Supporting Technology Independent Interoperability between Business Processes and IoT Devices

Pedro Valderas

PROS Research Centre, Universitat Politècnica de València, Spain

pvalderas@pros.upv.es

Abstract. *The Internet of Things allows digital business processes to use physical devices to automate and improve the execution of some of their tasks. A business process is executed by a process engine, which usually provides built-in mechanisms to interact with IoT devices through REST APIs. However, IoT devices are technologically heterogeneous by nature. If they are not based on REST, the native support provided by process engines is not enough to interact with them. In addition, in case IoT devices have REST APIs, built-in mechanisms of process engines provide technologically coupled solutions since the engine must be configured with the connection data of each specific device. Thus, time-consuming adaptation tasks are required to update the process engine if IoT devices need to be replaced due to changes in business requirements. In this work, we provide a solution to improve these problems based on ontologies, BPMN, and microservices. First, IoT devices are abstractly described by means of the notions proposed by the SAREF ontology. Then, executable BPMN models that use the ontological descriptions of IoT devices are defined to implement a business process. Finally, microservices are used as instantiations of the IoT devices defined with the SAREF ontology, playing the role of gateways between the process engine and the real IoT devices, providing a high level of technological independence between both.*

1. Introduction

The Internet of Things (IoT) enables the connection of the physical world to digital business processes (BP). BPs use IoT devices as digitalised physical resources that participate as actors to automate and improve the execution of some of their tasks (Beverungen et al., 2020). For instance, during the execution of a BP, robotic bridges can be automatically opened upon the arrival of a ship, or a CO2 sensor can be accessed to know the CO2 level of an environment at a specific time. The BPs that make use of IoT devices are known as IoT-enhanced BPs (Torres et al., 2020).

IoT devices are heterogeneous by nature. They differ regarding communication protocols, interaction paradigms, data formats, and computing and storage power. For instance, there are commercial CO2 sensors that save the measures into their cloud server and provide an API REST to access them, while others allow a direct wireless connection through the MQTT protocol[1]. BPs are executed by a process engine, which is responsible

[1] See as representative examples these two commercial CO2 sensors that provide a cloud-based REST API and wireless MQTT-based communication, respectively: (1) https://www.disruptive-technologies.com/products/wireless-co2-sensor, (2) https://envira.global/nanoenvi-iaq/

for instantiating and controlling their execution (Weske, 2012). Currently, the most used BP engines are those based on the Business Process Model and Notation standard (BPMN, 2010), and there is a myriad of different commercial options (e.g., Camunda[2], Bonita[3], Bizagi[4], Siganvio[5], etc.).

Most of the existing BPMN engines provide built-in mechanisms to access external resources through REST APIs, which can be used to interact with IoT devices that support this communication model. However, these solutions require configuring the BPMN engine with the connection data of the IoT devices (protocol (HTTP or HTTPS), IP, port, path, HTTP method, input data, output data, etc.), which creates coupled implementations of IoT-enhanced BPs. If IoT devices need to be replaced due to changes in business requirements after an IoT-enhanced BP is deployed, time-consuming adaptation tasks are required to update the BPMN engine accordingly. On the other hand, if the BPMN engine needs to interact with IoT devices supported by other technologies, such as MQTT-based communication, complex bridges need to be implemented and integrated into the engine, which, in addition, also creates coupled implementations.

Thus, the integration of BPMN engines with IoT devices in order to implement IoT-enhanced BPs is not an easy task since different technological restrictions must be considered depending on the devices' underlying technology. This is still more complex if we need an IoT-enhanced BP implementation decoupled from the IoT technology in such a way that changes in the IoT devices do not require the adaptation of the BPMN engine after an IoT-enhanced BP has already been deployed.

We think this problem can be improved with the use of microservices, ontologies, event-based communication, and the proper development tools. In our previous work (Valderas et al., 2022), we proposed an architecture based on microservices to support the development of IoT-enhanced BPs. In (Valderas et al., 2023), we demonstrate how this architecture can support interdisciplinary development teams. However, this solution introduces technological dependencies among the architectural elements, which produce interoperability problems when some of them need to be replaced to support new technological requirements. In (Valderas & Torres, 2023), we improved this problem by introducing a bottom-up (i.e., from IoT devices to the BPMN engine) event-based communication that allows IoT devices to inject data into the BPMN engine in a decoupled way. In this work, we complement all the previous work by supporting a decoupled top-down (i.e., from the BPMN engine to the IoT devices) ontology-based communication in such a way a BPMN engine can ask an IoT device for the execution of a task without requiring knowing the underlying technology of the device.

The rest of the paper is organised as follows: Section 2 introduces a motivation example in order to understand the problem faced in this work properly. Section 3 presents an approach based on ontologies, BPMN and microservices to develop IoT-Enhanced BPs, and Section 4 introduces an architecture to support their deployment and execution. Section 5 analyses the related work, and conclusions and further work are commented on in Section 6.

[2] https://camunda.com/
[3] https://www.bonitasoft.com/
[4] https://www.bizagi.com/
[5] https://www.signavio.com/

2. Motivation Example

In this section, we present a motivation example to understand better the challenge we face in this work. Let's consider the IoT-enhanced BP presented in Figure 1, which describes the process of managing the CO2 level in a smart library. It is defined in BPMN by applying the modelling guidelines presented in (Valderas et al., 2022). In a nutshell: (1) A pool is used to represent the whole IoT-enhance BP within an organisation; (2) each IoT device that participates in the process is represented by a lane within this pool; (3) each IoT device's action required by the BP is defined as a Service Task in the corresponding lane; and (4) data autonomously injected by the IoT devices into the process is represented by means of message flows drawn between a pool that represents the physical world and the pool that represents the IoT-enhance BP. These message flows are labelled with the IoT device that injects the data.

