International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 16 (2024) pp.677–695

Article

A Model-Driven Approach for IoRT-Aware Business Process Engineering

Najla Fattouch [1,†,*], Imen Ben Lahmar [2,†] and Khouloud Boukadi [1,†]

[1] FSEG Sfax, MIRACL Laboratory, Sfax University, Sfax 3029, Tunisia; khouloud.boukadi@fsegs.usf.tn
[2] ISIM Sfax, ReDCAD Laboratory, Sfax University Sfax 3029, Tunisia; imen.benlahmar@isims.usf.tn
[†] These authors contributed equally to this work.
[*] Correspondence author: fattouchnajla@gmail.com

Received date: 5 March 2024; Accepted date: 21 March 2024; Published online: 10 July 2024

Abstract: Recently, industries recognized the urgency of a new business generation called IoRT-aware Business Processes (BPs), which automates BPs using Internet of Robotics Things (IoRT) technology. According to the literature, an IoRT-aware BP can be modeled using numerous languages, such as the Business Process Model and Notation (BPMN). Nonetheless, the BPMN lacks a formal aspect, potentially leading to inconsistencies and ambiguities. Therefore, to effectively execute an IoRT-aware BP modeled using BPMN and scrutinize their behaviors, adopting the Model Driven Engineering (MDE) approach appears essential. With MDE, a BP can be modeled and transformed into a consistent formalism (e.g., Petri Net (PN), Discrete Event System Specification (DEVS), etc.), simplifying its verification. During this work, we consider the MDE approach to deal with the modeling, transformation, and checking of an IoRT-aware BP. From this perspective, we start, on the one hand, by proposing a meta-model for the IoRT-aware BP based on a top-down approach and, on the other hand, by extending the existing DEVS meta-model. The DEVS formalism is characterized by its benefits, marked by its hierarchical structure that facilitates the whole system representation and ensures model reuse. Using these meta-models, we propose during this work a model-to-model transformation that facilitates the conversion of an IoRT-aware Business Process (BP) into a DEVS model. After that, we address the checking of the IoRT-aware BP model structure based on a set of anti-patterns. To evaluate the proposed approach, we consider a case study of IoRT-aware BP in the agriculture field by checking its process model structure. The findings demonstrate the effectiveness of our approach in capturing structure errors.

Keywords: formal verification; process structure; model transformation; MDE; anti-pattern; IoRT-aware business process

1. Introduction

Recently, the IoRT-aware Business Process has emerged regarding the automation of classic BP using the Internet of Robotic Things (IoRT). This process automates classic BP, where its tasks are accomplished automatically using the Internet of Things (IoT) and robot devices rather than humans [1].

Numerous modeling languages can be used to model a BP. The Business Process Model and Notation (BPMN) sets an effective standard for modeling processes, characterized by its simple and understandable notations. Furthermore, the BPMN can be harnessed to model an automated process using emerging technologies (e.g., IoRT, robotics, etc.). Nonetheless, this modeling language has some limits, such as lacking a formal aspect, which might cause inconsistencies, ambiguities, and incompleteness in the developed processes [2]. Therefore, numerous researchers propose some approaches to formalize BP models. However, most do not deal with a complex process (e.g., IoRT-aware BP, etc.) [2]. To bridge these gaps, it is relevant to consider an expressive modeling formalism that allows the modeling of complex processes.

During this work, we choose the DEVS formalism as a formal structure used to model and simulate dynamic systems/processes regarding its hierarchical structure [3]. The DEVS is distinguished by its capacity to articulate the system's development by outlining its behaviors via transition functions, state variables, configuration of model interactions, etc. [3].

Several works address the modeling of the BP and its transformation into DEVS (e.g., [3, 4], etc.). Nonetheless, during the proposed transformation, the authors do not deal with the model-driven engineering (MDE) approach. This latter is regarded as one of the methods for software development that has demonstrated advantages for creating systems. Furthermore, the proposed approaches fail to account for endorsed meta-models, neither in the process nor in the DEVS.

During this research work, we address the transformation of an IoRT-aware BP into DEVS formalism. The proposed transformation relies on the MDE approach, which provides a simulation framework that facilitates the modeling and transformation of a BP into a formal representation. The proposed transformation is based on an IoRT-aware BP meta-model (source) and a DEVS one (target). Referring to the literature, we note a lack of approaches dealing with the modeling of an IoRT-aware BP. Therefore, our extensive literature exercise revealed the use of the approach [5], where the authors tackled the meta-modeling of the IoRT-aware BP. However, this work neglects a meta-modeling approach when building the authors' proposals. Consequently, an overload of information (e.g., concepts, attributes, relations, etc.) within the meta-model appears, which makes it unreliable.

To close the process modeling gaps, we propose a meta-model for an IoRT-aware BP based on the top-down approach. This approach guarantees the meta-model consistency by well-fixing its concepts, which avoids overloading within the meta-model. Additionally, dealing with a top-down often assists in the definition of the relationship between the heterogeneous meta-model concepts, aiming to reduce its ambiguity [6].

Using the IoRT-aware BP meta-model, we propose a model-to-model transformation approach based on a set of rules to transform the source (IoRT-aware BP) meta-model into a target (DEVS) one. More precisely, we focus on the checking of an IoRT-aware BP model structure. Consequently, the checking of the process structure model intends to avoid process model issues and guarantee its correct execution. In this paper, we extend our previous work proposed in [7]. Therefore, the process-checking structure based on a set of identified anti-patterns is addressed. Being aware of model anti-patterns can detect and prevent the occurrence of undesirable constructs within the process model [8]. Additionally, an anti-pattern acts as a warning sign, showing the weaknesses and issues that can slow the process. To demonstrate the efficacy of our proposition, we consider a case study of an IoRT-aware BP in the agriculture field by examining its structure. The results show that our proposal is effective in structure error detection.

The remainder of this paper is structured as follows: Section 2 presents the related works. Section 3 details the proposed IoRT-aware Business Process and DEVS Meta-modeling. The proposed model-to-model transformation of an IoRT-aware BP into DEVS is depicted in Section 4. Section 5 describes the proposed IoRT-aware BP structure verification using DEVS formalism. The experimentation of the proposed approach is presented in Section 6. Finally, the closing Section 7 outlines the main axes of our work and highlights its future directions.

2. Related Work

This section overviews recent approaches dealing with modeling, transforming, and verifying BPs. Hence, we considered a set of criteria, which are detailed in what follows:

- Business: Stands for the type of BP. This criterion allows us to fix the most considered process type used in process modeling, transformation, or verification. Indeed, it can be a classic process or an automated one.
- Standard: Sets forward the source for the used concepts that are integrated within the BP meta-model. This criterion aims to identify the most used modeling language or formalism for the BP.
- Meta-modeling approach: Indicates the set of methods considered during creating a meta-model. Referring to the literature, several approaches were proposed: top-down, bottom-up, etc. These methods are generally used to avoid overloading the concepts that are used.
- Formalism: Designates the formal method utilized for process verification. Numerous formalisms have been developed in the literature, including DEVS, Petri Net (PN), etc.
- Anti-pattern: Gives the undesirable structures for a BP model. The consideration of the anti-patterns is crucial for ensuring the effectiveness of process workflows.
- Checked dimension: Identifies the checked BP behaviors.
- Transformation method: Gives the set of approaches (e.g., MDE, artificial intelligence, etc.) used during the meta-model transformation.

Based on the selected criteria, a set of published approaches that address BP modeling, transformation, and verification are detailed in Table 1.

Table 1. Comparison of the studied approaches based on a set of criteria.

Papers	Business	Standard	Meta-Modeling Approach	Formalism	Anti-Pattern	Checked Dimension	Transformation Method
[9]	Classic BP	-	-	Petri net	-	Structure	-
[10]	Classic BP	ISO	-	Petri net	-	-	-
[11]	Classic BP	-	-	Petri net & TLA	-	Temporal	-
[12]	Classic BP	-	-	-	-	Structure	Artificial intelligence
[7]	IoRT-aware BP	BPMN	-	DEVS	-	Temporal	MDE
[13]	classic BP	BPMN	-	TLA	-	Spatial	-
[14]	-	-	-	DEVS	-	-	MDE
[2]	Classic BP	BPMN	-	ECATNet	-	Structure	-
[8]	Classic BP	BPMN	-	-	✓	Structure	-
[5]	IoRT-aware BP	BPMN & ISO	-	-	-	-	-
[15]	Classic BP	BPMN	Top-down	-	✓	-	-
[16]	Classic BP	BPMN	-	Petri net	-	Structure	-
[17]	Classic BP	BPMN	-	-	-	Data	-
[18]	-	-	-	Petri net	-	Structure	-
[19]	Classic BP	GPML	-	-	-	Structure, Semantic	MDE
[20]	Classic BP	BPMN	-	-	-	-	-
[21]	Classic BP	-	-	-	-	Event logs	-
[22]	IoT system	-	-	-	-	-	-
[23]	Classic BP	BPMN	-	-	-	-	MDE

By examination of Table 1, we note that most of the approaches, for instance, [2, 8, 9, 10, 11, 12, 13, 16, 17, 19, 20, 21, 23], dealt with the modeling, transformation, or verification of a classic BP. Among these works, we pointed out that only [5] addressed the modeling of an IoRT-aware BP. Also, the only approach presented in [7] tackled verifying an IoRT-aware BP. Additionally, we deduced from Table 1 that most of the studied methods (e.g., [2, 7, 8, 13, 16, 17, 20, 23], considered Business Process Modeling and Notation (BPMN) as an effective standard for the BP. This modeling language is widely used in the BP field [24]. Furthermore, we observed that a few works, such as [15], have considered the meta-modeling approach during their proposal development.

