «*FeatureOfInterest*». It is worth noting that the related properties are typed according to the SysML model elements annotated with stereotypes that inherit from «*BaseSensorData*».

The implementation of the *Broker* consists of various components. As described in Section 4.5, the core of the *Broker* implementation is represented by the open-source project Open Sensor Hub (OSH), which provides a service-based interface for interacting with IoT devices. The OSH implementation includes a set of *OSH Basic Drivers* that enact the data exchange with generic devices. The *App-specific Sensor Driver* component includes the implementation of the various drivers required for interacting with the actual sensors used in the specific application. As stated above, such drivers constitute the output of the `SysML-to-Sensor` *model-to-text* transformation. An example is given in Section 6. Similarly, the Java-based implementation of *Sensor Data* which represents the Broker data model, is dynamically obtained from the structural model of the sensors network by use of the same `SysML-to-Sensor` *model-to-text* transformation (see Figure 8 step 5). In addition, as clarified in Section 6, the experimentation has addressed steps 1 to 7 of the proposed methodology. As such, the current prototype does not include an implementation of the *Data Retrieval Layer*.

Finally, as regards the *Simulation Engine*, the prototype implementation includes the Java-based domain-specific simulation framework *eBPMN*, which is fully described in [37,38].

The next section illustrates the experimentation carried out to assess the feasibility and effectiveness of the proposed approach.

6. Example Application

This section provides an example that aims to demonstrate how the proposed IoT-aware BPM framework can be used in practice. The example application makes use of a framework prototype implementation that addresses the methodology steps 1 through 7.

In order to assess the technically soundness of the proposed IoT-aware BPM framework and to assess the feasibility and effectiveness of the relevant methodology along with its limitations, this section discusses the application of a framework prototype to a concrete case. Specifically, the prototype implementation of the framework addresses steps 1 through 7 of the proposed methodology.

Let us consider a *Water Monitoring System (WMS)*, which controls a sensors network for evaluating the flooding risk of a water basin. The system includes the following sensors:

- The water level and its vertical acceleration are monitored by a *GPS* and an *accelerometer*, both mounted on a buoy which floats on the water surface;
- The weather conditions are monitored by a weather station equipped with a *pluviometer/anemometer*.

The control flow executed by the WMS IoT system is summarized as follows:

1. All sensors have to be initialized;
2. Iteratively, the available sensors have to be inquired in order to obtain actual measures for the observed parameters: the water level, its vertical speed, the rainfall and the wind speed and direction;
3. According to the collected data, the system shall evaluate the flooding risk and, if required, an alert shall be sent to a human operator in charge of enacting the required countermeasures and properly managing the emergency operations.

As illustrated in Figures 8 and 9, the first step of the proposed methodology deals with the specification of an IoT-BPMN process model. In this respect, according to the functional requirements, a BPMN diagram is initially defined. Then, the non-functional requirements are used for extending the BP model with textual annotations compliant to the IoT-BPMN metamodel. Finally, a *model-to-text* transformation is executed to generate the XML-based representation of the IoT-aware BP.

Figure 10 illustrates the BPMN model for the Water Alert System. A fragment of the annotated model is shown in Figure 11.

At the second step, the *structural view* of the WMS is specified in terms of a SysML diagram extended with the *IoT4SysML* profile.

The structural model consists of various diagrams. The package diagram describes the various domain entities, as shown in Figure 12, which shows how the WMS entity, which acts as a `«system»`, observes the Basin.

The use of the *IoT4SysML* profile allows the identification of relevant concepts which are used by the model transformations for generating the corresponding output:

- Blocks representing IoT devices, e.g., the GPS or the accelerometer stereotyped as `«Sensor»` (see Figure 13);
- The structure of the data collected by a sensor. As an example, the attributes *rainFall* and *wind* owned by the *Pluvio-Anemometer* Block are stereotyped as `«SensorData»` and `«SensorDataRecord»`, respectively. While the former denotes a *basic* (i.e., Integer) datatype, the latter is a record type, further specified by the *WindDataTYpe* element (see Figure 13);
- The relationship between a *sensor* and the observed *features of interest*. As an example, the *GPS «observes»* the *weather conditions* (see Figure 13);
- The observable properties characterizing each feature of interest. As an example, the *Basin Water Condition* block is stereotyped as a `«FeatureofInterest»` which in turn is specified by two attributes stereotyped as «ObservableProperty»: the *waterLevel* and the *verticalAcceleration* (see Figure 14).