According to the process shown in Figure 1, we have a CO2 sensor informing the BP about the CO2 level periodically (let's suppose every ten minutes). When the CO2 level is lower or equal to 2000 ppm, the BP does nothing. However, if the CO2 level is greater than this value, an emergency protocol is executed. First, an emergency air renewal system is started, and the notification of CO2 levels is stopped in the sensor. After waiting for five minutes, the BP directly requests the current level of CO2 to the CO2 Sensor. If it is lower or equal to 1500 ppm, the emergency air renewal system is stopped, and the sensor notifications are configured every ten minutes again. Otherwise, the BP asks the CO2 Sensor to notify the CO2 level every minute, an alarm is activated to warn people who are inside the library that they must leave, and access to the library is forbidden. Once the library is empty, the alarm is stopped. Regarding the CO2 level, the BP waits until it is lower or equal to 1000 ppm. At this moment, the emergency air renewal system is then stopped, CO2 notifications are configured every 10 minutes, and access is allowed again.

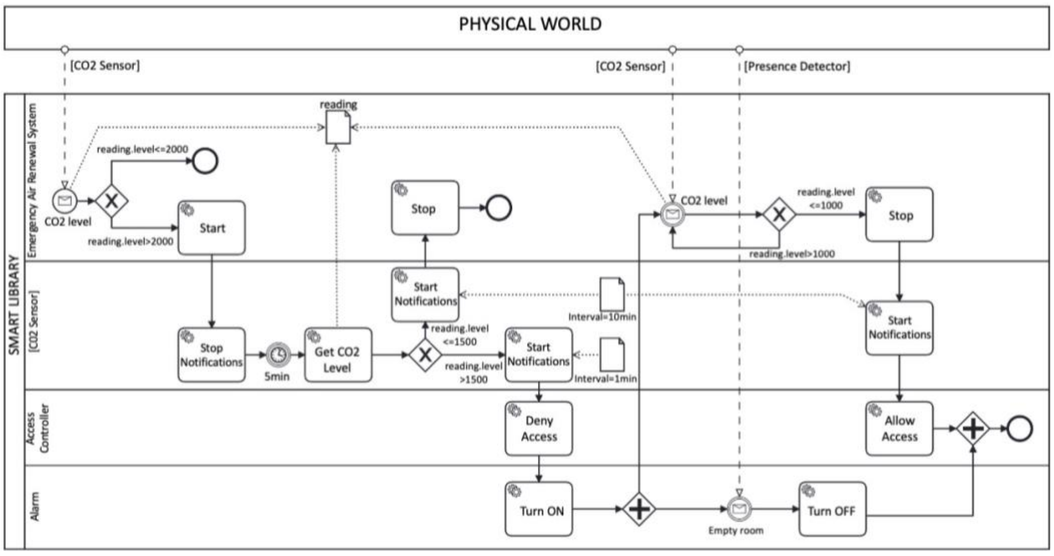
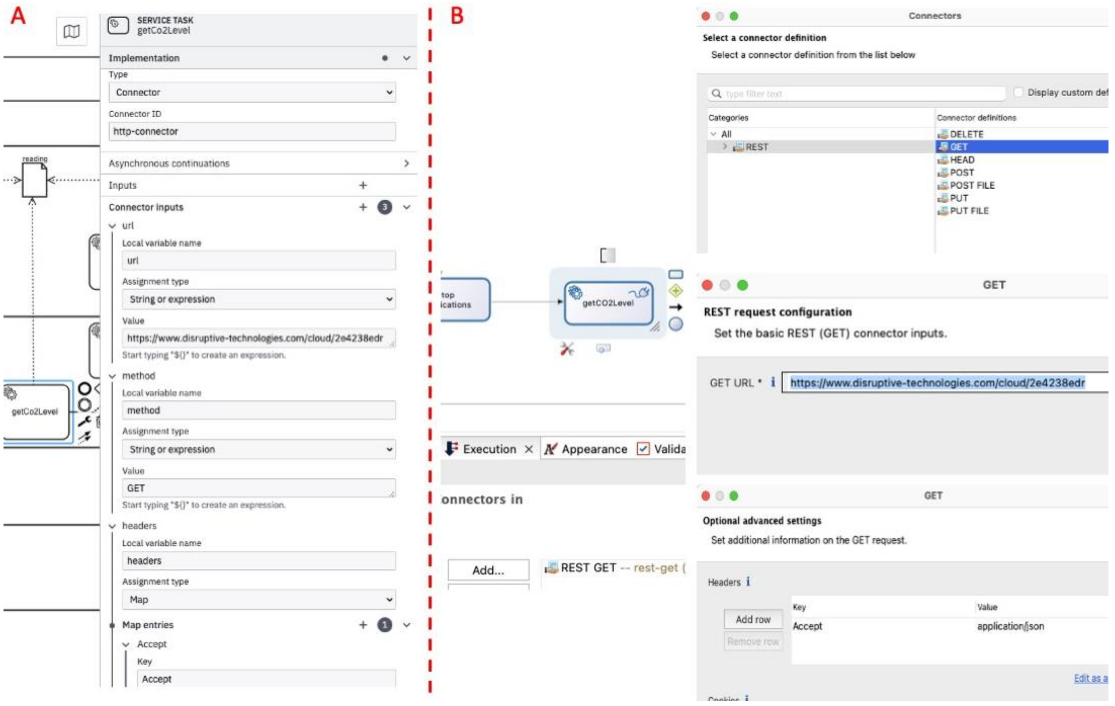


Figure 1. An IoT-enhanced BP for controlling CO2 level

How IoT devices autonomously inject data such as the CO2 level or the detection of presence into the process (i.e., message flows of the above-introduced BPMN model) was already faced in (Valderas & Torres, 2023). Thus, this challenge is out of the scope of this paper. In the current work, we face how a BP engine that executes the BPMN model in Figure 1 can interact, for instance, with the CO2 sensor in order to explicitly ask for the CO2 level (Service Task `Get CO2 Level` defined in the lane `CO2 Sensor`) or with the access controller to deny the access of people (Service Task `Deny Access` defined in the lane `Access Controller`) in a decoupled and independent way from the underlying technology of these IoT devices.