Moreover, several approaches such as: [2, 9, 10, 11, 16, 18], are based on the Petri Net formalism when dealing with process behaviors checking. Nonetheless, classical Petri Net lacks a hierarchical structure that impedes the modeling and checking of complex processes (e.g., IoRT-aware BP, etc.). During the transformation of a BP into such formalism (e.g., Petri Net, DEVS, etc.), we inferred that numerous approaches (e.g., [2, 8, 9, 10, 11, 13, 16, 17, 20, 21], etc.) do not take advantage of the MDE to ensure the model transformation, despite its ability to simplify complex transformations, reduce errors, automate repetitive tasks, etc.

In addition, we equally noticed from Table 1 that most of the studied works (e.g., [7, 11], etc.) dealt with the checking of the process structure. In fact, addressing the checking process structure is crucial before verifying other process dimensions. Indeed, dealing with the BP structure verifies whether the latter depicts a convenient structure.

In this setting, the anti-pattern models can be addressed, where they detect and prevent the process of undesirable constructs. Referring to our literature study, we note that most approaches neglected the checking of a BP using anti-patterns (e.g., [8, 15]). However, these anti-patterns have a great capacity to detect an incomplete and inaccurate process structure.

To bridge the existing approach gaps, we introduce through this work a meta-model for the IoRT-aware BP using the top-down approach. Subsequently, we leverage the MDE approach to transform the proposed IoRT-aware BP into DEVS formalism. After that, we consider an IoRT-aware BP modeled using DEVS to check its structure behaviors and to guarantee its internal correct structure. The proposed process checking is based, on one hand, on the DEVS formalism and, on the other hand, on the identification of anti-patterns.

3. IoRT-Aware Business Process and DEVS Meta-Modeling

During this work, we focus on integrating IoRT technology into BPs. Therefore, we propose a meta-model for an IoRT-aware BP. After that, we use the Atlas Transformation Language (ATL) to transform the proposed meta-model into a DEVS one. In fact, DEVS provides a formal presentation of discrete event systems, which may be better suited for modeling some complex systems regarding other general formalisms, etc. Subsequently, we verify the structural behaviors of the IoRT-aware BP, modeled using DEVS, based on a set of anti-patterns.

Figure 1 shows an overview of the suggested approach.

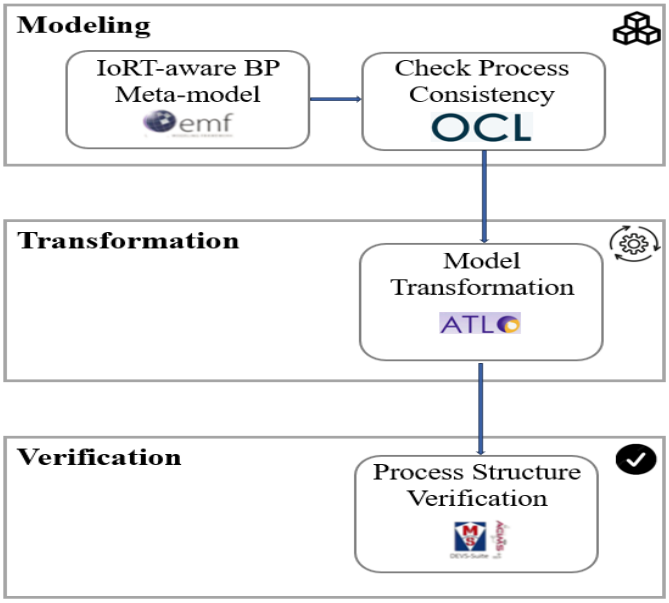

Figure 1. General overview of the proposed approach.

3.1 IoRT-Aware Business Process Meta-Modeling

In the context of Industry 4.0, the IoRT represents an attractive paradigm that serves to automate tasks and supply real-time data. Using the IoRT, a business manager can reduce productivity costs. In fact, the automation of process tasks using the IoRT technology saves the costs of manual tasks. Moreover, integrating the IoRT within the BP seeks to enhance the enterprises' competitiveness by improving product quality, increasing enterprise efficiency, etc.

Back to the literature, we deduce that there is a lack of approaches that address the integration of the IoRT within the traditional BP. In fact, in [5], the authors incorporate the IoRT technology within the classic BP, proposing a meta-model called IoRT-aware BP2M. In this setting, it is important to mention that the authors do not tackle a rigorous meta-modeling approach. Moreover, we point out that the authors' proposal has an overload of integrated concepts.

Using a meta-modeling approach is crucial to ensure credibility, and robustness, and to avoid concept overload. Consequently, automating a classic process using the IoRT technology, it is significant to consider a meta-modeling approach.

Referring to the literature, several meta-modeling approaches are developed, such as the top-down and bottom-up approaches [25]. In this work, we seek to take advantage of the top-down which is considered among the most attractive approaches. In fact, it offers a major benefit by allowing an early distinction between meta-model key

concepts. This helps designers to focus on the fundamental aspects of the studied field. Additionally, the early identifying concepts ensure a deeper understanding of the interaction between them.

Figure 2 presents an overview of the top-down approach steps.

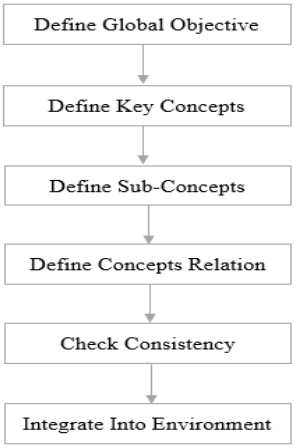

Figure 2. Top-down approach steps.

In the subsequent sections, we detail how we investigate the top-down approach to propose a meta-modeling for integrating IoRT within the BP.

3.1.1 Define Global Objective

This step helps the designer to identify the main meta-model objectives. During this work, we aim to construct a meta-model for integrating the IoRT technology within the classic BP. Our proposal seeks to give an abstract representation of the key concepts in the automated BP using IoRT technology. Moreover, it intends to guarantee the meta-model extensibility by adding new concepts, relations, attributes, etc.

3.1.2 Define Key Concepts

This step involves the identification of the meta-model core concepts. It ensures that the established concepts are aligned with the defined meta-model objectives already fixed in the previous step. During this work, we cope with the meta-model for integrating the IoRT within the classic BP (see Figure 3), which is an extension of the BPMN 2.0 meta-model. Our proposal integrates both BP, IoT, and robotic concepts.

To ensure a consistent and coherent representation of the meta-model, it is useful to consider standardized concepts. We use Business Process Modeling and Notation (BPMN) as an international standard to define the BP key concepts. This standard was developed by the Object Management Group (OMG). It offers an intuitive modeling approach for non-expert users in BP Modeling.

However, to integrate the IoT and robotic concepts into the meta-model, we consider respectively, the ISO/IEC 20924: 2021 (ISO, 2021) and the ISO/ DIS 8373:2021 (ISO, 2021) as effective standards. They provide universally accepted notations with detailed descriptions of IoT and robot concepts. In this setting, using these standards allows the avoiding of conflicts between concepts. Furthermore, the software robot (bot) concepts are selected while relying on the literature (e.g., [26, 27], etc.). In what follows, we detail the key concepts for the proposed meta-model:

- FlowElementsContainer: Is an abstract superclass for BPMN diagrams. It designates the superset of elements that are embedded in the BPMN model.
- SubProcess: Is a process in itself, where it works as part of the overall process. It contains a set of flow elements.
- BaseElement: Is the abstract superclass for most BPMN components.
- FlowElement: Is the abstract superclass for the components that can materialize within a process (e.g., gateway, event, etc.).
- Activity: Depicts a process component. It may be an atomic (task) or complex (sub-process). An activity is executed either automatically using a system or manually by humans.

- IoT device: Presents an entity used to interact with the physical world through sensing, actuating, etc. In IoT technology, the device can be a sensor, an actuator, or an IoT gateway.
- Physical entity: Shows the physical world; its properties are captured through one or more sensors and are actuated via one or more actuators.

