International Journal of Information Systems and Project Management

Volume 5 | Number 4 Article 4

2017

Using BPMN to model Internet of Things behavior within business process

Dulce Domingos
Universidade de Lisboa

Francisco Martins
Universidade dos Açores

International Journal of
Information Systems and Project Management
ISSN (print):2182-7796, ISSN (online):2182-7788, ISSN (cd-rom):2182-780X
Available online at www.sciencesphere.org/ijispm

Using BPMN to model Internet of Things behavior within business process

Dulce Domingos
LaSIGE, Faculdade de Ciências, Universidade de Lisboa
Campo Grande, Lisboa, 1749-016
Portugal
www.shortbio.org/mddomingos@fc.ul.pt

Francisco Martins
LaSIGE, Faculdade de Ciências, Universidade de Lisboa e
Faculdade de Ciências e Tecnologia, Universidade dos Açores
Rua da Mãe de Deus, Ponta Delgada, 9500-321
Portugal
www.shortbio.org/fmartins@acm.org

Abstract:
Whereas, traditionally, business processes use the Internet of Things (IoTs) as a distributed source of information, the increase of computational capabilities of IoT devices provides them with the means to also execute parts of the business logic, reducing the amount of exchanged data and central processing. Current approaches based on Business Process Model and Notation (BPMN) already support modelers to define both business processes and IoT devices behavior at the same level of abstraction. However, they are not restricted to standard BPMN elements and they generate IoT device specific low-level code. The work we present in this paper exclusivelly uses standard BPMN to define central as well as IoT behavior of business processes. In addition, the BPMN that defines the IoT behavior is translated to a neutral-platform programming code. The deployment and execution environments use Web services to support the communication between the process execution engine and IoT devices.

Keywords:
Internet of Things; Business Process modelling; BPMN; IoT-aware business process.

DOI: 10.12821/ijispm050403

Manuscript received: 3 July 2017
Manuscript accepted: 17 November 2017

1. Introduction

In the last years, organizations have been using more and more business processes to capture, manage, and optimize their activities. A business process is a collection of inter-related events, activities, and decisions points that involve actors and resources and that collectively lead to an outcome that is of value for an organization or a customer [1]. In areas such as supply chain management, intelligent transport systems, domotics, or remote healthcare [2], business processes can gain a competitive edge by using the information and functionalities provided by Internet of Things (IoTs) devices. The IoT is a global infrastructure that interconnects things (physical and virtual). IoT devices connects things with communication networks. These devices can also have capabilities of sensing, actuation, data capture, data storage, and data processing [3].

Business processes can use IoT information to incorporate real world data, to take informed decisions, optimize their execution, and adapt itself to context changes [4]. Moreover, the increase in processing power of IoT devices enables them to become active participants by executing parts of the business logic: IoT devices can aggregate and filter data, and make decisions locally, by executing parts of the business logic whenever central control is not required, reducing both the amount of exchanged data and of central processing [5]. Indeed, sensors and actuators can be combined to implement local flows, without needing central coordination.

However, decentralizing business processes into IoT devices presents two main challenges. First, IoT devices are heterogeneous by nature. They differ in terms of communication protocols, interaction paradigms, and computing and storage power. In addition, business modelers define processes using high-level languages (such as Business Process Model and Notation version 2.0 [6], henceforth simply referred as BPMN), as they must know the domain, but do not need to have specific knowledge to program IoT devices, nor want to deal with their heterogeneity. Therefore, this decentralization requires design as well as execution time support.

At design time, current approaches allow modelers to define both business processes and IoT devices behavior at the same level of abstraction, using, for instance, BPMN-based approaches [7], [8], [9], [10], [11], [12]. BPMN already provides the concepts to define the behavior of various participants, by using different pools. The interaction amongst participants is specified through collaboration diagrams. Supporting the execution of these hybrid processes requires bridging the gap between high-level BPMN and the programming code that IoT devices can execute. These approaches use a three-step procedure: (1) translation of the process model to a neutral intermediate language; (2) translation of the intermediate code to a platform specific executable code; and (3) deployment of the executable code into IoT devices.

By taking advantage of these approaches, business modelers can define both business processes and IoT behavior at the same (high) level of abstraction. However, they still use non-standard BPMN to integrate, for instance, IoT device information into business processes and they generate IoT device specific code, so that it must be generated again for each different IoT device.