In order to see how current commercial BPMN engines support the interaction with external resources, let's analyse two of the most used open-source platforms: Camunda and Bonita. Regarding **REST APIs**, Camunda provides different solutions to do so, including the delegation to Java code, scripts, or connectors (Deehan et al., 2022). All of these solutions need either to hard-code the connection data of IoT devices in a specific piece of code or use the modeller interface to configure it. For simplicity purposes, Figure 2A only shows how a connector can be configured through the modeller interface in order to interact with a CO2 sensor that publishes a REST API. In the same way, Bonita allows connecting to a REST API either through a custom connector[6] implemented in Java, Groovy or Kotlin, which requires to hard-code the connection data, or using the REST built-in connector and configuring it through a wizard (as shown in Figure 2B). Regarding **other technologies**, such as the MQTT protocol, Camunda provides two solutions: either to implement a Java module that manages the interaction with the message broker or to use an external library that must be integrated into the commercial tool (Zambroski & Pohl, 2016). In Bonita, a custom connector should be implemented by using the above-commented programming languages.



SERVICE TASK
getCo2Level

Implementation

Type: Connector

Connector ID: http-connector

Asynchronous continuations

Inputs

Connector inputs 3

url
Local variable name: url
Assignment type: String or expression
Value: https://www.disruptive-technologies.com/cloud/2e4238edr
Start typing "${}" to create an expression.

method
Local variable name: method
Assignment type: String or expression
Value: GET
Start typing "${}" to create an expression.

headers
Local variable name: headers
Assignment type: Map
Map entries 1
Accept
Key: Accept

REST GET -- rest-get (

Select a connector definition
Select a connector definition from the list below

Categories	Connector definitions
All	DELETE
REST	GET
	HEAD
	POST
	POST FILE
	PUT
	PUT FILE

GET

REST request configuration
Set the basic REST (GET) connector inputs.

GET URL * https://www.disruptive-technologies.com/cloud/2e4238edr

GET

Optional advanced settings
Set additional information on the GET request.

Headers

Key	Value
Accept	application/json

Figure 2. Connection of Camunda (A) and Bonita (B) with a REST API

Camunda and Bonita are only two representative examples of commercial BPMN engines. However, the rest of the engines that we can currently find in the market provide

[6] https://documentation.bonitasoft.com/bonita/latest/process/connector-archetype-tutorial

analogous solutions. Note that these solutions make the execution of the BPMN model that defines an IoT-Enhanced BP totally dependent on the technology that supports the IoT devices. Thus, changes in these devices require complex adaptations in the BPMN engine. For instance, consider that we initially configured the IoT-enhanced BP in Figure 1 to use a CO2 sensor that provides a REST API. We needed to create a configuration like the one shown in Figure 2 for each Service Task defined in the `CO2 Sensor` lane. Consider, also, that time after the deployment of the process, the CO2 Sensor needs to be replaced by another model that supports a communication based on MQTT. In this situation, time-consuming tasks of programming and re-configuration in the BPMN engine are needed in order to work with the new underlying technology. In addition, note that the CO2 sensor used in the IoT-enhanced BP in Figure 1 provides three operations through its REST API: `Get CO2`, `Start Notifications` and `Stop Notifications`. The newly deployed CO2 sensor should also provide these operations, or the logic of the process should be redefined.

To conclude this section, we can see that changing an IoT device when an IoT-enhanced BP is already deployed can be a complicated task if the BPMN engine that executes it directly interacts with the participant devices. In this work, we propose a solution based on the integration of BPMN and ontological descriptions of IoT devices in order to improve this problem.

3. Developing ontology-based IoT-Enhanced BPs

The solution we present in the work is a development process for IoT-Enhanced BPs based on the SAREF ontology, BPMN, and microservices. This process includes the following three stages:

1. IoT Device Ontological Description: In this stage, we propose to use the SAREF (Smart Appliances REFerence) ontology (Daniele et al., 2017) in order to describe IoT devices in an abstract way, totally independent from any manufacturer or technology. There are other ontologies that represent IoT concepts, such as SOSA[7] SensorML[8] or SAO[9]. We used SAREF because its concepts facilitate the description of the different functional capabilities that are provided by IoT Devices (actuating, meter, and sensing), which helps us to integrate them into a business process properly.
2. IoT-Enhanced BP Design: This stage consists of creating a BPMN model that describes the logic of the process that must be implemented. This BPMN model includes references to the abstract IoT devices defined in the previous stage.
3. IoT Device Microservice Implementation: As we explain in Section 4, the BPMN model defined in the previous stage is deployed in an architecture in which microservices play the role of gateway between the BPMN engine and the physical IoT devices. In this stage of the process, these microservices must be implemented as instances of the abstract descriptions of the IoT devices defined in the first stage. We use microservices because this architectural solution was proposed to create systems as a composition of highly decoupled independent business components and the main goal of this work is providing a high level of decoupling between BPMN engines and IoT Devices.

[7] https://www.w3.org/TR/vocab-ssn/
[8] https://www.ogc.org/standard/sensorml/
[9] http://iot.ee.surrey.ac.uk/citypulse/ontologies/sao/sao

Next, we introduce additional details about each stage through the motivating example. For replicability purposes, the ontology, BPMN model, and software artefacts developed to implement the motivating example can be found in a GitHub repository[10].

3.1. Ontological Description of IoT Devices

SAREF[11] is a reference ontology for smart devices that provides an important contribution to enabling semantic interoperability in the IoT. Among the different concepts defined in this ontology, we use the following: (1) *Device*, a tangible object designed to accomplish a particular task by means of one or more functions; (2) *Actuating Function*, a function that allows transmitting data to a device, such as level settings (e.g., temperature) or binary switching (e.g., open/close, on/off); (3) *Sensing Function*, a function that allows a device transmitting data autonomously, such as measurement values (e.g., temperature) or sensing data (e.g., occupancy); (4) *Meter Function*, a function that allows getting data from a device at a specific time such as current meter reading or instantaneous demand. The SAREF ontology also introduces concepts to describe the data structure that is managed by each function. However, this issue is left as further work.

As a representative example, Figure 3A graphically shows the SAREF description of the CO2 sensor used in the motivating example. This device is defined with one Sensing Function (`CO2 Level Sensing`), which is the one focused on autonomously transmitting the CO2 level; two Actuating Functions (`Start Notifications` and `Stop Notifications`), which allow to configure whether or not the CO2 level must be transmitted autonomously by the sensor and the periodicity; and one Metter Function (`getCO2Level`), which allows to get the CO2 level from the sensor at a specific instant.