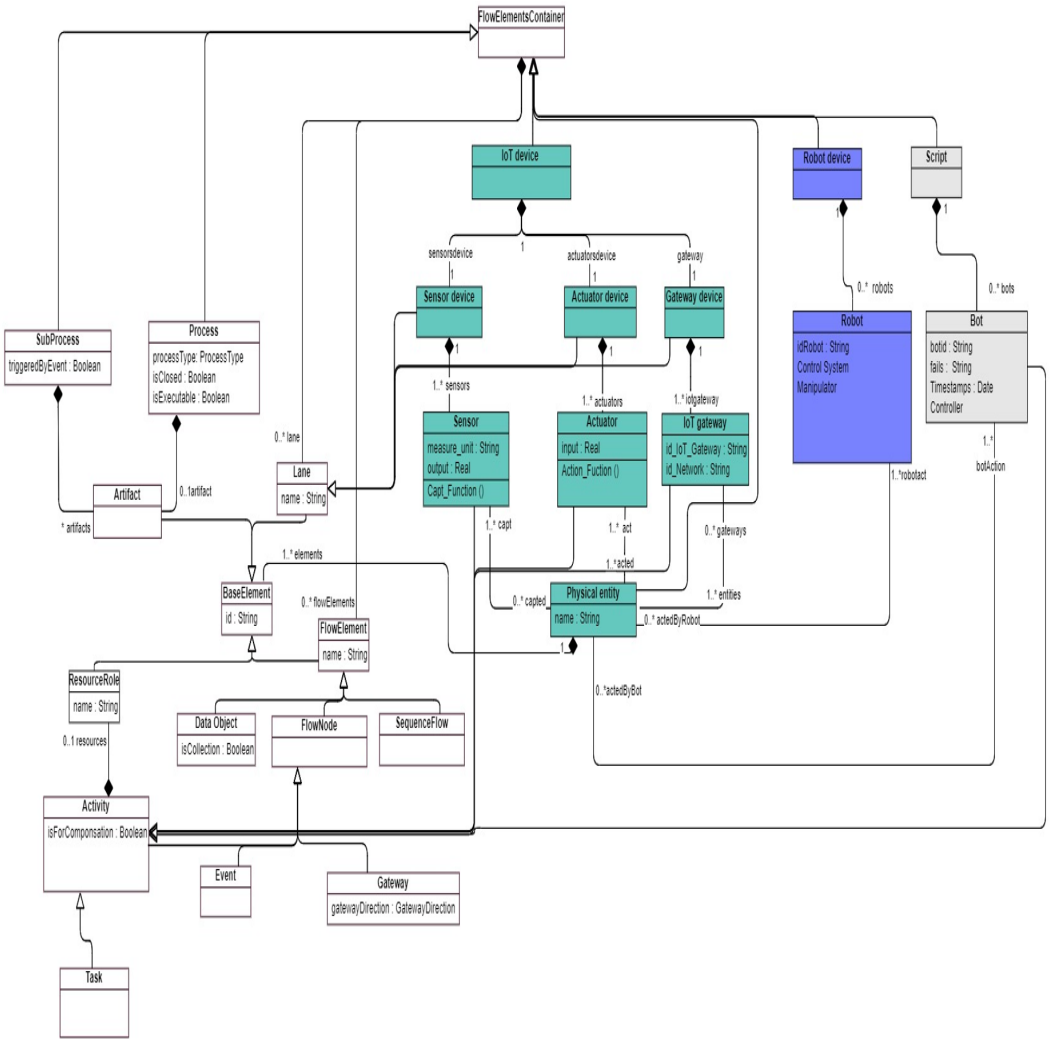

Figure 3. A meta-model for BP automation using IoRT technology.

To guarantee the simplicity and clarity of the proposed meta-model, we propound a set of meta-classes that are described in what follows:

- SensorDevice: Depicts a meta-class that displays the sensors of an IoRT-aware BP model. It is modeled using the lane component.
- ActuatorDevice: Presents a meta-class used to group the actuators considered for the process that integrates the IoRT.
- GatewayDevice: Is a meta-class used to group the IoT gateway devices for an IoRT-aware BP model. It is represented by a lane.
- RobotDevice: Is a concept that presents the process used concepts. It is depicted using a pool.
- Script: Shows a meta-class that displays bots used by the IoRT-aware BP model.

3.1.3 Identify Sub Concepts

The meta-model key concepts that are previously identified should be classified. This involves an in-depth examination of each key concept to pinpoint its sub-concepts, attributes, and functions. This decomposition facilitates the meta-model presentation and enhances its use and extension. In what follows, we give a detailed description of the selected sub-concepts:

- Sensor: Is an IoT device that serves to capture the physical entity's properties. The data captured by the sensor can be transmitted via numerous technologies (e.g., Bluetooth, WiFi, etc.).
- Actuator: Presents a device that acts to change the properties of a physical entity.
- IoT gateway: Is a device to store data coming from sensors. It facilitates device-to-device and device-to-cloud connections.
- Robot: Is a machine that moves within space to accomplish its tasks. It is equipped with sensors. Referring to the ISO/ DIS 8373:2020 standard, a robot has a Control System through which its functions are controlled. Moreover, it has a Manipulator which lays out a mechanism allowing it to accomplish its functions.
- Bot: Is a software program that runs on a physical or virtual machine.
- Event: Gives something that happens within a process. It typically has a cause or an impact on the process flow.
- Gateway: Gives the concept used to present process flows.
- Task: Gives an atomic activity within the process.
- Data Object: Provides information about activities used and provided data.

3.1.4 Define Concepts Relation

The main goal of this step is to facilitate the holistic representation of the meta-model by establishing relations between its fixed concepts and sub-concepts. It determines the way of interaction between the different concepts. These relations take numerous forms, including associations, dependencies, aggregations, hierarchies, etc.

Our proposal includes heterogeneous concepts relation. In what follows, the main considered interactions are presented:

- The IoT device that is modeled using a Pool, is composed of three lanes which are SensorDevice, ActuatorDevice, and GatewayDevice. The SensorDevice contains one or more sensors. ActuatorDevice defines one or more actuators. While GatewayDevice is composed of gateways.
- A sensor interacts with a physical entity, capturing its properties (e.g., temperature, humidity, etc.). The properties of the physical entity are captured using several sensors.
- An actuator can interact with a physical entity, actuating its properties. Moreover, the physical entity can integrate several actuators.
- The Robot device groups a set of robots, while the Script integrates bots.
- The Physical Entity can be actuated by robots and/ or bots.
- A Physical Entity can embed several elements such as Activity or FlowElement. A FlowElement can be a FlowNode, Data Object, or a Sequence Flow
- The ResourceRole, FlowElement, Lane, and Artifact inherit the BaseElement.
- A FlowNode is considered as either an Event or a Gateway.

3.1.5 Check Consistency

During this step, an in-depth check of the fixed meta-model relations, and the concepts attribute is established. The proposed checking aims mainly to identify inconsistencies and conflicts between different meta-model concepts. Checking the latter consistency ensures that each integrated concept aligns seamlessly with others. Back to the literature, this step may involve some experts in the domain, stakeholders, etc. For our proposal, we consider an expert to correctly check its identified concepts, attributes, and relations. Using a set of constraints fixed by our expert, we check our meta-model considering the Object Constraint Language (OCL). The OCL is a formal language that aims to describe the expression of Unified Modeling Language (UML). To accomplish our objective, we tackle both Eclipse Modeling Framework (EMF) and OCLinEcore plug-ins. The EMF is a modeling framework that offers a set of heterogeneous modeling tools. However, the OCLinEcore plugin allows the integration of some OCL constraints within an Ecore model (Meta-model) to check its consistency.

3.1.6 Integrate into Environment

This step aims to integrate the meta-model into its appropriate environment or ecosystem. It involves aligning the proposed meta-model with existing technologies. During our work, we look to integrate the proposed IoRT-aware

BP meta-model within the Industry 4.0 field. Indeed, industry 4.0 is defined as the fourth industrial revolution, which marks a transformative era for businesses and industries, characterized by advanced technologies (e.g., IoT, robots, artificial intelligence (AI)), etc.) [28]. Throughout this step, we aim to extend the BPMN 2.0 plug-in as an open-source Eclipse editor. This plug-in allows us to extend the traditional process concepts by adding new concepts, properties, or icons. This extension ensures that the suggested meta-model actively contributes to the modeling of BP which embeds IoRT concepts. In Section 6, further details are provided regarding the integration of the proposed meta-model within the environment.