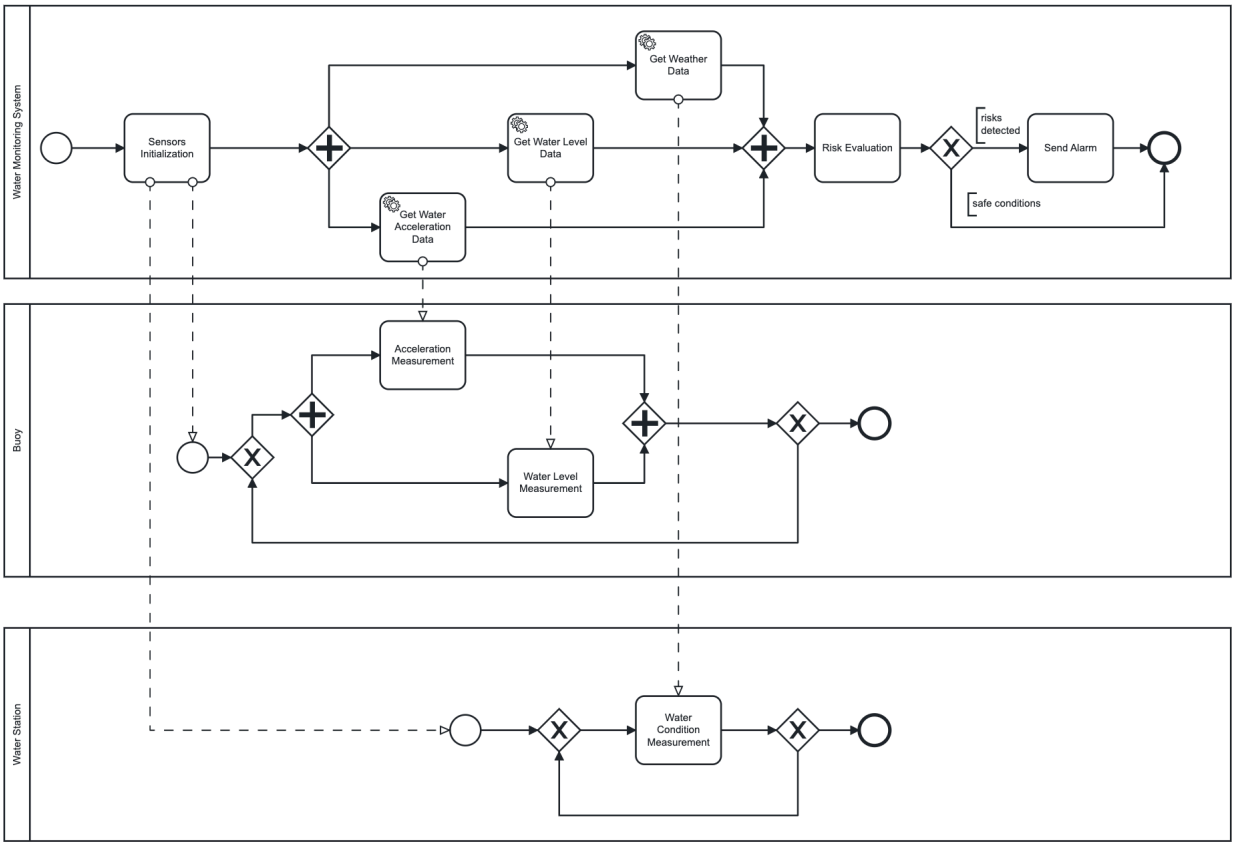


Figure 10. BPMN Model for the Water Alert System.