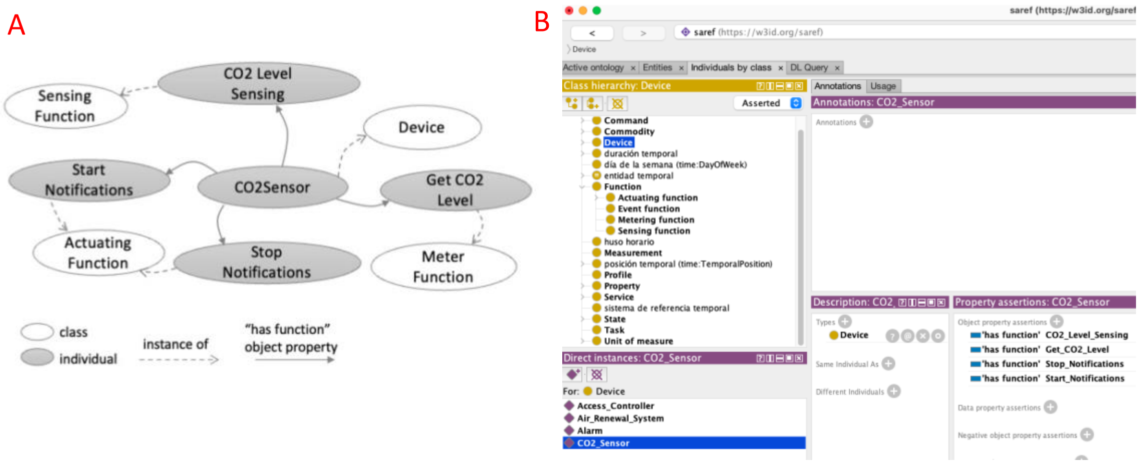


Figure 3. SAREF ontological description of a CO2 Sensor (left) created in Protégé (right)

Currently, there are many tools and languages that can be used to create ontologies (Slimani, 2015). In this work, we used Protégé[12] (see Figure 3B), which is a very popular open-source tool in the field of semantic web and computer science research. In order to

[10] https://github.com/pvalderas/pvalderas-iot-enhanced-bp-saref
[11] https://w3id.org/saref
[12] https://protege.stanford.edu/

integrate the semantic description of IoT devices with the BPMN modeller presented below, it was exported as an OWL file.

3.2. BPMN Modelling

In order to define the BPMN model that represents an IoT-Enhanced BP, we use the modelling approach presented in (Valderas et al., 2022), which has been summarised when introducing the motivating example in Section 2.

In our previous work, we created a web modelling tool[13] that allow us to apply this modelling approach to create BPMN models of IoT-Enhanced BPs. However, this tool includes specific data of the IoT devices into the BPMN models to make them executable, which links these models with the devices, creating a coupled solution. In this work, we have extended the web modelling tool in order to create BPMN models associated with ontological descriptions of IoT devices. This is complemented by the microservice architecture presented in Section 4, which allows the discovery of IoT devices that fit an ontological description and the interaction with them by means of decoupled event-based communication.

A SAREF OWL parser[14] has been developed and integrated into the tool to load ontological descriptions of IoT devices. As commented in Section 2, our modelling approach proposes to include IoT devices as BPMN lanes and the actions of these devices as service tasks. Thus, the extended version of the tool, once a SAREF ontology is loaded, allows the modeller to link each BPMN lane with one of the IoT devices defined in the SAREF ontology (see Figure 4A) and link the Service Tasks with the SAREF Actuating or Meter functions defined for the device (see Figure 4B). Note that Sensing functions are not provided in this wizard since they define the autonomous behaviour of the devices instead of operations that can be called by an external system.

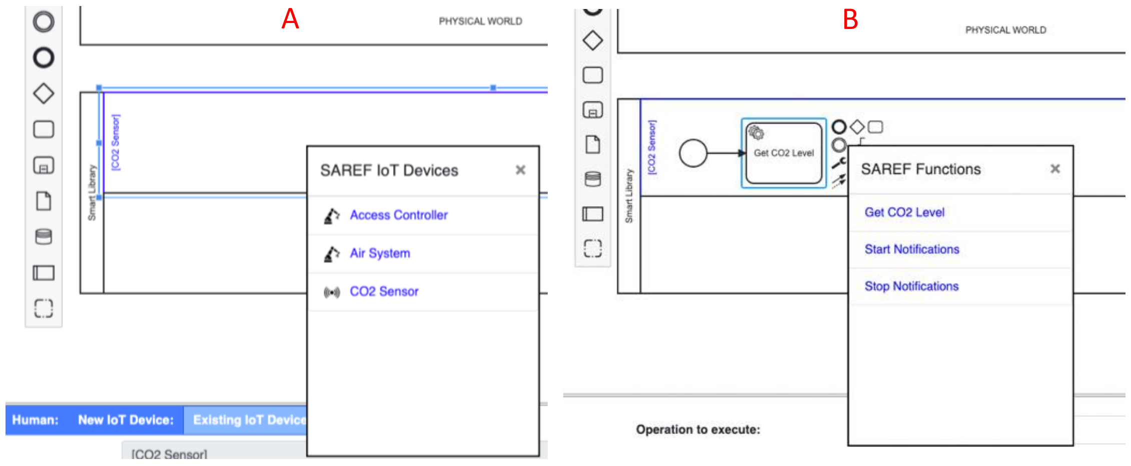


Figure 4. Snapshots of the web modelling tool

The BPMN model created by this tool is totally independent of the data required to interact with IoT devices. It only includes the names of the SAREF devices associated with BPMN lanes and the name of the SAREF functions associated with Service Tasks.