3.2 DEVS Meta-Modeling

To check the IoRT-aware BP model structure behaviors, we chose DEVS as a formal method. DEVS is characterized by its ability to simulate complex systems. Therefore, it facilitates their modeling, validation, and implementation. DEVS defines two heterogeneous models, the first one is called the atomic model. It sets forward the discrete event system behaviors (see equation 1). This model depicts how the system reacts to a receiving event.

$$DEVSatomic =< X, Y, S, \delta int, \delta ext, \lambda, ta > \tag{1}$$

Where X presents the input ports of the atomic model. Y sets forward the model output ports. S shows a collection of system states. δint displays the system's internal transition function and it allows the system to adjust its state regarding an internal event. δext gives the model's external transition function and it helps the system to change its state regarding an external event. The λ presents the system output function and it facilitates the interaction of the current model with others by sending events through the output port. Finally, the ta gives the time of each system state.

Furthermore, DEVS provides a coupled model (see Equation (2)) which is used to create complex systems.

$$DEVScoupled =< X, Y, D, Md/d\epsilon D, EIC, EOC, IC > \tag{2}$$

where X sets forward the input ports of the coupled model through which the latter receives events from another atomic/coupled model. Y presents the output ports of the model allowing it to send events to other atomic/coupled models. D depicts the component names that are embedded within the coupled model. The $Md/d\epsilon D$ presents the numerous components integrated within the model. The External Input Coupling (EIC) exhibits a set of model input couplings. They connect the input ports of a coupled model to its internal components. The External Output Coupling (EOC) presents the set of relations that ensure the connection between the model components and its output ports. The Internal Coupling (IC) ensures the interconnection between the coupled model components.

Back to the literature, there is no endorsed meta-model for the DEVS formalism. Therefore, numerous researchers competed to propose a DEVS abstract presentation. However, these propositions fail to encompass concepts for effectively validating DEVS models. To bridge this gap, we extended the DEVS meta-model proposed in [14] (see Figure 4), where we appended some new meta-classes (colored in green).

In what follows, we detailed each DEVS meta-class:

- Transition: Exhibits the change for the atomic model state. DEVS defines two transition types: an *Internal Transition* and an *External Transition*. The *Internal transition* depicts the change of the model state independently of the external event. However, the *External Transition* sets forward the change of the internal model state regarding the receiving of one or more external events.
- State: Depicts the model state at a given time.
- Event: Refers to an action that causes one or more changes in the model state. DEVS defines an (i) Input event received in the model input port and (ii) an Output event provided by the model.
- Output Function: Describes the model outputs. It is generated based on the model's current state and captured events.
- DEVS Coupling Model: Reveals the association defined by the DEVS coupled model. Three types of relation are set forward. The *External Input Coupling*, *Internal Coupling*, and *External Output Coupling*.
- DEVS Port: Outlines a DEVS model port. They allow models to interact with each other. DEVS defines the *DEVS Input Port* and *DEVS Output Port*.
- Structure Control: Denotes an atomic DEVS model used to check the structure for an IoRT-aware BP model based on a set of anti-patterns. It interacts with its corresponding controlled model through the inCS and outCS ports.

- Control Ports: Outlines ports utilized to connect the DEVS model with its Structure Control Model (SCM). These ports allow the exchange of notifications and events relying on the IoRT-aware BP anti-patterns.

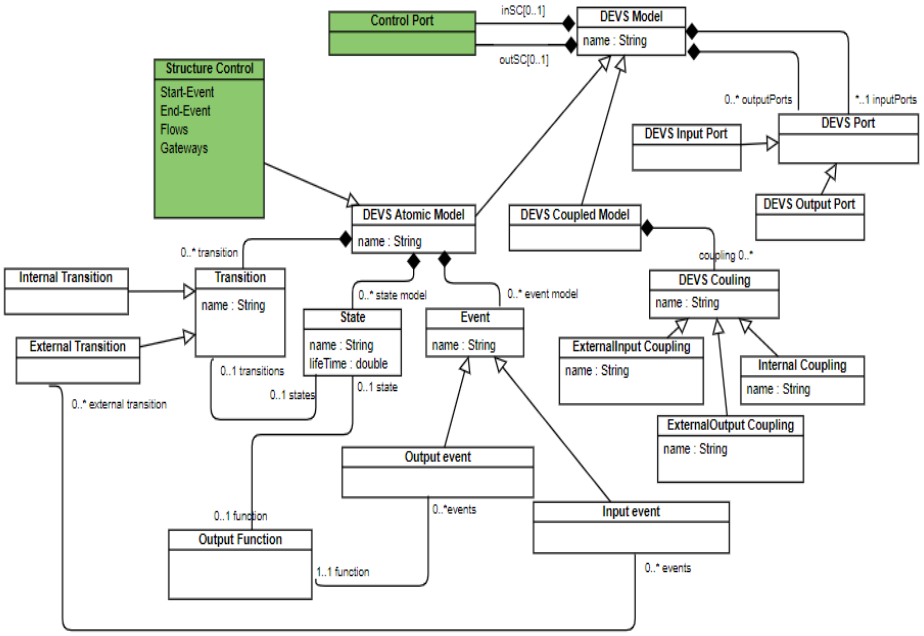

Figure 4. DEVS meta-model.

4. Model-Based Transformation of an IoRT-Aware Business Process into DEVS Formalism

To perform the transformation of the IoRT-aware BP meta-model into DEVS one, we deal with the MDE approach. It is a software development methodology that has proven worthwhile in systems development. It contains the model-driven architecture (MDA) for software development. The MDA corresponds to an approach that separates the functional system specifications from the implementation ones.

During this work, we address the model-to-model transformation, relying on the MDA approach. This transformation ensures the identification of links between meta-model concepts. Considering the IoRT-aware BP and DEVS meta-models, we define a model-to-model transformation from IoRT-aware BP to DEVS. The suggested MDA-based transformation is built upon a set of rules. To support the process structure checking, we have extended our previous work [7]. These rules are the following:

- Rule 1: According to [3, 4, 14], the BP activities are typically converted into atomic DEVS models. With inspiration from previously mentioned works, we convert the sensor into an atomic DEVS model. To ensure that the atomic model interacts with its all predecessors and successors, this model defines input and output ports. Additionally, we suggest adding inCS and outCS ports to the model to guarantee its interaction with the matching Structure Control Model (SCM).
- Rule 2: Back to [3], the BP components displaying straightforward behaviors are converted into a DEVS atomic model. Consequently, we propose to convert an actuator into an atomic DEVS model. To interface with its predecessor, this model defines an input port. Moreover, it delineates inCS and outCS ports to interact with its associated SCM.
- Rule 3: The Data Object in the BP can exhibit simple behavior regarding its several states, including receive, send, store, wait, and so on. We propose transforming the Data Object into an atomic DEVS model to mitigate this complexity. It simplifies the management of Data Object states. These states are modified regarding the received events in the input port. However, to send a notification/ event to other process parts, the DATA Object defines an out port.
- Rule 4: Referring to the literature (e.g., [3, 4]), the DEVS atomic model is used for the modeling of the complex behaviors of such process concepts. Therefore, we propose to transform the bot, robot, IoT gateway, and task into atomic DEVS models that have n input ports and m output ports. The structure of these concepts is checked using an SCM.

- Rule 5: In many previous studies (e.g., [3, 14], etc.), process gateways (e.g., parallel, exclusive, inclusive, etc.) have been transformed into the atomic DEVS model. As a result, we have been inspired by these works to transform the process gateways into an atomic DEVS model. During this transformation, each gateway's (model) defines an input and output ports regarding its corresponding input and output flows. For this model, we propose to add inCS and outCS ports that allow it to interact with its SCM.
- Rule 6: Given that process events (e.g., Message, Timer, Conditional, Signal, Link, Error, Multiple, Escalation, Termination, Cancel, Compensation, etc.) exhibit straightforward behavior, our proposal involves transforming them into logs (events).
- Rule 7: Since a process launches with a start event, we suggest converting it to an atomic model with a single output port. Moreover, the DEVS atomic model defines inCS and outCS ports that ensure its interaction with the SCM model to check whether the IoRT-aware BP model has a start event. However, to deal with the process end event, we propose transforming its end event into a DEVS atomic model, which defines only one input port. The proposed DEVS model has inCS and outCS ports to assure its collaboration with an SCM to check whether the model IoRT-aware BP defines an end event.
- Rule 8: Process, Subprocess, FlowElementsContainer, PhysicalEntity, ActuatorDevice, SensorDevice, GatewayDevice, RobotDevice, IoTDevice, Script are transformed into a coupled model. Since these concepts have a complex internal structure, where each of them can integrate numerous other sub-concepts.

5. IoRT-Aware Business Process Structure Verification

5.1 IoRT-Aware Business Process Structure Behaviors

For an IoRT-aware BP model, it is worthwhile to check, in advance, its structure correctness along with meeting a set of anti-patterns. This determines whether the process presents a convenient structure or not. A process with a correct structure is defined as the model structure consistent with the real world [29]. Consequently, the checking of the process structure model intends to avoid its issues and guarantee its correct execution.