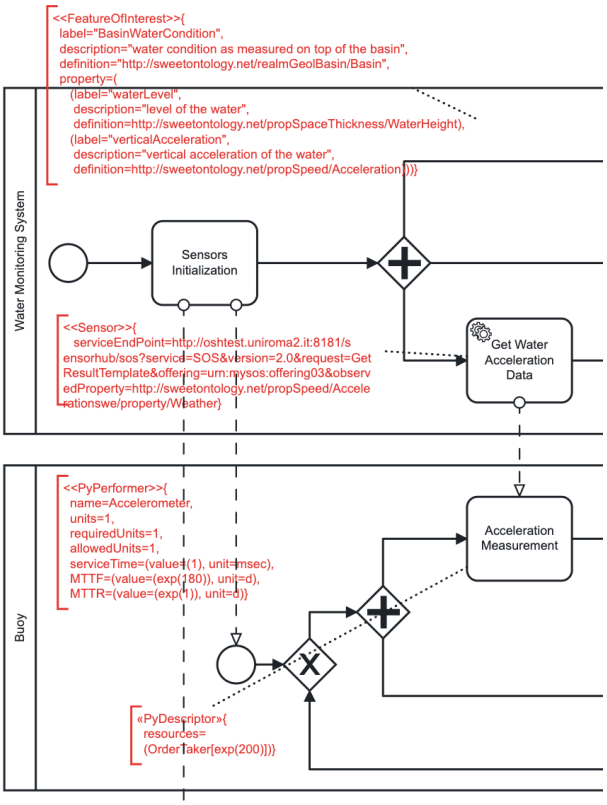


Figure 11. Annotation of the BPMN model with performance- and IoT-related properties (fragment).

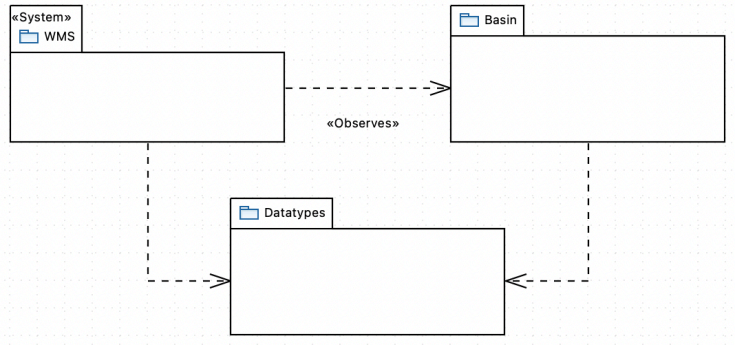


Figure 12. Water Alert System overview.

The WMS package contains the Block Definition Diagram (BDD) specifying the WMS structure (see Figure 13). In addition, the BDDs contained in the Basin package and in the Datatypes package are shown in Figures 14 and 15, respectively.

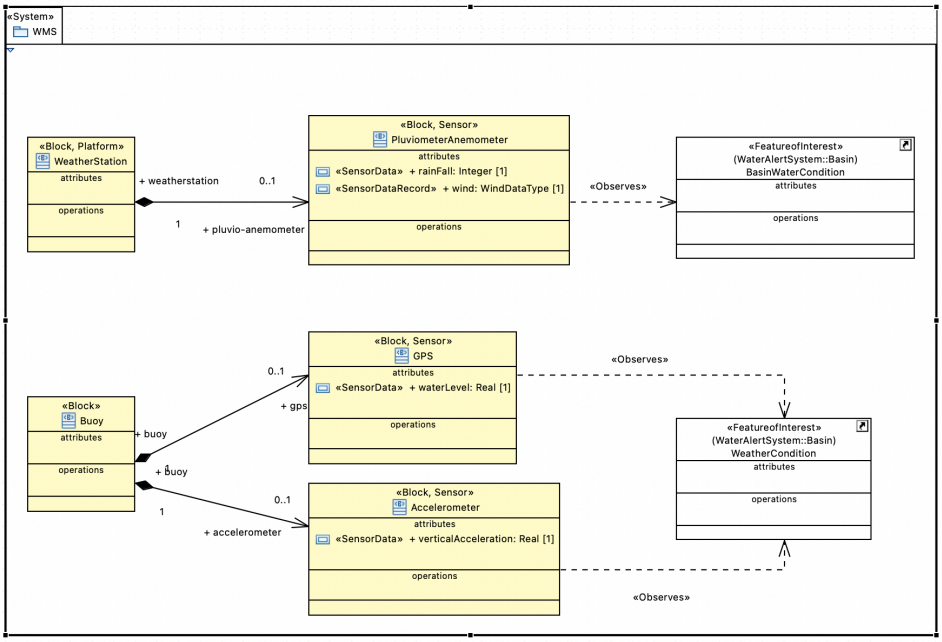


Figure 13. WAS Package: structure of the Water Monitoring System.