[13] http://pedvalar.webs.upv.es/iot-enhanced-bp-modeller/

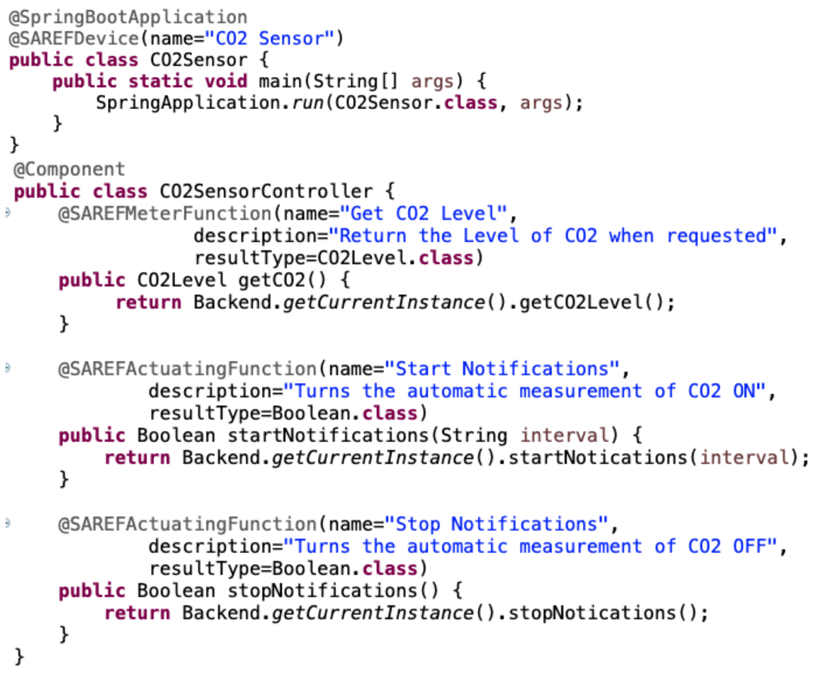
[14] The reader can test the parser by uploading the OWL file available in the GitHub repository (or a analogous one) from the menu option 'IoT System->Create an IoT System from an Ontology'

How this definition is transformed into the actual execution of an operation in a real IoT device is achieved by means of the implementation of the proper microservices, as we explain next.

3.3. Microservice Implementation

The last stage in the process proposed to develop IoT-Enhanced BPs is the implementation of microservices that play the role of instantiations of the devices described through the SAREF ontology in stage 1. These microservices must implement the code required to interact with a real IoT device in order to support the functions defined in the ontological descriptions.

As we explain in the next section, the BPMN engine that executes the BPMN model of an IoT-Enhanced BP will interact with these microservices through an event-based bus. This means that these microservices must react by executing a specific operation in the real IoT device as a response to an event published in a bus by the BPMN engine. These events will represent the request for the execution of an operation in terms of the SAREF ontology.



```
@SpringBootApplication
@SAREFDevice(name="CO2 Sensor")
public class CO2Sensor {
    public static void main(String[] args) {
        SpringApplication.run(CO2Sensor.class, args);
    }
}
@Component
public class CO2SensorController {
    @SAREFMeterFunction(name="Get CO2 Level",
                description="Return the Level of CO2 when requested",
                resultType=CO2Level.class)
    public CO2Level getCO2() {
         return Backend.getCurrentInstance().getCO2Level();
    }

    @SAREFActuatingFunction(name="Start Notifications",
            description="Turns the automatic measurement of CO2 ON",
            resultType=Boolean.class)
    public Boolean startNotifications(String interval) {
        return Backend.getCurrentInstance().startNotications(interval);
    }

    @SAREFActuatingFunction(name="Stop Notifications",
            description="Turns the automatic measurement of CO2 OFF",
            resultType=Boolean.class)
    public Boolean stopNotifications() {
        return Backend.getCurrentInstance().stopNotications();
    }
}
```

Figure 5. Example of IoT Microservice implementation

In order to support this event-based reactive behaviour and facilitate the implementation of the proposed microservices, we have implemented a library based on the Java Spring Boot framework. Inspired by a solution we already used in our previous work (Valderas et al., 2023), this library provides a set of Java annotations that allow microservice developers to semantically annotate the Java code according to the IoT device descriptions done with the SAREF ontology. This library uses reflection mechanisms to detect these annotations and inject the functionality required to respond to the event-based requests performed by the BPMN engine. As a representative example, the code that is shown in Figure 5 implements the microservice that is in charge of managing a CO2 sensor. Note how the main class is annotated with `SAREFDevice` and the methods that implement the interaction with IoT devices are annotated with

`SAREFActuatingFunction` or `SAREFMeterFunction`. These two last annotations link the name of a SAREF function to a specific Java method in such a way that when the BPMN Engine publishes an event requesting the execution of a function (e.g., `Get CO2 level`), the functionality injected by the library is able to know the Java method that must be executed. The code that implements each of these methods depends on the underlying technology of the IoT device, which is out of the scope of this paper. Further details about the event-based reactive behaviour that is supported by this library are presented in the next section.

4. Supporting microservice architecture

The previous section has introduced how IoT-Enhanced BPs are developed by means of the SAREF ontology, BPMN models, and microservices. In this section, we introduce a microservice architecture that integrates these software artefacts in order to execute this type of process in a technology-independent way. This architecture improves the one presented in (Valderas et al., 2022) by supporting event-based communication between BPMN engines and IoT devices. It is shown in Figure 6.