The checking of a process model can be achieved using a set of relevant patterns and anti-patterns [8]. A pattern is defined as a solution for a persistent modeling issue. However, an anti-pattern aims to detect modeling issues [8].

During this research paper, we seek to verify the IoRT-aware BP structure using anti-pattern models. An anti-pattern was introduced by Andrew Koening [30] in 1995. It helps to detect and prevent the undesirable constructs within the model [8]. Moreover, it acts as a warning sign, showing the weaknesses and issues that slow the process.

To check an IoRT-aware BP model structure, we rely on two anti-pattern categories: syntax and control-flow errors. Each category includes a set of anti-pattern models.

The syntax error category describes the process model syntax, where it has been used wrongly. Referring to the literature, we distinguish numerous anti-pattern models that address the process syntax. In what follows, we detail anti-patterns considered during our proposal:

- An IoRT-aware BP without a start event: The start event marks the process's beginning. It initializes the process workflow. Without this component, it is hard to understand when and how the process should begin. Therefore, it is significant to verify whether the process defines a start event.
- An IoRT-aware BP without an end event: The end event designates the end of the process. It lets stakeholders understand whether the process has reached its goals or outcomes. Consequently, each process should define an end event.
- Sequence flow crosses pool boundaries: It designates the interaction between two tasks from different pools. Therefore, interaction between pools should be established using only message flows.
- AND/ OR gateways between two sequences of tasks: This anti-pattern defines a single input and output flow for an AND/ OR gateway. However, within a process, it is impossible to define the AND/ OR gateway with only one input and one output.
- Terminate event for AND/ OR divergence gateways: This anti-pattern serves to define more than one terminate event for the AND/ OR divergence gateways.
- Flows between a Data Object and Events/Gateways: This anti-pattern designates the interaction of the data object via the receiving or sending of data from/to events or gateways. However, the events and gateways can not provide data to be stored within the data object.

The control-flow error anti-patterns category poses issues with the process model's control flow. These problems are considered as soundness property violations [31]. It includes the following anti-pattern models:

- Deadlock: When two tasks wait for each other to complete. As a result, no task can proceed, and the process becomes blocked. For instance, we suggest two tasks *launch irrigation* and *soil moisture monitoring* within an IoRT-aware BP model. The first task depends on the data provided by the second one to determine when and how much water to apply. The second task relies on the first one to adapt the irrigation systems based on the measured moisture levels. Nonetheless, whether there is a malfunction in either the first task or the second one, the deadlock appears. To avoid process blocked situations, the dependencies between tasks must be prevented.
- Dead Activity: Stands for process parts (e.g., task, sub-process, etc.) where the process flow can never reach.
- Infinite loop: Depicts infinite loops within the BP. It means that one or more actions are carried out indefinitely since there is no stopping condition.

5.2 Structure Control Model

To accomplish the checking of the IoRT-aware BP structure, we propose to associate each process component with a Structure Control Model (SCM). This model serves to check the structure of the components based on the set of aforementioned anti-patterns. Moreover, we propose to model the SCM using an atomic DEVS. The model defines input and output ports. The input port allows it to get events from the controlled process component. However, the output port allows the SCM to send the structure-checking results to the corresponding controlled component. In Figure 5, we associate an SCM (colored in cyan) with an IoRT-aware BP component (colored in gray).

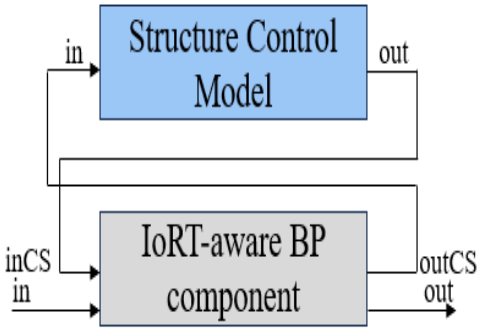

Figure 5. Structure Control Model associated with an IoRT-aware BP component.

In what follows, we give the DEVS formal presentation (see equation 3) of the SCM while checking the IoRT-aware BP component structure.

$$Structure\ Control\ Model =< X, Y, S, \delta int, \delta ext, \lambda, ta > \tag{3}$$

where, X= {in}, where in events= {Check_events: Start, Check_events: End, Check_DataObject, Check_gateways, Check_activities }.
Y= {out}, where out events= {Error_check_event: Start, Error_check_event: End, Well_CheckEvent, Error_check_gateway, Well_CheckGateway, Well_CheckActivity, Error_check_activity, Error_check_DataObject, Well_CheckDataObject}.
S={Waiting,Start_violation,End_violation,Start_End_violation,Boundry_violation,
Boundry_violation,DataObject_Violation,AND_Violation,OR_Violation,
AND_Divergency_Violation,OR_Divergency_Violation,Deadloc_violation,
Loop_violation,Dead_violation,Activity_check, GatewayCheck,Event_check, DataObject_check }.
The external function (δext):
δext(Waiting, in ?Check_events: Start)= Start_violation
δext(Waiting, in ?Check_events: End)= End_violation
δext(Waiting, in ?Check_events: Start)= Event_check
δext(Waiting, in ?Check_events: End)= Event_check
δext(Waiting,in?Check_activities)=A1,A1={Deadloc_violation,Loop_violation,Dead_violation,Boundry_violation
}
δext(Waiting, in ?Check_activities)= Activity_check

δext(Waiting, in ?Check_gateways)=AND_Violation
δext(Waiting, in ?Check_gateways)=OR_Violation
δext(Waiting,in?Check_gateways)=AND_Divergency_Violation
δext(Waiting,in?Check_gateways)=OR_Divergency_Violation
δext(Waiting, in ?Check_gateways)= GatewayCheck
δext(Waiting, in ?Check_DataObject)= DataObject_Violation
δext(Waiting, in ?Check_DataObject)= DataObject_check
The output function (λ):
λ(Event_check) = out! Well_CheckEvent
λ(Start_violation) = out! Error_check_event: Start
λ(End_violation) = out! Error_check_event: End
λ(S1) = out! Error_check_activity, S1 = { Deadloc_violation,Loop_violation, Dead_violation, Boundry_violation }
λ(Activity_check) = out! Well_CheckActivity
λ(GatewayCheck) = out ! Well_CheckGateway
λ(S2) = out! Error_check_gateway, S2 = {AND_Violation,OR_Violation, AND_Divergency_Violation, OR_Divergency_Violation}
λ(Check_DataObject) = out! Well_CheckDataObject
λ(DataObject_Violation) = out! Error_check_DataObject
ta(S3)=∞,S3=Start_violation,End_violation,Start_End_violation,Boundry_violation,DataObject_Violation, AND_Violation,OR_Violation,AND_Divergency_Violation,OR_Divergency_Violation,Deadloc_violation, Loop_violation,Dead_violation,Activity_check, GatewayCheck,Event_check, DataObject_check}
ta(Waiting) = $\tau(Waiting)$

Based on the aforementioned SCM formal description, initially, the model is in the *Waiting* state. Upon receiving the *Check_events: Start* event, it initiates the checking of the process start event. If the SCM encounters a violation for this event, it modifies its state to the *Start_violation*. Additionally, it sends an error event called *Error_check_event: Start* to the corresponding event component, prompting the process to halt its checking action.

When receiving the *Check_events: End* event, the SCM starts checking the process end event. Subsequently, whether the control model captures a violation on the end event, it changes its state to *End_violation* and it transmits an error event named *Error_check_event: End* to the corresponding process end event.

However, the SCM emits a *Well_CheckEvent* to the correspondent components, indicating that there are no violations on the start/ end events, allowing the process to proceed with its checking goals.

As a result, the SCM proceeds with the checking of process gateways. Upon receiving *Check_gateways* event, it initiates the checking action. In this setting, where it faces a violation for the convergency *OR* / *AND* gateways, the SCM changes its state to *OR_Violation*/ *AND_Violation*. Nonetheless, where the control model captures a violation on the AND divergency gateway/ OR divergency gateway, it alters its state to *AND_Divergency_Violation*/ *OR_Divergency_Violation*. Regarding caught errors, the SCM transfers an *Error_check_gateway* to the corresponding gateway, signaling it to halt its checking. Otherwise, it changes its state to *GatewayCheck* and it transmits a *Well_CheckGateway* event to the gateway components, enabling the process to continue its checking.

To check the activity structure, the latter forwards an event to its corresponding SCM. Subsequently, the control model initiates the checking step. Upon encountering a deadlock, loop, or dead activities violation, it changes its state, respectively to, *Deadloc_violation*, *Loop_violation*, and *Dead_violation*. However, when the SCM faces a sequence flow between activities from heterogeneous pools, it alters its state to *Boundry_violation*. Regarding the captured states, the SCM transfers an *Error_check_activity* event to the controlled activity. Otherwise, it modifies its state to *Activity_check* and dispatches a *Well_CheckActivity*.