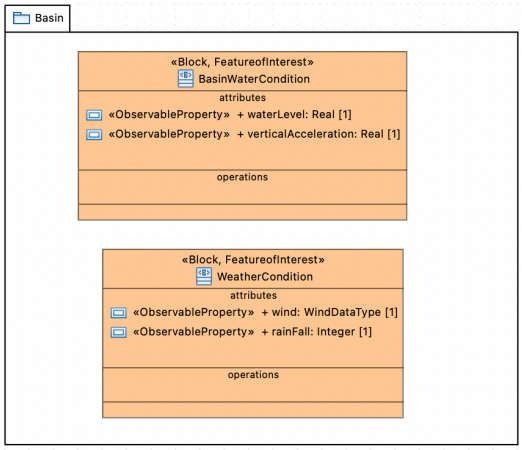


Figure 14. Basin Package: WMS Features of Interest.

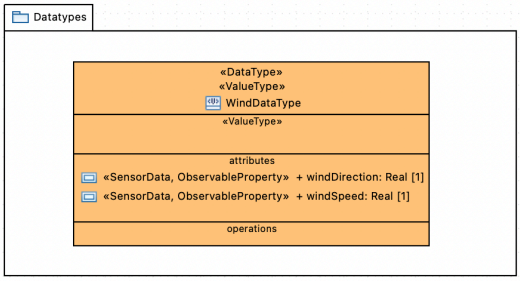


Figure 15. Datatypes Package.

Steps 4 to 6 of the methodology deal with the implementation of the executable BP process (i.e., the physical twin) and the related eBPMN simulation model (i.e., the digital twin).

In this respect, an example of how the *IoTBPMN-to-Engine* transformation adapts the IoT-aware BP model to make it compliant to Camunda is given in Figure 16, while the output of the *IoTBPMN-to-eBPMN* transformation is provided in Figure 17. Finally, Figure 18 shows an example of how the *IoTBPMN-to-sensor* is used for supporting the generation of the broker's Sensor Data Layer, as well as the required sensor's Drivers.

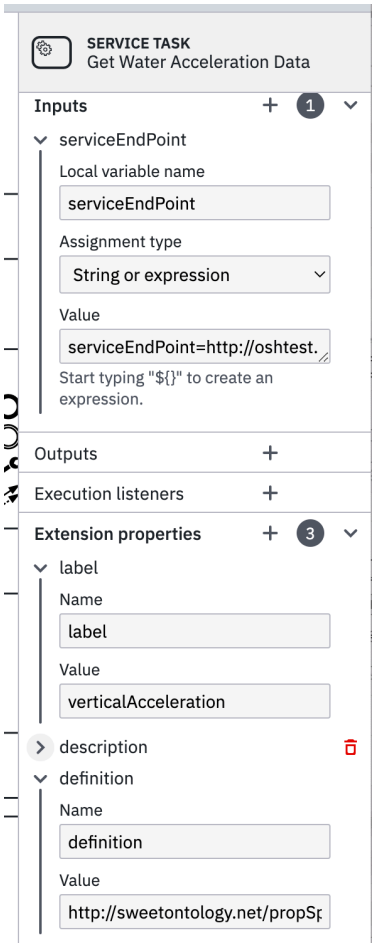


Figure 16. Specification of IoT-related properties in Camunda.

Figure 17. Fragment of the IoT-ware BP Digital Twin implemented in eBPMN-based Java code.

It is worth noting that the experimentation was not intended to be based on a test campaign and on the collection of measures to evaluate the performance- or reliability-related properties of the prototype, or the software/hardware components of the Water Alert System. The experimentation does not include the validation of the simulation model, as it has been investigated in a previous contribution (e.g., [19]).

Differently, the experimentation has allowed us to:

- Assess the completeness and effectiveness of the IoT-BPMN extension and the IoT4SysML profile to model a simple but concrete IoT System;
- Assess the usefulness and the usability of the IoT4SysML profile;
- Verify how the use of a model-transformation approach is feasible and effective to support the development of sensor drivers and the data model;
- Define the procedure through which the drivers and the data model can be deployed at run-time to the extend middleware capabilities;
- Verify how the adoption of a model-transformation approach eases the development of a digital twin, by generating the relevant code from design models of the corresponding physical system.