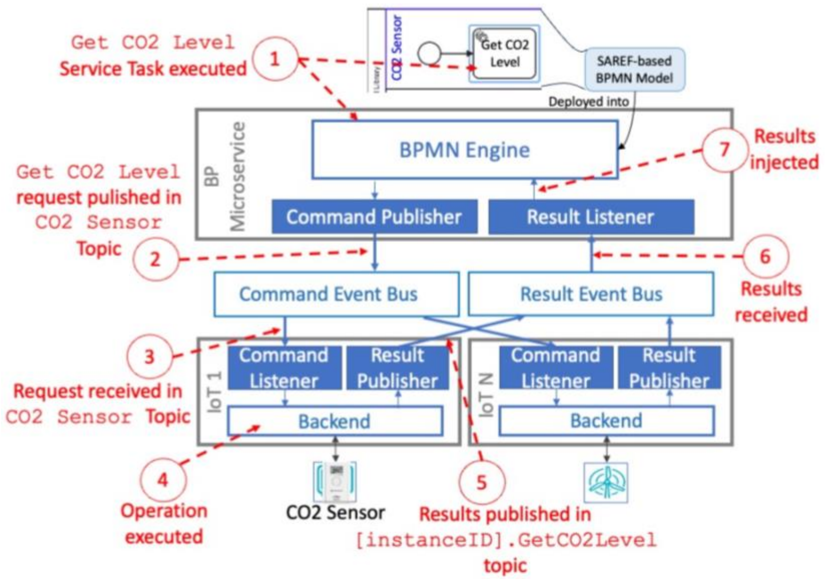


Figure 6. Microservice-based architecture to execute SAREF-based BPMN models of IoT-Enhanced BPs

The microservices included in the proposed architecture are the following:

(1) *BP microservice*, which is endowed with a BPMN engine that oversees executing IoT-enhanced BPs such as the one presented in Figure 1. In addition, the communication adapters *Command Publisher* and *Result Listener* are also deployed into this microservice. These adapters support event-based communication with IoT microservices (presented below). The tasks they perform to achieve this communication are introduced below in more detail. Note that this is an infrastructure microservice that can be used to deploy several IoT-enhanced BPs independently from their domain.

(2) An *IoT microservice* for each IoT device, which supports each SAREF device by implementing a backend that is in charge of managing the interaction with the real device in order to execute operations. This backend is implemented according to the

technology used by the device. Two communication adapters are also included in these microservices to support event-based communication with the BP microservice: *Command Listener* and *Result Publisher*. These adapters contain the functionality that is automatically injected when IoT microservices are implemented by using the Java library presented in Section 3.3.

The communication between the BP microservice and the IoT microservices is asynchronously done through topic-based event buses, which implement publish-subscribe communication (Eugster et al., 2003) through message queues. Publish-subscribe communication allows for the decoupling of producers and consumers, enabling scalability and a loosely coupled architecture. Message queuing ensures reliable and asynchronous communication with features like load balancing and message persistence. In addition, we use these event buses to implement the Command pattern (Dupire & Fernandez, 2001), which is a behavioural pattern used in object-oriented programming. A request is wrapped under an object as a command and passed to an invoker object. The invoker object looks for the appropriate object which can handle this command and passes the command to it to be executed. We adapt this pattern so that the request is created by the BP microservice and published to a topic-based *Command Event Bus* (see Figure 6). This request asks for the execution of a SAREF function. IoT microservices are subscribed to the *Command Event Bus* in such a way the request is handled by the appropriate microservice, which executes the corresponding operation in the IoT device. Once the operation is executed, the obtained result is published by the microservice into a topic-based *Result Event Bus*. The BP microservice is subscribed to this bus in such a way it obtains the results of executing the requested SAREF function.

4.1 Command-based communication in detail

Let's explain in more detail how the request to execute a SAREF function by the BP microservice is transformed into the execution of the operation of a real IoT device by applying the Command pattern (see execution steps depicted in red in Figure 6). A BPMN model such as the one presented in Figure 1 is executed by the BPMN engine deployed into the BP microservice. For each BPMN Service Task that is associated with a SAREF function (step 1), the BP microservice creates a request that asks for the execution of this function in the context of a specific running instance. This request is defined in JSON format by the *Command Publisher* adapter (step 2), and it is published in the topic-based *Command Event Bus*. The topic in which the request is published is the name of the IoT device that must execute it. An example is shown below, which asks for the execution of the `Get CO2 Level` function. This request is published on the `CO2 Sensor` topic. Note how the IoT device that defines the topic and the function included in the request are defined according to the description done with the SAREF ontology. They are obtained by the *Command Publisher* adapter from the name of the Service Task that is being executed by the BPMN engine and the name of the lane in which the task is defined (see the BPMN model in Figure 1). The name of the process is obtained from the BPMN pool. The name of the device is also included in the request for validation purposes.

```
{
  "process": "Smart Library",
  "instance": "582b8617-bcf5",
  "device": "CO2 Sensor",
  "function": "Get CO2 Level"
}
```

Once the request is published, the BP microservice, which is subscribed to the *Result Event Bus*, waits for the publication of the response by the corresponding IoT microservice. IoT microservices are subscribed to the *Command Event Bus* and are waiting for requests published in the topics associated with their name. For instance, the CO2 Sensor microservice is waiting for requests published on the `CO2 Sensor` topic (step 3 in Figure 6). Once a request is published on this topic, the *Command Listener* adapter of this IoT microservice oversees executing the Java method that corresponds to the SAREF function included in the request (step 4). These methods are identified by using the annotations presented in Section 3.3.

Once the Java method that corresponds to the requested SAREF function is executed, the result is managed by the *Result Publisher* adapter of the IoT microservices, which contains the functionality required to publish them into the topic-based *Result Event Bus* (step 5). The BP microservice is subscribed to this bus in order to receive the results of the requested function (step 6), which are managed by the *Result Listener* adapter in order to be injected into the BPMN engine (step 7). In this case, the topic used to communicate the IoT microservices and the BP microservice is defined by a string made up of the process instance ID and the function name. For instance, the response to the previous request is published in the topic `582b8617-bcf5.GetCO2Level` in such a way the BP microservice is able to obtain the response for the execution of a function for a specific running process instance. The response constructed by the *Result Publisher* adapter of each IoT microservice is also based on the SAREF ontology. However, details about this issue are omitted due to space restrictions.

4.2 Proof of concept validation

According to Völter (2006), a way of preliminary evaluating the proposal of a new architecture is through developing a prototype. Next, we introduce a realisation of the above-introduced architecture as a prototype involving mapping technology choices onto the solution concepts.

The main goal of the proposed microservice architecture is to facilitate decoupled interoperability among BPMN engines and IoT devices when the former needs to execute operations of the latest. Thus, in addition to developing a prototype with specific technology choices, we faced changing a participant IoT device in order to analyse the adaptation tasks that are required.