The proposal we present in this paper only uses standard BPMN to define both central and IoT behavior of business processes. Besides using pools and collaboration diagrams, we use the BPMN resource class to integrate the information about IoT devices into the model, and we use the BPMN performer class to define the IoT devices that will be participants of the process. In addition, the BPMN that defines the IoT behavior is translated into Callas bytecode [13]. We use the Callas sensor programming language as an alternative to the target platform-specific languages taken by previous proposals, since it can be executed in every IoT device for which there is a Callas virtual machine available. This way, we abstract hardware specificities and make executable code portable among IoT devices from different manufacturers. Business process and IoT devices communicate via web services (directly or indirectly through gateways). In addition, Callas also supports remote IoT devices reprogramming, a feature that is the first step to support ad-hoc changes [14] in the parts of business processes that define IoT behavior. A preliminary version of this work can be found elsewhere [15].

This paper is organized as follows. Section 2 presents the related work. Our proposal is described in the following two sections: whereas section 3 describes how we support modelling IoT behavior within BPMN business processes, section 4 focuses on the implementation aspects, including the description of the translation procedure into Callas source code, and the overview of the deployment and execution phases. Finally, section 5 concludes the paper and hints for future work directions.

2. Background

Business modelers define business processes with languages such as Web Services Business Process Execution Language (WS-BPEL) [16] or BPMN, which use an abstraction level closer to the domain being specified. At this level, modelers should not deal with IoT devices heterogeneity and specificities: IoT devices use different operating systems (e.g., TinyOS, Contiki [17]), different programming languages (e.g., nesC [18], Protothreads), and different communication protocols. Traditionally, web services are used to provide IoT information and functionalities, abstracting and encapsulating low-level details. This way, web services are the glue between IoT and business processes, as model languages already support their usage. In addition, more recent approaches take a step forward by supporting IoT behavior definition within the business process [19], [20].

2.1 IoT as web services – the centralized approach

Current IoT technology exposes IoT information and functionalities as web services, facilitating interoperability and encapsulating heterogeneity and specificities of IoT devices. Zen, Guo, and Cheng survey two approaches to implement IoT web services [21]. Some works provide web services directly in IoT devices: they simplify, adapt, and optimize Service-Oriented Architecture (SOA) tools and standards to deal with the well-known limitations of resource-constrained devices. Other approaches provide web services indirectly through middleware systems. This way, IoT devices that do not support web services can still be accessed.

Taking a step forward on integrating IoT into business processes, some authors propose the explicit integration of IoT concepts into business process models. Domingos et al. [22], [23] and George and Ward [24] extend WS-BPEL with context variables to monitor IoT information, abstracting the set of operations to interact with IoT devices. The IoT-A project proposes some BPMN extensions to explicitly include IoT devices and their services in an IoT-aware process model, as well as some characteristics of IoT devices, such as uncertainty and availability [25], [26]. The uBPMN project adds ubiquitous elements to BPMN: it defines the BPMN Task extension for Sensor, Reader, Collector, Camera, and Microphone, as well as an IoT-driven Data Object to represent the data transmitted from IoT devices [27], [28].

GWELS [29] provides a graphical user interface to design IoT processes and send them automatically to IoT devices as a sequence of operation calls that have been uploaded to IoT devices in advance. It uses proprietary communication protocols to interact with IoT devices. IoT processes are provided as web services and, in this way, can also be integrated into business processes.

The above approaches assume a centralized control of the process execution, where a single central system executes and coordinates processes and communicates with IoT devices using web services. However, business modelers are unable to define the behavior of IoT devices, they can only use services whose behavior is previously defined.

2.2 IoT as active participants of business processes – a decentralized approach

In a decentralized approach, IoT devices can work together to execute parts of business processes, reducing the number of exchanged messages and promoting the scalability of central process engines, since information is processed locally. Another important advantage, present in many scenarios, is that the lower network traffic between the central engine and IoT devices improves battery lifetime of IoT devices. To model business processes according to this decentralized approach, business modelers need a unified framework where they can specify the behavior of IoT devices as well as

their interactions with the central system. BPMN already provides the concepts to define the behavior of various participants by using different pools; their interactions are specified through collaboration diagrams.

Following a decentralized approach, Caracas and Bernauer [7], [8], [9] use standard BPMN to model both central and IoT behavior. However, IoT device information is integrated to the BPMN model in a non-standard way, by appending it to the pool name or with additional attributes added to the pool element. They translate the BPMN that defines the IoT behavior to target IoT device specific code. The authors state that the sensor code they generate does not perform much worse than hand-written code.

Casati et al. [10] propose the makeSense framework. They extend BPMN with attributes to support the new intra-WSN participant, which contains the part of the process that IoT devices will execute. To model IoT behavior, makeSense uses a second meta model [10], [30], [12]. The translation procedure into executable code for IoT device uses two models: the application capability model has information about available sensors and actuators and their operations, while the system capability model has additional information about the target IoT devices, which is used to generate different code based on IoT device capabilities. MakeSense uses its own message format and transmission encoding to support the communication between the central process engine and IoT devices [12].