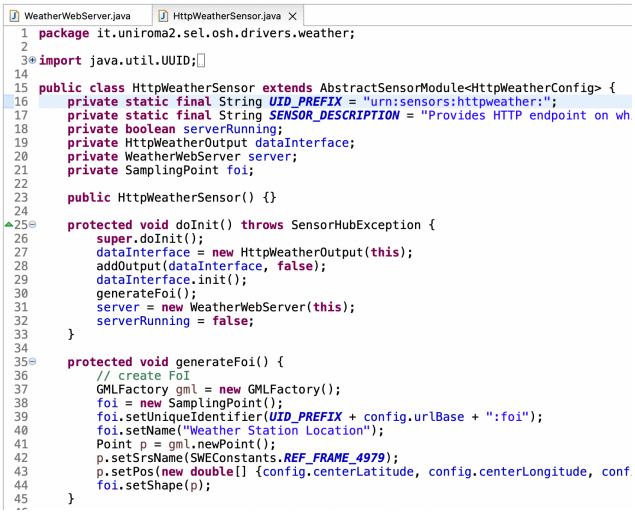


Figure 18. Fragment of the Weather Sensor Driver implementation.

7. Conclusions

This paper has introduced a framework for the simulation-based analysis of IoT-aware BPs. The strength of the proposed approach lies in the adoption of MDA-based standards

and technologies, which effectively ease the implementation of executable simulations associated to the actual BP state, according to a digital twin perspective. Moreover, the adoption of MDA has also allowed the development of a framework flexible enough to be used in various operational contexts. In this respect, the broker is virtually compliant with any IoT device, as the provided model transformations are able to effectively support the implementation of both the required drivers and the classes implementing the sensors' data models. Similarly, no assumptions have been made about the software environment used for specifying and executing the BP model, as the methodology includes an adaption step in which the IoT-BPMN model is translated to fit the actual process engine. In order to preliminarily assess the validity of the proposed approach, a framework prototype, based on the Camunda BPM platform, has been implemented. red According to the results collected from prototype experimentation, the use of the framework in a concrete case has demonstrated the feasibility of the approach for the simulation-based analysis of IoT-aware BPs. The experimentation has also allowed us to verify the completeness and effectiveness of the proposed modeling approach to represent a sensors network from operational and a structural perspectives, respectively. Finally, the adopted model transformations have shown their effectiveness to promote a *low-code* development approach, which reduces the effort required to implement the system. On the other hand, the experimentation has shown that the adoption of an MDE paradigm does not completely relieve developers from the need of refining and completing the generated code. Moreover, the annotation of a SysML model requires specific knowledge of the adopted profile both from a semantic and a syntactic point of view. Finally, even though the experimentation has demonstrated that the use of an IoT-BPMN compliant visual tool is not strictly required, its unavailability can be considered as a limitation to address in future work. Ongoing work includes the development of a full-fledged implementation of the framework, which specifically addresses the dynamic and automated update of the DT at execution time, according to state changes of the actual system.

Author Contributions: Conceptualization, P.B. and A.D.; Investigation, P.B. and A.D.; Methodology, P.B. and A.D.; Software, P.B. and T.P.; Supervision, A.D.; Validation, P.B. and A.D.; Writing—original draft, P.B., A.D. and T.P.; Writing—review & editing, P.B. and A.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The study did not employ or report any data.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Rose, K.; Eldridge, S.; Chapin, L. The internet of things: An overview. *Internet Soc. (ISOC)* **2015**, *80*, 1–50.
2. Gianni, D.; D'Ambrogio, A.; Tolk, A., Eds. *Modeling and Simulation-Based Systems Engineering Handbook*; CRC Press: Boca Raton, FL, USA, 2014.
3. Ojbect Management Group. MDA Guide, Revision 2.0 (ormsc/14-06-01). 2003. Available online: https://www.omg.org/cgi-bin/doc?ormsc/14-06-01.pdf (accessed on 20 December 2022).
4. Bocciarelli, P.; D'Ambrogio, A.; Falcone, A.; Garro, A.; Giglio, A. A Model-Driven Approach to Enable the Distributed Simulation of Complex Systems. In *Proceedings of the Complex Systems Design & Management*; Auvray, G., Bocquet, J.C., Bonjour, E., Krob, D., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 171–183.
5. Madni, A.M.; Madni, C.C.; Lucero, S.D. Leveraging Digital Twin Technology in Model-Based Systems Engineering. *Systems* **2019**, *7*, 7. [CrossRef]
6. Peterson, J.L. Petri nets. *ACM Comput. Surv. (CSUR)* **1977**, *9*, 223–252. [CrossRef]
7. van der Aalst, W.; ter Hofstede, A. YAWL: Yet another workflow language. *Inf. Syst.* **2005**, *30*, 245–275. [CrossRef]
8. Ryan, J.; Heavey, C. Process modeling for simulation. *Comput. Ind.* **2006**, *57*, 437–450. [CrossRef]
9. Chinosi, M.; Trombetta, A. BPMN: An introduction to the standard. *Comput. Stand. Interfaces* **2012**, *34*, 124–134. [CrossRef]