The implemented prototype. For the development of the initial prototype, we considered the running example presented in this paper, and we take as a basis the technology choices we made in previous work (Valderas et al., 2022):

- The event buses were implemented by means of the RabbitMQ Message broker.
- The BP microservice was supported by a Windows system with the community version of the Camunda 7 engine. We also deployed the *Command Publisher* and the *Result Listener* adapters.
  - The *Command Publisher* was developed as a Java library integrated into Camunda in order to manage the execution of each Service Task. This library also contains configurable functionality to facilitate the publication of messages into the *Command Event Bus*. Currently, the RabbitMQ message broker is supported.

    - The *Result Listener* adapter was developed as a Java library that contains configurable functionality to facilitate the subscription to the *Result Event Bus* (currently, RabbitMQ) in order to process the results published by the IoT microservices. This library also includes the functionality required to inject the results obtained after the execution of an operation into the corresponding process instance.
- IoT microservices were implemented in Java by using the library introduced in Section 3.3. Their backends were initially implemented to interact with IoT devices that provide a REST API. Each of these microservices was deployed into Windows systems. The *Command Listener* and *Result Publisher* adapters were also deployed into each microservices. These adapters were developed as Java libraries that contain configurable functionality to facilitate the interaction with RabbitMQ in order to both receive the commands published into the *Command Event Bus* and publish the operation results in the *Result Event Bus*, respectively.

Replacement of an IoT device. Once this prototype was developed, we analysed its correctness through the execution of the BPMN model of the running example and the analysis of logs. Afterwards, we replaced the CO2 Sensor based on REST with an analogous one that uses an MQTT queue to publish data. To support this change, we just needed to reimplement the backend of the IoT microservice that manages the interaction with the CO2 Sensor in order to work with an MQTT queue instead of accessing a REST API. The communication adapters of this microservice did not require any modification since the communication with the event buses is totally decoupled from the backend. Also, no other microservice in the architecture requires an adaptation, neither the BP microservice nor the IoT microservices that manage other devices.

Discussion. The proposed architectural solution was shown to facilitate adaptation when an IoT device is replaced by another with a different technology. The decoupled communication model based on the Command pattern and event-based buses provided a high level of independence among the BPMN engine and the IoT devices. Also, this architectural solution facilitated the simultaneous interaction of the BPMN engine with IoT devices that are technologically different. For instance, once the initial REST-based CO2 Sensor was replaced by the MQTT-based version, Camunda interacted with IoT devices supported by two different technologies (i.e., the CO2 Sensor based on MQTT and the other devices based on REST). However, this was totally transparent for the engine. On the other hand, the use of microservices as gateways between the BPMN engine and the IoT devices allows for providing a common interface for all the IoT devices (based on the defined SAREF functions), although they do not support some operations natively. For instance, the CO2 sensor defined in the SAREF ontology provided three operations: Get CO2, Start Notifications and Stop Notifications. As the newly deployed MQTT-based sensor did not support the start and stop of notifications, we could implement some code that emulates them in the microservices' backend, maintaining a correct interaction with the BPM engine. Note that although some coding tasks were required to replace one IoT device with another, they were totally decoupled from the BPMN engine, and it was not necessary to stop the BPMN engine, adapt the BPMN model, and redeploy it.

5. Related Work

How to deal with the design and implementation of IoT-enhanced BPs has already been addressed in the literature by different authors. In this section, we provide an overview of some of these works and the architecture they present but also consider other works more focused on cyber-physical systems in general and how they deal architectonically with the heterogeneity and changing nature commonly found in this type of systems.

Regarding the specification and execution of IoT-enhanced BPs, most of the proposals we find in the literature (Mandal et al., 2017; Mottola et al., 2019; Suri et al., 2017) focus on providing a supporting infrastructure to specify and execute this type of BP properly. However, none of them addresses the technology decoupling issues along the whole architecture as we do, meaning that we always find solutions that are tight to the underlying technology. These solutions extend existing modelling notations (e.g., BPMN) to represent specific aspects of an IoT system (e.g., a sensor, an actuator, an event triggered by the physical world, etc.) as well as the interaction with them. Thus, BP engines need to be extended to be able to execute such extensions, turning proposals into highly coupled solutions with the BP engine. Our solution, in contrast, can be used with any commercial BPMN engine without the need to extend it to support new constructors.

To decouple all the infrastructure required to specify and execute such systems, other works make a clearer separation between the different layers of their proposed architecture (Kim et al., 2014; Sasirekha & Swamynathan, 2016; Schönig et al., 2018)). For example, Kim et al. (2014) propose to achieve the integration between BPs and IoT devices by means of adapters. In the proposal, there is a four-layer division that separates BPs (BP layer), software services (SW Service layer), adapters (Adaptation layer), and IoT services (IoT Service layer). While the BP layer directly invokes services from the SW Service layer and the Adaptation layer, there is no direct interaction with the IoT Service layer. In fact, this interaction is performed through the Adaptation layer, which includes adapters that respond to services requested by BPs and invoke IoT services in response to such requests. Besides this, IoT services are described according to the SensorML modelling language. Sasirekha & Swamynathan (2016) also propose an architecture of four layers: the BP layer, the semantic service layer, the web service layer, and the device layer, where BPs are specified in BPEL, services using semantic web languages, i.e., ontologies, and IoT devices are represented by means of an ontology which is defined as the integration of a defined domain ontology and the SSN ontology. Finally, Schönig et al. (2018) propose an architecture composed of three layers: the IoT layer, which contains sensors and wearable devices of human participants; the IoT infrastructure and communication middleware; and the BPMS. Besides this architecture, this work proposes to extend BPMN with Data variables to enrich BP models with data obtained from physical objects (e.g., machine status or actor position) and to specify how and where connected IoT devices influence the process. All these approaches try to achieve a more decoupled architecture, although the different layers communicate with each other through direct calls. We improve these solutions by achieving a more decoupled interaction based on the Command pattern and the use of asynchronous event buses based on a publish/subscribe communication.