Similarly, the SCM addresses the checking of the process data object. It initiates the checking upon receiving the *Check_DataObject* event. In case of a violation, it modifies its state to *DataObject_Violation* and it conveys an *Error_check_DataObject* event. Otherwise, the *DataObject_check* state appeared and the *Well_CheckDataObject* submitted.

5.3 Process Component Model

To achieve the check of the IoRT-aware BP model structure based on the fixed anti-patters, we tackle the different process components (e.g., event, gateway, task, etc.) as atomic DEVS models (see Figure 5). Each one defines an input port inCS through which it receives notifications from the SCM. Moreover, it has an output port outCS across

which it sends events/ notifications to the corresponding SCM for its structure checking. In what follows, we detail the formal presentation of a process component behaviors using DEVS (see equation 4).

$$Process\ Model =< X, Y, S, \delta int, \delta ext, \lambda, ta > \tag{4}$$

where, X={inCS, in}, where inCS events={Error_check_event: Start, Error_check_event: End, Well_CheckEvent, Error_check_gateway, Well_CheckGateway, Well_CheckActivity, Error_check_activity, Error_check_DataObject, Well_CheckDataObject}.
Y= {outCS, out}, where outCS events= {Check_events: Start, Check_events: End, Check_DataObject, Check_gateways, Check_activities}.
S={Wait_Check, Error,Well_Check}.
The external function (δext):
δext(Wait_Check,inCS?S4)=Error,S4={Error_check_event:Start,
Error_check_event:End,Error_check_gateway,Error_check_activity, Error_check_DataObject}
δext(Wait_Check,inCSS5)=Well_Check,S5={Well_CheckEvent,Well_CheckGateway,Well_CheckActivity,
Well_CheckDataObject}
The output function (λ):
λ(Wait_Check) = outCS! Check_events: Start
λ(Wait_Check) = outCS! Check_events: End
λ(Wait_Check) = outCS! Check_DataObject
λ(Wait_Check) = outCS! Check_gateways
λ(Wait_Check) = outCS! Check_activities
ta(S6) = ∞, where S6= S {Error,Well_Check}
ta(Wait_Check) = $\tau(Wait\_Check)$

During the proposed process checking approach, the controlled component is initially in the *Wait_Check* state. Subsequently, it sends one of the following events to its corresponding SCM:

- *Check_events: Start*: Depicts that the event is sent by a start component.
- *Check_events: End*: Indicates that the event is forwarded by an end component.
- *Check_DataObject*: Identifies that the event is transferred by a data object.
- *Check_gateways*: Presents that the event is transmitted by a gateway.
- *Check_activities*: Sets forward that the event is sent by an activity to launch the checking action.

Regarding the component output, the process receives one event and changes its state either to *Error* or *Well_Check*. Indeed, *Error* state captures with the receiving of *Error_check_event:Start*, *Error_check_event:End*, *Error_check_gateway*, *Error_check_activity*, or *Error_check_DataObject* events. However, the *Well_Check* state is used when the components get a *Well_CheckEvent*, *Well_CheckGateway*, *Well_CheckActivity*, or *Well_CheckDataObject* events.

6. Experimentation

As a proof of concept for the proposed meta-model, we generate an IoRT-aware BP model in the agriculture field. To accomplish this goal, we extend the BPMN Modeler 2.0 plug-in. This plug-in gives an open-source Eclipse editor, which allows the extension of the traditional process by adding concepts, properties, and icons. During the proposed extension, we add a new concept category called *IoRT components* which contains the IoT, bot, and robot concepts that are identified in the IoRT-aware BP meta-model (see Figure 6).

Using the extended BPMN 2.0 plug-in, we generate an IoRT-aware BP in the smart irrigation management system, which aims to improve water use efficiency. The process starts with the capturing of water level and temperature degree using two sensors which are respectively *Capture water* and *Capture temperature*. The captured data are stored using the *Storage water data* and *Storage temperature data* tasks. Considering these data, the *Request irrigation decision* task, sends an event to launch *Ckeck irrigation model* in order to make the irrigation decision. According to this decision, the process will finish where no irrigation is needed. Otherwise, two actions are launched: *Request start irrigation* and *Request picking weeds*. An IoT actuator *Launch irrigation* performs the first task while a robot device accomplishes the second task.

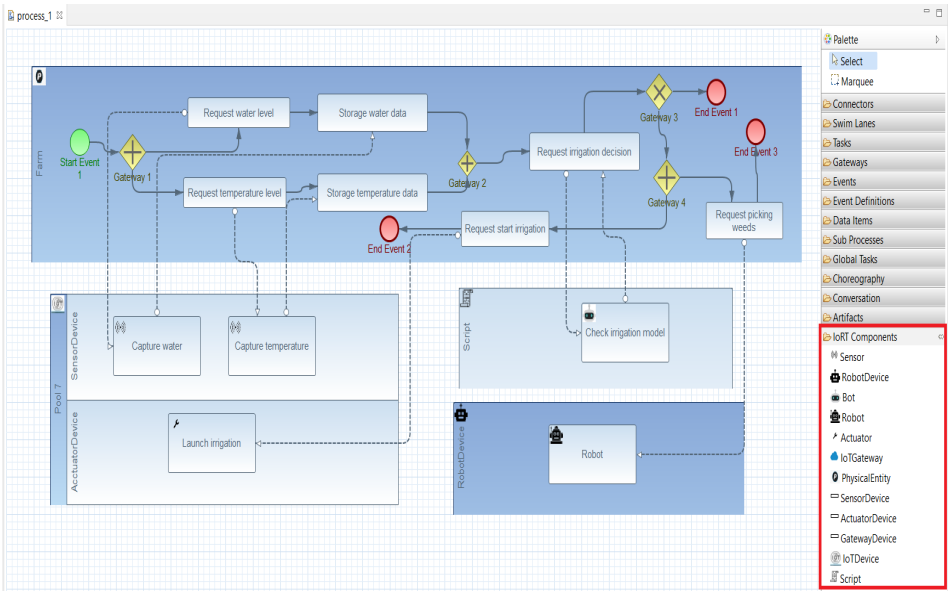

Figure 6. Extension of BPMN 2.0 modeler plug-in to support IoRT concepts.

To accomplish our proposed model-to-model transformation based on the MDE approach, we address the Atlas Transformation Language (ATL). It serves as a language that is used to transform a source meta-model to a target one. During the model-to-model transformation, we tackle the Eclipse Modeling Framework (EMF), which offers a range of standard development tools. Initially, the meta-model source and target are defined in ecore files. Using these files, our transformation rules, are crafted with ATL. Figure 7 shows an overview of the proposed ATL transformation rules.

IoRTP2DEVS.atl

```
1  module IoRTP2DEVS;
2  -- @path BPV2 = /TransformationBP2DEVS/BPV2.ecore
3  -- @path DEVS2 = /TransformationBP2DEVS/DEVS2.ecore
4  create OUT: DEVS2 from IN: BPV2;
5  rule Sensor2DEVSAtomic {
6    from
7      Sensor : BPV2!Sensor
8    to
9      At : DEVS2! DEVS_Atomic_Model (Name <- Sensor.Name,
10         transitions <- Sensor.OutCS,At <- At.inCS
11         At <- At.outControl, inputPort <- Sensor.capted,
12         outputPorts <- Sensor.capt, At <- At.states, At <- At.events )
13 }
14 rule Actuator2DEVSAtomic {
15   from
16     Actuator : BPV2!Actuator
17   to
18   Atomic : DEVS2! DEVS_Atomic_Model ( Name <- Actuator.Name,
19    inputPort <- Actuator.Input,OutCS <- Actuator.act,inCS <- Actuator.acted)
20 }
```

Figure 7. Example of transformation rules implemented using ATL.

As proof of the proposed process structure verification approach, we deal with two scenarios of the IoRT-aware BP in the agriculture field. These scenarios rely on the IoRT-aware BP model in Figure 6.

Toward the process checking goal, we use the DEVS-Suite tool to simulate the proposed IoRT-aware BP scenarios. Leveraging this tool, we associate each process component (e.g., event, task, gateway, etc) to an SCM. This control model checks the process components structure where it captures the violated errors.

This simulation allows us to observe the process components and the SCM behaviors.