Whereas in these two proposals, IoT devices execute device specific code, Pryss et al. [31] follow a different approach by executing process engines in IoT devices. Despite the advantage of avoiding generating the executable code for IoT devices, this option is only applicable for IoT devices with higher computational capabilities.

In our work, we exclusivelly use standard BPMN to define all the business process, and IoT device information is added to the model by using the BPMN resource class. We translate the BPMN that defines the IoT behavior into Callas bytecode [13], a non-platform-specific language that IoT devices with an available Callas virtual machine can execute.

3. Modelling the behavior of IoT devices

This section describes how we use the BPMN language to model IoT behavior within business processes, both at the same level of abstraction. It starts by presenting the use case scenario we choose to illustrate the application of our proposal.

3.1 Use case scenario

Fig. 1 presents our use case scenario, a simplified process for automatic irrigation control. The process includes three participants: the central process (named Irrigation) and two IoT devices, the IoT irrigation device and the IoT read rainfall device. The behavior of each participant is modelled in separated pools.

The IoT read rainfall device, periodically, wakes up. Its process starts by reading the rainfall sensor, and only sends a message to the IoT irrigation device if it is not raining; otherwise, it stops. When the IoT irrigation device receives the message from the IoT read rainfall device, it starts the irrigation by activating an actuator, which lasts for a pre-defined period. Upon finishing the irrigation, the IoT irrigation device reads the flowmeter sensor to make sure it is sealed. If water still flows, it sends an alert to the central process. This way, the central process receives a notification when the IoT irrigation device detects a water leek that needs human intervention to be fixed.

This simplified process omits a lot of details, such as the functionalities to change both timers (the irrigation interval and duration), for instance.

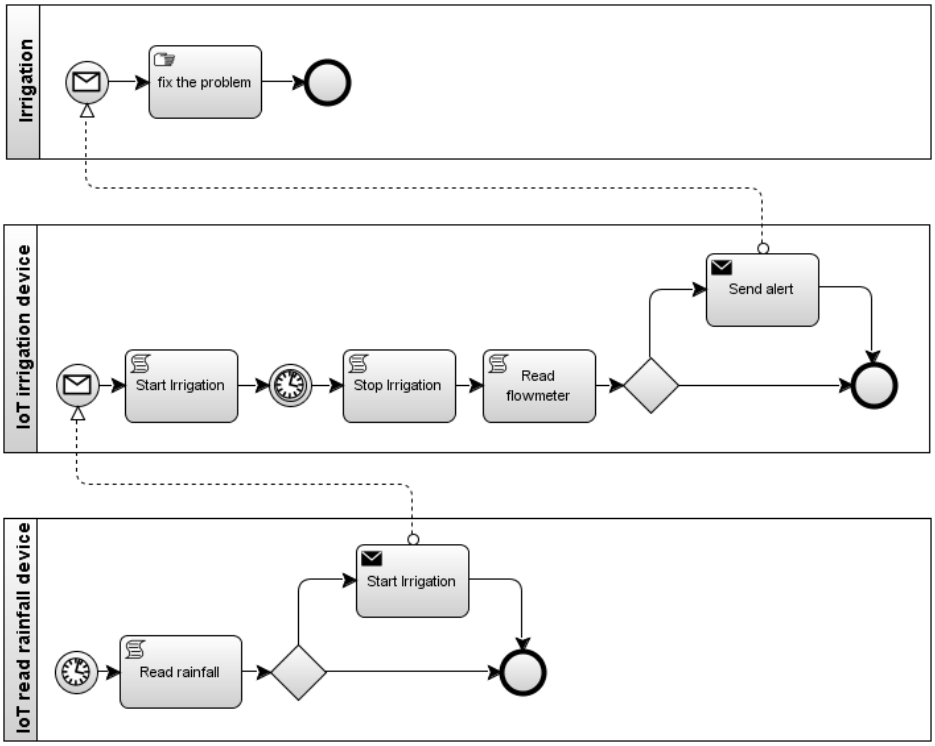

Fig. 1. BPMN use case scenario

3.2 Using BPMN to model the behavior of IoT devices

To model all the business process, including the behavior of IoT devices, we only use standard BPMN elements. BPMN already provides the concepts to define the behavior of various participants, by using different pools, as well as the interaction amongst participants, through collaboration diagrams. This approach is illustrated making use of our use case scenario described in the previous subsection.