Regarding the field of cyber-physical systems, we find the works developed by Cao et al. (2021) and Gómez et al. (2021), which propose the use of event-driven architectures (EDA) together with other architectonical solutions to overcome the technology decoupling challenge of such systems. On the one hand, Gómez et al. (2021) propose an architecture based on SOA, microservices, and Complex Event Processing (CEP) to offer a flexible service catalogue that allows them to connect to the system on any kind of device and interact in different scenarios. On the other hand, Cao et al. (2021) propose the use of an ontology model in an event-driven architecture to make sure that the event-driven services provided by organisations follow the organisational event access rules. Specifically, the ontology model is used to define the domain-specific concepts related to events. Although these works propose a similar solution to ours, the use of the Command pattern based on the SAREF ontology provides a higher level of decoupling, in addition to the fact that these solutions do not consider the specification of IoT-enhanced BPs.

6. Conclusions

This work has presented an approach to improve interoperability issues when implementing IoT-Enhanced BPs. It is based on the use of BPMN, the SAREF ontology and a microservice architecture that implements the Command pattern through asynchronous event buses.

Unlike current solutions of commercial BPMN engines, which are technology-coupled since they insert the connection data of IoT devices into the BPMN models, our approach allows an implementation of IoT-Enhanced BPs in which BPMN engines can interact with IoT devices without knowing their underlying technology. This makes it easier to replace IoT devices when requirements change since the BPMN engine and the deployed models do not need to be modified.

As further work, we plan to extend this approach by introducing goal-oriented modelling techniques in such a way the BPMN engine can execute the tasks of an IoT-Enhanced BP by using the IoT devices that better help achieve the specific goals in terms of, for instance, non-functional properties such as performance, energy consumption, user experience, and so on.

Acknowledgement

This work is part of the R&D&I project PID2020-114480RBI00 funded by MCIN/AEI/10.13039/501100011033

References

BPMN. (2010). *Business Process Model and Notation Concepts*. http://www.omg.org/spec/BPMN/20100501

Cao, H., Yang, X., & Deng, R. (2021). Ontology-Based Holonic Event-Driven Architecture for Autonomous Networked Manufacturing Systems. *IEEE Transactions on Automation Science and Engineering*, *18*(1), 205–215.

Daniele, L., García-Castro, R., Lefrançois, M., & Poveda-Villalon, M. S. (2017). *The Smart Applications REFerence Ontology*.

Deehan, N., Lea, N., Richtsmeier, I., Wulf, J., Graham, A., Ruecker, B., Oliver, R., & Ausley, C. (2022). *Make REST calls from Camunda Platform 7 Example*. https://github.com/camunda-community-hub/Make-Rest-Calls-From-Camunda-7-Example

Dupire, B., & Fernandez, E. B. (2001). The Command Dispatcher Pattern. In *8th Conference on Pattern Languages of Programs*. https://hillside.net/plop/plop2001/accepted_submissions/PLoP2001/bdupireandebfernandez0/PLoP2001_bdupireandebfernandez0_1.pdf

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. ACM computing surveys (CSUR), 35(2):114–131.

Gómez, H. D., Garcia-Rodriguez, J., Azorin-Lopez, J., Tomás, D., Fuster-Guillo, A., & Mora-Mora, H. (2021). IA-CPS: Intelligent architecture for cyber-physical systems management. *Journal of Computational Science*, *53*, 101409.

Kim, S. D., Lee, J. Y., Kim, D. Y., Park, C. W., & La, H. J. (2014). Modeling BPEL-based collaborations with heterogeneous IoT devices. *Proceedings - 2014 World Ubiquitous Science Congress: 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, DASC 2014*, 289–294.

Mandal, S., Hewelt, M., & Weske, M. (2017). A framework for integrating real-world events and business processes in an IoT environment. *Lecture Notes in Computer*, *10573 LNCS*, 194–212. https://doi.org/10.1007/978-3-319-69462-7_13

Mottola, L., Picco, G. P., Oppermann, F. J.,…., Tranquillini, S., & Voigt, T. (2019). MakeSense: Simplifying the Integration of Wireless Sensor Networks into Business Processes. *IEEE Trans. on Software Engineering*, *45*(6), 576–596.

Sasirekha, S., & Swamynathan, S. (2016). Collaboration of IoT devices using semantically enabled resource oriented middleware. *ACM International Conference Proceeding Series*, *21-24-Sept*, 98–105.

Schönig, S., Ackermann, L., Jablonski, S., & Ermer, A. (2018). An integrated architecture for IoT-aware business process execution. *Lecture Notes in Business Information Processing*, *318*, 19–34.

Slimani, T. (2015). Ontology Development: A Comparing Study on Tools, Languages and Formalisms. *Indian Journal of Science and Technology*, *8*(24).

Suri, K., Gaaloul, W., Cuccuru, A., & Gerard, S. (2017). Semantic framework for internet of things-aware business process development. *Proceedings - 2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2017*, 214–219.

Torres, V., Serral, E., Valderas, P., Pelechano, V., & Grefen, P. (2020). Modeling of IoT devices in Business Processes: A Systematic Mapping Study. *Proceedings - 2020 IEEE 22nd Conference on Business Informatics, CBI 2020*, *1*, 221–230.

Valderas, P., & Torres, V. (2023). Towards a Semantic Interoperability in IoTEnhanced Business Processes. An Event-Driven Solution based on Microservices. *2023 19th International Conference on Intelligent Environments, IE 2023 - Proceedings*.

Valderas, P., Torres, V., & Serral, E. (2022). Modelling and executing IoT-enhanced business processes through BPMN and microservices. *J. of Systems and Software*, *184*.

Valderas, P., Torres, V., & Serral, E. (2023). Towards an Interdisciplinary Development of IoT-Enhanced Business Processes. *Business & Information Systems Engineering*, 1–24.

Völter, M. (2006). Software architecture: A pattern language for building sustainable software architectures. In *EuroPLoP 2006 - 11th European Conference on Pattern Languages of Programs* (pp. 31–66). Mar.

Weske, M. (2012). Business process management: Concepts, languages, architectures, second edition. *Business Process Management: Concepts, Languages, Architectures, Second Edition*, 1–403.

Zambroski, S., & Pohl, T. (2016). *MQTT Camunda BPMN.*