The first scenario (see Figure 8) presents a successful execution of the IoRT-aware BP, where all the process components are verified. As shown in Figure 8, all the process components attain the *well_check* state which designates that they are successfully checked. Additionally, the proposed simulation reveals that the Structure Control Models (e.g., SCM1, SCM14, etc.) used to check the process events; are achieved in the *Event_check* state. Moreover, Figure 8 exhibits that the Control Models (e.g., SCM2, SCM7, SCM9, etc.) reached the *GatewayCheck*

during the checking of the process gateways. Similarly, this Figure highlights that the activities control models (e.g., SCM3, SCM4, SCM5, etc.) have effectively attained the Activity_check state. Table 2 gives more details about the first scenario checked process.

The second scenario (see Figure 9) executes the IoRT-aware BP depicted in Figure 6, where numerous violations are captured. Indeed, during this scenario, we do not define an end event for the process. Moreover, concerning Gateways 1 and 3, we define an error in their outputs, where we fix one output for each gateway. Additionally, we outline some dead activities within the process, for instance *Request water level* and *Launch irrigation* (see Figure 9). During the checking of this process instance, the *SCM1* associated with the start event launches to check whether the process defines a start event or not. Therefore, it correctly checks the start event component. Then the *SCM2* associated with gateway 1 initiates its checking and a violation of gateway outputs is detected. Subsequently, it transmits an error event *AND_Divergency _Violation* to the gateway, which reinforces it to stop its execution and to not launch its successor tasks.

As demonstrated in Table 2, the verification process applied to the second scenario checks the process start event while detecting a violation in gateway 1. Consequently, the model halts, thus avoiding the checking of other process components.

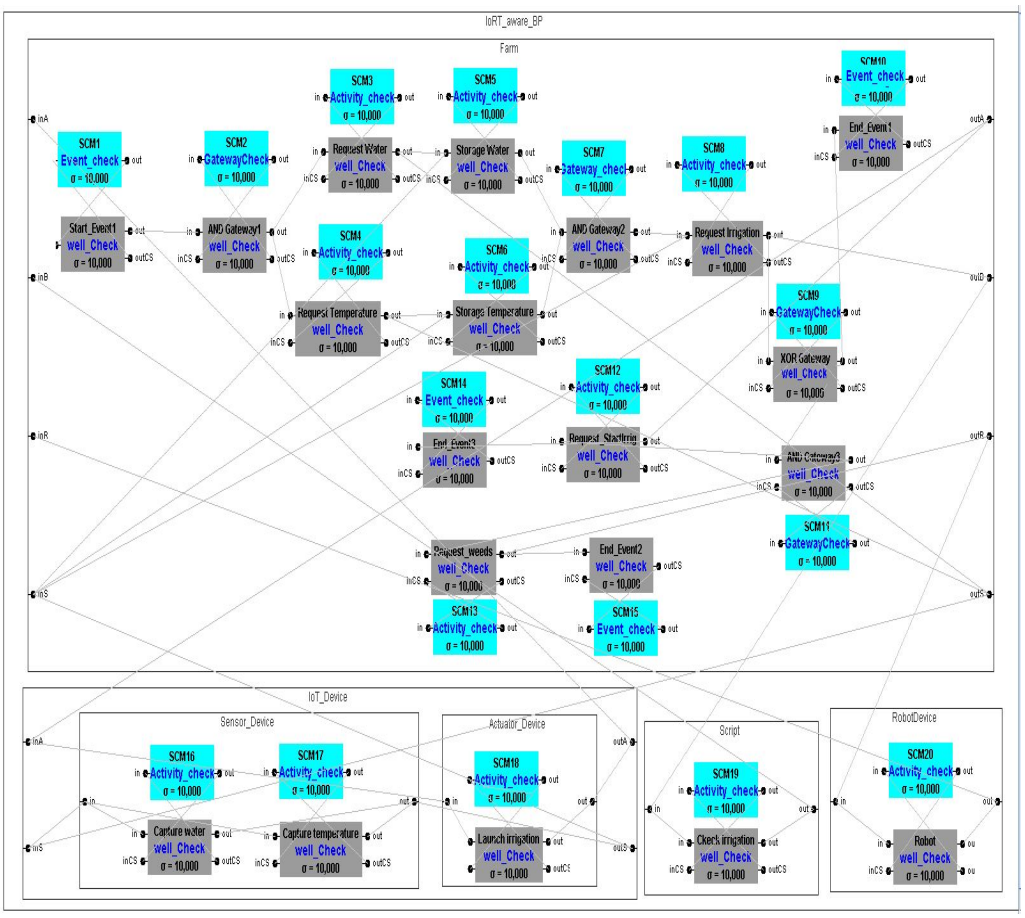

Figure 8. First scenario with successful structure verification.

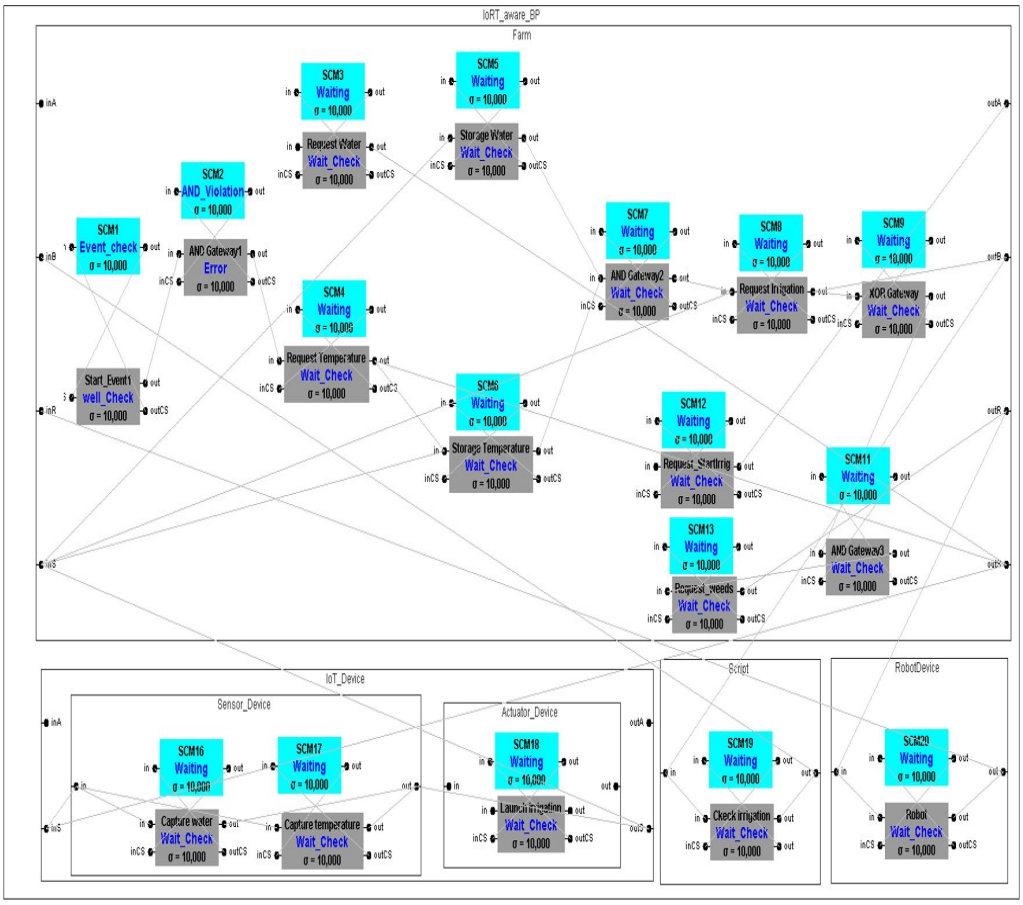

Figure 9. Second scenario with error structure verification.

Table 2. Results for the first and second scenarios checking using the DEVS-suite simulator.

Process Components	**Fist Scenario**	**Second Scenario**
Start Event 1	✓	✓
Gateway 1	✓	violate
Request Water level	✓	-
Request temperature level	✓	-
Storage water data	✓	-
Storage temperature level	✓	-
Gateway 2	✓	-
Request irrigation decision	✓	-
Gateway 3	✓	-
Gateway 4	✓	-
End Event 1	✓	-
Request start irrigation	✓	-
End event 2	✓	-
Request picking weeds	✓	-
Capture water	✓	-
Capture temperature	✓	-
Launch irrigation	✓	-
Check irrigation model	✓	-
Robot	✓	-

7. Conclusions

The current research work focuses on verifying IoRT-aware BP structure demeanors based on a set of fixed anti-patterns and using the DEVS formalism. This verification aims to avoid incorrect executions resulting from incoherent process structure. To accomplish this objective, we propose a meta-model of an IoRT-aware BP, where we adhere to the top-down approach. Moreover, we extend a DEVS meta-model published in the literature. Using these meta-models, we propose an approach based on MDE for transforming the IoRT-aware BP into DEVS formalism. The proposed transformation relies on a set of rules. Referring to the transformed model, we verify an IoRT-aware BP model based on a set of anti-patterns within the process model. Consequently, each process component is associated with a control model called the Structure Control Model (SCM). It is used to check its structure regarding a set of fixed process anti-patterns. To assess our proposal, we consider a case study of IoRT-aware BP in agriculture.