We select a subset of BPMN to model the behavior of IoT devices, avoiding the use of two different meta models. The selection of the subset considers two main factors:

- To model the behavior of IoT devices, business modelers do not need all BPMN elements, as stated by Caracas [9]. This way, we include in our subset the BPMN elements that Caracas use to model the IoT programming patterns, and
- Callas already considers general IoT devices limitations, for instance, it does not support parallel tasks. In addition, it is a block-structured language, consequently, it also does not support unstructured control flow, unlike BPMN, which allows gateways to loop back and forward. Fig. 2 illustrates an example of a flow control that we do not support in IoT behavior definitions, since there is no way to represent the flow from Task B to Task A.

The BPMN subset includes the following elements:

- Flow control: events (message received, timers, and the start and end events), activities (script task, send task, and receive task), and gateways (exclusive gateway and merging exclusive gateway);
- Connecting objects: sequence flow, message flow, and data associations; and
- Data: data objects.

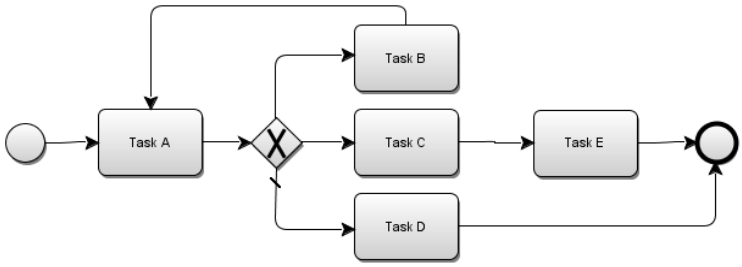

Fig. 2. Example of a non-block control flow structure

We use script tasks to define the tasks that corresponds to invocations of hardware functionalities of IoT devices. For instance, in our use case example, IoT read rainfall device has a rainfall sensor, and we use the script task Read rainfall to model the sensor data acquisition. In a similar way, within the process of the IoT irrigation device, we also use a script task (named Start irrigation) to model the activation of the actuator that starts the irrigation.

The BPMN Resource class and the BPMN Performer class are used to define the IoT device that will execute the processes, avoiding the integration of this information into BPMN models in a non-standard way. The Resource class is used to specify resources (i.e., IoT devices) that can be referenced by processes, whereas the Performer class defines the resource that will perform the processes. Fig. 3 presents a simplified example based on our use case scenario, which can be reused in other business processes. It includes the definition of the resource named IoTdevice with three parameters: deviceType, address, and operations. The performer definition exemplifies the way we apply the deviceType resource parameter in queries for resource assignment. We use the operations parameter to access to the list of the hardware functionalities of the IoT device and their signatures, i.e., the name, its return type, and the type of its parameters.

```
<resource id="IoTdevice" name="IoTdevice">
  <resourceParameter id="deviceType" name="deviceType" type="xs:string"/>
  <resourceParameter id="address" name="address" type="xs:string"/>
  <resourceParameter id="operations" name="operations" type="ns1.xsd:tOperations"/>
</resource>
…
<performer id="performer" name="performer">
   <resourceRef>IoTdevice<resourceRef>
   <resourceParameterBinding parameterRef="deviceType">
     <formalExpression>IoT irrigation device<formalExpression>
   </resourceParameterBinding>
</performer>
```

Fig. 3. Example of using the BPMN resource class and the BPMN performer class

4. Implementing the behavior of IoT devices

The implementation phase includes the translation of BPMN processes into Callas bytecode (the executable program for IoT devices), and how to deploy and execute it in IoT devices.

4.1 Translating BPMN to Callas code

Unlike related work approaches that use platform-specific code to specify the behavior of IoT devices, we translate it into Callas bytecode [13]. By choosing the Callas programming language, we take advantage of some of its characteristics and functionalities specifically tailored to address IoT devices. For instance, Callas takes, as a pattern, the path followed by the Java programming language, and proposes a virtual machine for IoT devices that abstracts hardware specificities and makes executable code portable among IoT devices from different manufacturers. The Callas language is founded on a well-established formal semantics and, along with its virtual machine, statically guaranties

that well-typed programs are free from certain runtime errors (type-safety). Moreover, the Callas virtual machine is sound, that is, its semantics corresponds to that of the Callas language. It includes domain-specific IoT operations, such as for sending and for receiving messages from the network. These Callas operations are supported directly at the Callas virtual machine level, and may have different implementations depending on the hardware where it is deployed. Currently the Callas virtual machine is available for SunSpot, TinyOS, Arduino, VisualSense, and Unix platforms (more information can be found in the Callas website http://www.dcc.fc.up.pt/callas/). Other interactions with the hardware of IoT devices are performed by calling external functions of the language. This typically corresponds to operating system calls or to direct implementations (on the bare metal) of functions on the Callas virtual machine. The operations made available by each kind of device is described by a type in the Callas language, allowing the compiler to verify if the source code is adequate to run on a specific target device. A distinguished feature of the language is its ability to deploy executable code, which we use to install the code in IoT devices, remotely. We also consider this feature as the first step to support ad-hoc changes [14] in IoT business process parts.