10. Suri, K.; Gaaloul, W.; Cuccuru, A.; Gerard, S. Semantic Framework for Internet of Things-Aware Business Process Development. In Proceedings of the 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Poznan, Poland, 21–23 June 2017; pp. 214–219. [CrossRef]
11. EFPL. Global Sensor Networks. 2003. Available online: http://lsir.epfl.ch (accessed on 15 December 2022).
12. International Telecommunication Union. Architectural Reference Models of Devices for Internet of things Applications. 2019. Available online: https://www.itu.int/rec/T-REC-Y.4460/en (accessed on 22 December 2022).
13. Open Geospatial Consortium. *Java Business Process Model (jBPMN)*. 2022. Available online: https://www.w3.org/TR/vocab-ssn/ (accessed on 22 December 2022).
14. Want, R.; Schilit, B.N.; Jenson, S. Enabling the Internet of Things. *Computer* **2015**, *48*, 28–35. [CrossRef]
15. OSH Community. Opens Sensor Hub. 2022. Available online: https://opensensorhub.org/ (accessed on 22 December 2022).
16. Open Geospatia Consortium. Sensor Web Enablement (SWE). 2022. Available online: https://www.ogc.org/node/698 (accessed on 20 December 2022).
17. Fattouch, N.; Ben Lahmar, I.; Boukadi, K. IoT-aware Business Process: comprehensive survey, discussion and challenges. In Proceedings of the 2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Bayonne, France, 10–13 September, 2020; pp. 100–105. [CrossRef]
18. Schönig, S.; Ackermann, L.; Jablonski, S.; Ermer, A. IoT meets BPM: A bidirectional communication architecture for IoT-aware process execution. *Softw. Syst. Model.* **2020**, *19*, 1443–1459. [CrossRef]
19. Petrasch, R.; Hentschke, R. Towards an Internet-of-Things-aware process modeling method. In Proceedings of the 2nd Management and Innovation Technology International Conference (MITiCON2015), Bangkok, Thailand, 16–18 November, 2015; pp. 168–172.
20. Sperner, K.; Meyer, S.; Magerkurth, C. Introducing Entity-Based Concepts to Business Process Modeling. In *Proceedings of the Business Process Model and Notation*; Dijkman, R., Hofstetter, J., Koehler, J., Eds.; Springer Berlin Heidelberg: Berlin/Heidelberg, Germany, 2011; pp. 166–171.
21. Meyer, S.; Ruppen, A.; Hilty, L. The Things of the Internet of Things in BPMN. In *Proceedings of the Advanced Information Systems Engineering Workshops*; Persson, A., Stirna, J., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 285–297.
22. Dar, K.; Taherkordi, A.; Baraki, H.; Eliassen, F.; Geihs, K. A resource oriented integration architecture for the Internet of Things: A business process perspective. *Pervasive Mob. Comput.* **2015**, *20*, 145–159. [CrossRef]
23. Schönig, S.; Ackermann, L.; Jablonski, S.; Ermer, A. An Integrated Architecture for IoT-Aware Business Process Execution. In *Proceedings of the Enterprise, Business-Process and Information Systems Modeling*; Gulden, J., Reinhartz-Berger, I., Schmidt, R., Guerreiro, S., Guédria, W., Bera, P., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 19–34.
24. Euroean Commission. IOT-A Internet of Things Architecture. 2013. Available online: https://cordis.europa.eu/project/id/257521/ (accessed on 22 December 2022).
25. Red Hat. Java Business Process Model (jBPMN). 2022. Available online: http://www.jboss.org/jbpm/ (accessed on 20 December 2022).
26. Jones, D.; Snider, C.; Nassehi, A.; Yon, J.; Hicks, B. Characterising the Twin: A systematic literature review. *CIRP J. Manuf. Sci. Technol.* **2020**, *29*, 36–52. [CrossRef]
27. Singh, M.; Fuenmayor, E.; Hinchy, E.P.; Qiao, Y.; Murray, N.; Devine, D. Digital Twin: Origin to Future. *Appl. Syst. Innov.* **2021**, *4*, 36. [CrossRef]
28. Negri, E.; Fumagalli, L.; Macchi, M. A Review of the Roles of Digital Twin in CPS-based Production Systems. *Procedia Manuf.* **2017**, *11*, 939–948. [CrossRef]
29. Atkinson, C.; Kühne, T. Model-driven development: A metamodeling foundation. *IEEE Softw.* **2003**, *20*, 36–41. [CrossRef]
30. Object Management Group. *Meta Object Facility (MOF) Core Specification*. 2013. Available online: http://www.omg.org/MOF/ (accessed on 20 December 2022).
31. Object Management Group. *XML Metadata Interchange (XMI) Specification*. 2011. Available online: https://www.omg.org/spec/XMI/ (accessed on 20 December 2022). .
32. Object Management Group. *Meta Object Facility (MOF) 2.0 Query/View/Transformation*. 2015. Available online: http://www.omg.org/spec/QVT/ (accessed on 20 December 2022).
33. OMG. *MOF Model to Text Transformation Language (MOFM2T), 1.0*. 2008. Available online: https://www.omg.org/spec/MOFM2T (accessed on 20 December 2022).
34. Bocciarelli, P.; D'Ambrogio, A.; Wagner, G. Resource Modeling in Business Process Simulation. In *Proceedings of the 2022 Winter Simulation Conference*; Feng, B., Pedrielli, G., Peng, Y., Shashaani, S., Song, E., Corlu, C., Lee, L., Chew, T., Roeder, E., Eds.; Institute of Electrical and Electronics Engineers, Inc.: Piscataway, NJ, USA, 2022.
35. Bocciarelli, P.; D'Ambrogio, A. Performability-Oriented Description and Analysis of Business Processes. In *Business Process Modeling: Software Engineering, Analysis and Applications*; Beckmann, J.A., Ed.; Nova Science Publisher: New York, NY, USA, 2011; pp. 1–36.
36. Bocciarelli, P.; D'Ambrogio, A. A BPMN Extension for Modeling Non Functional Properties of Business Processes. In Proceedings of the Symposium on Theory of Modeling and Simulation, DEVS-TMS '11, Boston, MA, USA, 3–7 April 2011; pp. 160–168.