At this analysis stage, we emphasize that our research represents a step that can be further advanced, expanded, and developed. Indeed, it can be enhanced by examining spatial constraints, particularly given that robot devices are integrated within the IoRT-aware BP framework. Also, checking the data correctness within the IoRT-aware BP constitutes one of our future directions, given that IoT and robot devices handle a massive amount of data.

Author Contributions

Conceptualization, N.F., I.B.L. and K.B.; methodology, N.F., I.B.L. and K.B.; software, N.F., I.B.L. and K.B.; validation, N.F., I.B.L. and K.B.; formal analysis,N.F., I.B.L. and K.B.; investigation, N.F., I.B.L. and K.B.; resources, N.F., I.B.L. andK.B.; data curation, N.F., I.B.L. and K.B.; writing—original draft preparation, N.F., I.B.L. and K.B.; writing—review and editing, N.F., I.B.L. and K.B.; visualization, N.F., I.B.L. and K.B.; supervision, K.B. Project administration, K.B. All authors have read and agreed to the published version of the manuscript.

Funding

This research has no external funding

Conflict of Interest Statement

The authors declare no conflict of interest.

Data Availability Statement

Not applicable.

References

[1] Fattouch, N., Lahmar, I.B., Boukadi, K.: A comprehensive architecture for an iort-aware business process outsourcing into fog and cloud computing. *The Tunisian-Algerian Joint Conference on Applied Computing* 3067, 164–172 (2021).

[2] Kheldoun, A., Barkaoui, K., Ioualalen, M.: Formal verification of complex business processes based on high-level petri nets. *Inf. Sci.* 385, 39–54 (2017).

[3] Boukelkoul, S., Maamri, R.: Optimal model transformation of BPMN to DEVS. The International Conference of Computer Systems and Applications, 1–8 (2015).

[4] Bazoun, H., Bouanan, Y., Zacharewicz, G., Ducq, Y., Boyé, H.: Business process simulation: transformation of BPMN 2.0 to DEVS models (WIP). The Symposium on Theory of Modeling and Simulation—DEVS Integrative M&S Symposium, 20 (2014).

[5] Fattouch, N., Lahmar, I.B., Boukadi, K.: Towards A meta-modeling approach for an iort-aware business process. The International Conference on Modelling and Simulation, 29–35 (2022).

[6] Pan, Z., Wang, X., Teng, R., Cao, X.: Computer-aided design-while-engineering technology in top-down modeling of mechanical product. *Comput. Ind.* 75, 151–161 (2016).

[7] Fattouch, N., Ben Lahmar, I., Boukadi, K.: A model-based approach for the transformation and verification of an iort-aware business process. The International Conference on Intelligent Systems Design and Applications (2023).

[8] Lehmann, B., Alexander, P., Lichter, H., Hacks, S.: Towards the identification of process anti-patterns in enterprise architecture models. *International Workshop on Quantitative Approaches to Software Quality co-located with 27th Asia-Pacific Software Engineering Conference* 2767, 47–54 (2020).

[9] Kang, G., Yang, L., Zhang, L.: Verification of behavioral soundness for artifact-centric business process model with synchronizations. *Future Gener. Comput. Syst.* 98, 503–511 (2019).

[10] Medina-Garcia, S., Medina-Marin, J., Montaño-Arango, O., Gonzalez-Hernandez, M., Hernandez-Gress, E.S.: A petri net approach for business process modeling and simulation. *Applied Sciences* 13(20), 11192 (2023).

[11] Zhang, M., Feng, F., Zhang, Z., Wen, J.: A new business process verification approach for e-commerce using petri nets. *Int. J. Enterp. Inf. Syst.* 16(1), 92–107 (2020).

[12] Nosova, S., Norkina, A., Makar, S., Gerasimenko, T., Medvedeva, O.: Artificial intelligence as a driver of business process transformation. *The International Conference on Brain-Inspired Cognitive Architectures for Artificial Intelligence* 213, 276–284 (2022).

[13] Saddem-Yagoubi, R., Poizat, P., Houhou, S.: Business processes meet spatial concerns: The sbpmn verification framework. *Formal Methods - International Symposium* 13047, 218–234 (2021).

[14] Wu, X., Yan, X., Li, X., Wa, Y.: Simulating hybrid sysml models: a model transformation approach under the DEVS framework. *J. Supercomput.* 79(2), 2010–2030 (2023).

[15] Roa, J., Chiotti, O., Villarreal, P.: Specification of behavioral anti-patterns for the verification of block-structured collaborative business processes. *Information and Software Technology* 75, 148–170 (2016).

[16] Meyer, T.: A symmetric petri net model of generic publish-subscribe systems for verification and business process conformance checking. *The International Workshop on Petri Nets and Software Engineering co-located with the International Conference on Application and Theory of Petri Nets and Concurrency* 3430, 88–109 (2023).

[17] Beerepoot, I., Ciccio, C.D., Reijers, H.A., Rinderle-Ma, S., Bandara, W., Burattin, A., Calvanese, D., Chen, T., Cohen, I., Depaire, B., Federico, G.D., Dumas, M., Dun, C.G.J., Fehrer, T., Fischer, D.A., Gal, A., Indulska, M., Isahagian, V., Klinkmüller, C., Kratsch, W., Leopold, H., Looy, A.V., López, H.A., Lukumbuzya, S., Mendling, J., Meyers, L., Moder, L., Montali, M., Muthusamy, V., Reichert, M., Rizk, Y., Rosemann, M., Röglinger, M., Sadiq, S., Seiger, R., Slaats, T., Simkus, M., Someh, I.A., Weber, B., Weber, I., Weske, M., Zerbato, F.: The biggest business process management problems to solve before we die. *Comput. Ind.* 146, 103837 (2023).

[18] Bédard, A., Hallé, S.: Formal verification for event stream processing: Model checking of beepbeep stream processing pipelines. *Inf. Comput.* 293, 105058 (2023).

[19] Ma, Q., Kaczmarek-Heß, M., Kinderen, S.: Validation and verification in domain-specific modeling method engineering: an integrated life-cycle view. *Softw. Syst. Model.* 22(2), 647–666 (2023).

[20] Szelagowski, M., Biernacki, P., Berniak-Wozny, J., Lipinski, C.R.: Proposal of BPMN extension with a view to effective modeling of clinical pathways. *Bus. Process. Manag. J.* 28(5/6), 1364–1390 (2022).

[21] Beest, N., Groefsema, H., Cryer, A., Governatori, G., Tosatto, S.C., Burke, H.M.S.: Cross-instance regulatory compliance checking of business process event logs. *IEEE Trans. Software Eng.* 49(11), 4917–4931 (2023).

[22] Franceschetti, M., Seiger, R., Weber, B.: An event-centric metamodel for iot-driven process monitoring and conformance checking. *Business Process Management Workshops - International Workshops, Utrecht* 492, 131–143 (2023).

[23] Fattouch, N., Rekik, M., Wakrime, A.A., Boukadi, K.: A model-driven engineering approach for business process based saas services composition. The International Conference on Computer Systems and Applications, 1–8 (2019).

[24] Sperner, K., Meyer, S., Magerkurth, C.: Introducing entity-based concepts to business process modeling. Business Process Model and Notation: Third International Workshop, 166–171 (2011).

[25] Gore, R., Diallo, S., Lynch, C., Padilla, J.: Augmenting bottom-up metamodels with predicates. Journal of Artificial Societies and Social Simulation **20**(1) (2017).

[26] Egger, A., Hofstede, A.H., Kratsch, W., Leemans, S.J., Röglinger, M., Wynn, M.T.: Bot log mining: Using logs from robotic process automation for process mining. The International Conference on Conceptual Modeling, 51–61 (2020).

[27] Romao, M., Costa, J., Costa, C.J.: Robotic process automation: A case study in the banking industry. The Iberian Conference on information systems and technologies (CISTI), 1–6 (2019).

[28] Abulibdeh, A., Zaidan, E., Abulibdeh, R.: Navigating the confluence of artificial intelligence and education for sustainable development in the era of industry 4.0: Challenges, opportunities, and ethical dimensions. Journal of Cleaner Production, 140527 (2024).

[29] Becker, J., Rosemann, M., Uthmann, C.: Guidelines of business process modeling. *Business Process Management, Models, Techniques, and Empirical Studies* 1806, 30–49 (2000).

[30] Koenig, A.: Patterns and antipatterns. *J. Object Oriented Program.* 8(1), 46–48 (1995).

[31] Koschmider, A., Laue, R., Fellmann, M.: Business process model anti-patterns: a bibliography and taxonomy of published work (2019).