Fig. 4 presents the Callas source code that implements the behavior of the IoT devices of our case study. The left column has the source code that corresponds to the behavior of the IoT read rainfall device, whereas the right column has the source code that corresponds to the behavior of the IoT irrigation device. Programs start by declaring two module types: Nil, an empty module type used to represent void function returns; and a second module that defines the message signatures that flow on IoT devices.

The implementation of the IoT_read_rainfall_device spans from line 9 to line 17. Each function is implemented using the def construct. The checkIrrigation function reads the rainfall sensor (using the readRainfall external function) and binds it to the isRaining variable. Then, in case it is not raining, the function continues by sending a message to start the irrigation (send startIrrigation). The main program (lines 19–24) loads the module into the device's memory and installs a timer to call checkIrrigation every day.

Within the implementation of the IoT_irrigation_device, upon receiving the startIrrigation message, it invokes the startIrrigation external function. The system irrigates for 20 minutes, stops by calling the stopIrrigation function, and checks whether the water valve is sealed. It alerts the central process in case there is a water leak (send waterLeakAlert).

```
defmodule Nil:                                                  1
  pass                                                          2
                                                                3
defmodule IoT_irrigation_system :                               4
  Nil startIrrigation ()                                        5
  Nil waterLeakAlert ()                                         6
                                                                7
# declare module IoT_read_rainfall_device                       8
module m of IoT_irrigation_system:                              9
  def checkIrrigation (self):                                   10
    isRaining = extern readRainfall ()                          11
    if not isRaining :                                          12
        send startIrrigation ()                                 13
  def startIrrigation (self):                                   14
    pass                                                        15
  def waterLeakAlert (self):                                    16
    pass                                                        17
                                                                18
mem = load                  # load the device memory            19
newMem = mem || m           # update with Sampler module m      20
store newMem                # replace the device memory         21
                                                                22
# invoke tasks every day                                        23
checkIrrigation () every 24∗60∗60∗1000                          24
```

```
defmodule Nil:                                                  25
  pass                                                          26
                                                                27
defmodule IoT_irrigation_system:                                28
  Nil startIrrigation ()                                        29
  Nil waterLeakAlert ()                                         30
                                                                31
# declare module IoT_irrigation_device                          32
module m of IoT_irrigation_system:                              33
  def startIrrigation (self):                                   34
    extern startIrrigation ()                                   35
    sleep (20∗60∗1000)                                          36
    extern stopIrrigation ()                                    37
    curFlowmeter = extern readFlowmeter ()                      38
    if curFlowmeter > 0                                         39
      send waterLeakAlert ()                                    40
  def waterLeakAlert (self):                                    41
    pass                                                        42
                                                                43
mem = load                  # load the device memory            44
newMem = mem || m           # update with Sampler module m      45
store newMem                # replace the device memory         46
```

Fig. 4. The Callas code of the use case scenario

The translation from BPMN to Callas is divided into three phases: (1) parse of BPMN XML files; (2) syntactic and semantic validations of BPMN processes; and (3) translation into Callas.

The parsing phase takes the BPMN XML file and creates an abstract syntax tree (AST) representation. At this phase, we rule out programs with constructs that are not compliant with the BPMN subset we define for modelling IoT devices' behavior.

The second phase traverses the AST multiple times for validations: it verifies that (a) the control flow is plausible for being translated into a block-structured language, and that (b) the domain-specific script tasks are valid. The former checks that tasks and events only have one input and one output sequence flow and that every possible control flow path from an outgoing exclusive gateway arrives at the corresponding incoming exclusive gateway. Otherwise, it denotes a control flow only possible to represent using a goto statement (absent in Callas). This validation is challenging, since we need to figure out the correspondence between the outgoing and the incoming exclusive gateways, and that control flow may include various exclusive gateways (corresponding to nested if statements in structured languages). As for the latter, valid domain-specific script tasks have well-known names fixed in advance, associated with information specific per each task.

Semantic validation checks that the domain-specific tasks are used correctly on what concerns the types of the data objects being used and that these types are in conformance with the domain-specific task signatures (also provided in advance).