37. Bocciarelli, P.; D'Ambrogio, A.; Paglia, E. A Language for Enabling Model-Driven Analysis of Business Processes. In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development*, Lisbon, Portugal, 7–9 January 2014; SciTePress: Lisbon, Portugal, 2014.
38. Bocciarelli, P.; D'ambrogio, A.; Giglio, A.; Paglia, E.; Gianni, D. A Transformation Approach to Enact the Design-Time Simulation of BPMN Models. In Proceedings of the IEEE 23rd International WETICE Conference, Parma, Italy, 23–25 June 2014; pp. 199–204. [CrossRef]
39. Eclipse Foundation. *Acceleo*. 2012. Available online: https://www.eclipse.org/acceleo/ (accessed on 20 December 2022).
40. Scowen, R.S. Extended BNF—A generic base standard. In Proceedings of the Software Engineering Standards Symposium, Lake Buena Vista, FL, USA, 1–5 November 1998; Volume 3, pp. 6–12.
41. Bocciarelli, P.; D'Ambrogio, A.; Giglio, A.; Paglia, E. BPMN-Based Business Process Modeling and Simulation. In *Proceedings of the 2019 Winter Simulation Conference*; Mustafee, N., Bae, K.H.G., Lazarova-Molnar, S., Rabe, M., Szabo, C., Haas, P., Son, Y.J., Eds.; Institute of Electrical and Electronics Engineers, Inc.: Piscataway, NJ, USA, 2019; pp. 1439–1453. [CrossRef]
42. Eclipse Foundation. *Eclipse Modeling Project*. 2016. Available online: https://eclipse.org/modeling (accessed on 20 December 2022).
43. Eclipse Foundation. *Eclipse Modeling Framework (EMF)*. 2016. Available online: https://eclipse.org/modeling/emf/ (accessed on 20 December 2022).
44. Eclipse Foundation *QVT Operational Project*. 2016. Available online: http://projects.eclipse.org/projects/modeling.mmt.qvt-oml (accessed on 19 January 2022).
45. Camunda. *Camunda Platfom 7*. 2022. Available online: https://camunda.com/platform-7/ (accessed on 20 December 2022).