The translation from the AST into Callas proceeds as follows:

- Event elements: (a) message received start events are translated into function calls and the process triggered by these events are translated into function definitions. We figure out the function signature from the types of the values in the message. The function name can be set arbitrarily, since the whole interaction process is going to be generated automatically at compile time and set for both participants (the BPMN engine or the IoT devices). In Fig. 4 we do not use arbitrarily function names for the sake of legibility. Upon message reception, the Callas virtual machine takes the responsibility of invoking the correct message handler function. For instance, upon reception of an irrigation message by the IoT device, the Callas virtual machine dispatches it to the startIrrigation function that implements the behavior of the IoT irrigation device; (b) There are two distinct behaviors for business process timers, depending whether they are attached to starting events or used in the middle of processes to model the passing of time. The former are directly supported by a Callas timer that invokes a function encoding the timer behavior (*vide* line 24, Fig. 4). The latter, is naturally implemented using the Callas delay function; (c) the end event is twofold: when it happens inside a process, it is translated into a return statement inside the function that models the process; when it marks the end of the top-level process there are multiple ways to interpret it. The simplest is just to ignore the event, since we can think of a IoT program as a never-ending program, always ready to handle new requests. The other extreme is to end the Callas virtual machine, but then we need to get explicit access to the IoT device hardware to reset it, or put in place an additional software procedure to reset the IoT devices remotely. We have decided to follow the former option and simply ignore the end event;
- Activity elements: send and receive tasks have a direct correspondence with the send and receive constructs from the language. The script tasks we support are predefined and implemented as part of a Callas library for the BPMN subset. As an example, the Read rainfall script tasks is translated into a call to the hardware functionality named readRainfall (*vide* line 11, Fig. 4);
- Gateway elements: Exclusive gateways are represented by if statements. In case the gateway has more than two alternatives, it is translated into a series of nested if−else statements. Merging exclusive gateways are ignored during the translation process, since their behavior is captured by the Callas sequential composition statement. It is just used during semantic validation as described above;

- Connecting object elements: sequence flows are modelled by Callas control flow mechanisms, which is sequential composition, branching, looping, and function calls. The validation phase guarantees that the IoT BPMN model can be represented by these control flow primitives. Message flows are initiated with a send task, concluded with a receive task, and the flow itself is supported by the underlying data layer of the communication protocol stack of IoT devices;
- Data elements: data objects and their associations are modelled by Callas program variables that store objects and, when used in expressions, represent data associations, capturing the data flows specified at the BMPN model.

During the whole translation procedure, we keep track of each BPMN element identification, defined in the XML model file. We use them to map the errors we unveil during the validation and the translation phases, and report them to modelers in the BPMN design tool context.

4.2 Deployment and execution

Deployment and execution phases include the installation of Callas bytecode in IoT devices as well as the creation of the web services to support the communication between the process execution engine (jBPMN) and IoT devices.

The steps our deployment algorithm performs are the following:

- Generate the Callas code and deploy it to the IoT device by invoking the install code service. For that, we provide the target IoT device identification taken from the IoT device database, by using a query based on the parameters of the resource, and the bytecode produced during the Callas compilation;
- Remove IoT pools from the BPMN model file, since this behavior is going to be performed by IoT devices, instead of the jBPM server;
- Update the BPMN model file by setting send message tasks (or throw events) that target the IoT pool to use IoT web services, providing its address; and
- For each receive message tasks (or catch event) that initiates at an IoT pool, we take its address and pass it to the IoT devices so they can deliver messages using the jBPM RESTful API.

4.3 Prototype

In our prototype, we use the Eclipse IDE [32] and the jBPM [33], a BPMN server from RedHat. Our irrigation use case is composed of two types of components: the IoT devices and the application. For the IoT side, the one installed in the garden, we devised a hardware board for controlling irrigation. The board uses the ATmega644PA processor from Atmel corporation. We adapted the Callas virtual machine for this processor and programmed a firmware that controls the garden's irrigation following a programmable schedule. The devices address directly jBPM via its RESTful API; likewise, the application can address each IoT device using its service address. The application includes several irrigation related processes modeled and deployed with our proposal. We currently have the prototype deployed at Avenida da Liberdade in collaboration with Lisbon city council, where we control four electrovalves managing a total of 40 sprinklers. The project is running with success for three months.

5. Conclusion and future work

The IoT opens an opportunity to create a new generation of business processes that can benefit from IoT ubiquity, taking advantage of their computational power, networking, and sensing capabilities. IoT devices can even execute parts of the business logic.

The work we present in this paper allows modelers to define IoT behavior within business process and with the same level of abstraction, by only using standard BPMN. By translating BPMN, the IoT behavior part, into Callas, we generate neutral-platform portable code, which can also be sent to IoT devices remotely. Our approach opens new opportunities to bring together process modelling and information and functionalities provided by IoT devices.

Modelers do not need to cope with IoT specificities, and use BPMN without any extensions; Callas allows to abstract from the hardware, making the generated code able to run on different devices, if these devices offer the required functionalities. Moreover, the Callas ability for remote reprogramming facilitates code deployment and adds support for dynamic ad- hoc business process changes.

As future work, we want to support the automatic decomposition of business process to determine which process parts can be executed by IoT devices.

In addition, this decentralized approach brings new challenges considering security, as we need to assure confidentiality and authenticity between central process and IoT devices as well as between IoT devices.

Acknowledgments

This work is partially supported by National Funding from FCT - Fundação para a Ciência e a Tecnologia, under the projects PTDC/EEI-ESS/5863/2014 and UID/CEC/00408/2013. We thanks to Fábio Ferreira for implementing part of the prototype.

References

[1] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, "Introduction to business process management," *Fundamentals of Business Process Management*. Springer Berlin Heidelberg, pp. 1-31, 2013.

[2] A. R. Ahlan and B. I. E. Ahmad, "An overview of patient acceptance of Health Information Technology in developing countries: a review and conceptual model," International Journal of Information Systems and Project Management, vol. 3, no. 1, pp.29-48, 2015.

[3] International Telecommunication Union, Overview of the Internet of Things. Recommendation ITU-T Y.2060, 2012.

[4] R. Jedermann and W. Lang, "The benefits of embedded intelligence–tasks and applications for ubiquitous computing in logistics," in *The Internet of Things*, Springer, 2008; pp. 105–122.

[5] S. Haller, S. Karnouskos and C. Schroth, "The Internet of Things in an Enterprise Context," in *Future Internet Symposium*, 2008, pp. 14–28.

[6] OMG, "Business Process Model and Notation (BPMN), Version 2.0", Technical report, Object Management Group, 2011.

[7] A. Caracas, "From business process models to pervasive applications: Synchronization and optimization," in *IEEE International Conference on Pervasive Computing and Communications* (PERCOM Workshops), IEEE, 2012, pp. 320–325.

[8] A. Caracas and A. Bernauer, "Compiling business process models for sensor networks," in *International Conference on Distributed Computing in Sensor Systems and Workshops* (DCOSS), 2011, pp. 1-8.

[9] A. Caracaş, "Modeling, compiling, and efficiently executing business processes on resource-constrained wireless sensor networks," PhD. dissertation, Eth Zurich, 2012.

[10] N. Casati, F. Daniel, G. Dantchev, J. Eriksson, N. Finne, S. Karnouskos, P. Montero, L. Mottola, F. Oppermann, G. Picco, A. Quartulli, K. Römer, P. Spiess and S. Tranquillini, "Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks," in *International Conference on Software Engeneering* (ICSE), 2012.

[11] C. T. Sungur, P. Spiess, N. Oertelad Kopp O. Extending BPMN for wireless sensor networks. Conference on Business Informatics (CBI). IEEE, 2013, pp. 109–116.

[12] S. Tranquillini, P. Spieß, F. Daniel, S. Karnouskos, F. Casati, N. Oertel, L. Mottola, F. J. Oppermann, G. P. Picco, K. Römer and T. Voigt, "Process-based design and integration of wireless sensor network applications. *Business Process Management*, Springer, pp. 134–149, 2012.

[13] L. Lopes and F. Martins, "A safe-by-design programming language for wireless sensor networks," *Journal of Systems Architecture – Embedded Systems Design*, vol. 63, pp. 16-32, 2016.

[14] H. Schonenberg, R. Mans, N. Russell, N. Mulyar and W. N. van der Aalst, "Towards a Taxonomy of Process Flexibility," in *CaiSE Forum*, 2008, vol. 344, pp. 81–84.

[15] F. Martins and D. Domingos, "Modelling IoT Behaviour Within BPMN Business Processes, " *Procedia Computer Science*, vol. 121, pp 1014-1022, 2017.

[16] OASIS, "Web Services Business Process Execution Language Version 2.0", Technical report, Organization for the Advancement of Structured Information Standards, 2007.

[17] A. Dunkels B. Grönvall and T. Voigt, "Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors," in *First IEEE Workshop on Embedded Networked Sensors* (Emnets-I), 2004.

[18] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer and D. Culler D, "The nesC Language: A Holistic Approach to Network Embedded Systems," in *ACM SIGPLAN Conference on Programming Language Design and Implementation* (PLDI), ACM Press, 2003, pp. 1–11.

[19] C. Chang, S. N. Srirama and R. Buyya, "Mobile cloud business process management system for the internet of things: a survey," *ACM Computing Surveys* (CSUR), vol. 49, no. 4, pp. 1-42, February, 2017.

[20] J. Mass, C. Chang and S. N. Srirama, "Workflow Model Distribution or Code Distribution? Ideal Approach for Service Composition of the Internet of Things," in *IEEE International Conference on Services Computing* (SCC), IEEE, 2016, pp. 649-656.

[21] D. Zeng, S. Guo and Z. Cheng, "The Web of Things: A Survey" (Invited Paper). *Journal of Communications*, vol. 6, 2011.

[22] D. Domingos, F. Martins and C. Cândido, "Internet of Things Aware WS-BPEL Business Process," in *International Conference on Enterprise Information Systems*, 2013, pp. 505–512.

[23] D. Domingos, F. Martins, C. Cândido and R. Martinho, "Internet of Things Aware WS-BPEL Business Processes Context Variables and Expected Exceptions," *Journal of Universal Computer Science*, vol. 20, no. 8, pp. 1109-1129, 2014.

[24] A. A. George and P. A. S. Ward, "An architecture for providing context in WS-BPEL processes," in *Conference of the center for advanced studies on collaborative research: meeting of minds* (CASCON), ACM Press, 2008, pp. 22:289–22:303.

[25] S. Meyer, K. Sperner, C. Magerkurth and J. Pasquier, "Towards modeling real-world aware business processes," in *Second International Workshop on Web of Things*, ACM, 2011.

[26] S. Meyer, A. Ruppen and C. Magerkurth, "Internet of Things-Aware Process Modeling: Integrating IoT Devices as Business Process Resources," *Advanced Information Systems Engineering*. Springer, pp. 84–98, 2013.

[27] A. Yousfi, C. Bauer, R. Saidi and A. K. Dey, "uBPMN: A BPMN extension for modeling ubiquitous business processes," *Information and Software Technology*, vol. 74, pp. 55-68, 2016.

[28] A. Yousfi, A. de Freitas, A. K. Dey and R. Saidi, "The use of ubiquitous computing for business process improvement," *IEEE Transactions on Services Computing*, vol. 9, no. 4, pp. 621-632, 2016.

[29] N. Glombitza, M. Lipphardt, C. Werner and S. Fischer, "Using graphical process modeling for realizing SOA programming paradigms in sensor networks," in *Sixth International Conference on Wireless On-Demand Network Systems and Services* (WONS), 2009, pp. 61 –70.

[30] F. Daniel, J. Eriksson, N. Finne, H. Fuchs, A. Gaglione, S. Karnouskos, P. M. Montero, L. Mottola, F. J. Oppermann, G. P. Picco, K. Römer, P. Spieß, S. Tranquillini and T. Voigt, "makeSense: Real-world Business Processes through Wireless Sensor Networks," in *4th International Workshop on Networks of Cooperating Objects for Smart Cities* (CONET/UBICITEC), 2013, pp.58-72.

[31] R. Pryss, M. Reichert, A. Bachmeier and J. Albach, "BPM to go: Supporting business processes in a mobile and sensing world," in *BPM Everywhere: Internet of Things, Process of Everything*, Layna Fischer (Ed.). Future Strategies, FL, pp. 167–182, 2015.

[32] Eclipse IDE for Java EE Developers, 2014. Available: http://www.eclipse.org/. Page visited on May 10th, 2017.

[33] Redhat, *Jboss jBPM*. Available: www.jboss.org/jbpm/. Page visited on May 10th, 2017.

Biographical notes

Dulce Domingos
She received the BSc in "Informática" from Faculdade de Ciências da Universidade de Lisboa, Portugal, in 1993, the MSc degree in "Engenharia Electrotécnica e de Computadores" from Instituto Superior Técnico da Universidade Técnica de Lisboa, Portugal, in 1997, and the PhD degree in "Informática" from Faculdade de Ciências da Universidade de Lisboa, Portugal, in 2005. She is an Assistant Professor at the Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa and a senior researcher of the Large Scale Computer Systems Laboratory (LaSIGE). Her current research interests include security, business processes, and Internet of Things (IoT). She is the coordinator of the master program in information security of Faculdade de Ciências, Universidade de Lisboa and Pró-rector at Universidade de Lisboa.

www.shortbio.org/mddomingos@fc.ul.pt

Francisco Martins
Francisco Martins is an Assistant Professor at University of Azores. Until September 2017, he was an Assistant Professor at the Department of Informatics, Faculty of Sciences, University of Lisbon. Until September 2006, he was Assistant Professor at the Department of Mathematics, University of Azores where he began teaching (as teaching assistant) in October 1997. Previously he was an I. T. manager at Banco Comercial dos Açores since 1990. Francisco received his Ph.D. in Computer Science at University of Lisbon (Faculty of Sciences) in 2006, his M.Sc. (by research) in Computer Science at University of Azores in 2000, and his B.Sc. in Mathematics and Informatics at the University of Azores in 1995.

www.shortbio.org/fmartins@acm.org