Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 121 (2017) 1014–1022

Procedia Computer Science

www.elsevier.com/locate/procedia

CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies, CENTERIS / ProjMAN / HCist 2017, 8-10 November 2017, Barcelona, Spain

Modelling IoT behaviour within BPMN Business Processes

Francisco Martins, Dulce Domingos*

Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, Portugal
LaSIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal

Abstract

As the computational power of IoT (Internet of Things) devices increases, business processes can use them to provide information about real world as well as to execute part of business processes, reducing the amount of exchanged data and central processing. Current BPMN-based approaches already support modellers to define both business processes and IoT devices behaviour 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 only uses standard BPMN to define both central and IoT behaviour of business processes. In addition, the BPMN that defines the IoT behaviour is translated to a neutral-platform programming code.

Peer-review under responsibility of the scientific committee of the CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies.

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

* Corresponding author. Tel.: +351 217500524
E-mail address: mddomingos@fc.ul.pt

1877-0509
Peer-review under responsibility of the scientific committee of the CENTERIS - International Conference on ENTERprise Information Systems / ProjMAN - International Conference on Project MANagement / HCist - International Conference on Health and Social Care Information Systems and Technologies.
10.1016/j.procs.2017.11.131

1. Introduction

Nowadays, organisations use more and more business processes to capture, manage, and optimise their activities. In areas such as supply chain management, intelligent transport systems, domotics, or remote healthcare, business processes can gain a competitive edge by using the information and functionalities of IoT devices (sensors and actuators). Business processes use IoT information to incorporate real world data, to take informed decisions, optimise their execution, and adapt itself to context changes [14]. In addition, the increase in processing power of IoT devices enables them to take part in the execution of the business logic. This way, 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 [13].

However, on the one hand, IoT devices are heterogeneous by nature. They differ in terms of communication protocols, interaction paradigms, and computing and storage power. On the other hand, business modellers define processes using high-level languages (such as Business Process Model and Notation version 2.0 [19], 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.

Current approaches allow modellers to define both business processes and IoT devices behaviour at the same level of abstraction, using, for instance, BPMN-based approaches [1, 2, 3, 4, 22, 23]. BPMN already provides the concepts to define the behaviour 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 Wireless Sensor Network (WSN) 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.

Indeed, with these approaches, business modellers can define both business processes and IoT behaviour 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.

In our work, we only use standard BPMN to define both central and IoT behaviour of business processes. We use the BPMN resource element to integrate IoT device information into the model, and we translate the BPMN that defines the IoT behaviour into Callas bytecode [15]. 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. In addition, Callas also supports remote IoT devices reprogramming, a feature that is the first step to support ad-hoc changes [21] in the parts of business processes that define IoT behaviour.

This paper is organised as follows. Section 2 presents the related work, while section 3 describes our proposal to model IoT behaviour within BPMN business processes and how we translate it to Callas source code. Finally, section 4 concludes the paper and hints for future work directions.

2. Related work

Business modellers define business processes with languages such as Web Services Business Process Execution Language (WS-BPEL) [18] or BPMN, which use an abstraction level closer to the domain being specified. At this level, modellers should not deal with IoT devices heterogeneity and specificities: IoT devices use different operating systems (e.g., TinyOS, Contiki [8]), different programming languages (e.g., nesC [10], Protothreads), and different communication protocols. Traditionally, web services are used to provide IoT information and functionalities, abstracting and encapsulating low-level details. More recent approaches take a step forward by supporting IoT behaviour definition within the business process [5, 16].

Zen, Guo and Cheng survey two approaches to implement IoT web services [24]. 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 [7, 11, 17]. In [7, 11], the authors extend WS-BPEL with context variables to monitor IoT information, abstracting the set of operations to interact with IoT devices. In [17], the authors define how to express IoT devices and their services in an IoT-aware process model.

GWELS [12] 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 with business processes.

The above approaches assume a centralised control of the process execution, i.e., the central system executes and coordinates processes and communicates with IoT devices using web services. However, business modellers cannot define the behaviour of WSNs, they can only use services whose behaviour is previously defined.

In a decentralised approach, IoT devices (sensors and actuators) can work together to execute parts of business processes, reducing the number of exchanged messages and promoting central process engine scalability, since information is processed locally. Another important advantage, present in many scenarios, is that the lower network traffic between the central system and IoT devices improves battery lifetime of IoT devices. To model business processes according to this decentralised approach, business modellers need a unified framework where they can also specify IoT devices behaviour and its interactions with the central system. BPMN already provides the concepts to define the behaviour of various participants by using different pools; their interactions are specified through collaboration diagrams.

Caracas and Bernauer [1, 2, 3] use standard BPMN to model both central and IoT behaviour. 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 behaviour 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. propose the makeSense framework [4]. 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 behaviour, makeSense uses a second meta model [4, 6, 23]. To generate executable IoT device code, they use 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 that 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 [23].

In our work, we only use standard BPMN to define all the business process and IoT device information is added to the model by using the BPMN resource element. We translate the BPMN that defines the IoT behaviour into Callas bytecode [15], a non-platform-specific language that IoT devices with an available Callas virtual machine can execute.

3. From BPMN modelling to IoT device execution

This section describes how we use the BPMN language to model IoT behaviour within a business process, at the same level of abstraction. After that, it presents how to translate the BPMN IoT behaviour to Callas code and how to deploy and execute it in IoT devices.

3.1. Using BPMN to model IoT behaviour

As BPMN-based approaches the related work presents, we use pool elements to distinguish and model both central and IoT devices behaviour. The interaction amongst participants (i.e., pools) is specified through collaboration diagrams.

Figure 1 presents a use case scenario. It is a simplified process for automatic irrigation control. The process includes two participants, whose behaviour we model in two separated pools. The central process (named Irrigation) receives a notification when the IoT Irrigation process detects a water leek that needs human intervention to be fixed. The IoT

Irrigation process, periodically, wakes up to irrigate the garden. The process starts by reading the rainfall sensor, and only starts irrigating if it is not raining; otherwise, it stops. Upon finishing the irrigation task, the process reads the flowmeter to make sure it is sealed. If water still flows, the IoT Irrigation process sends an alert to the central process. This simplified process omits a lot of details, such as the capability to change both timers (the irrigation interval and duration) of the IoT Irrigation process, for instance.

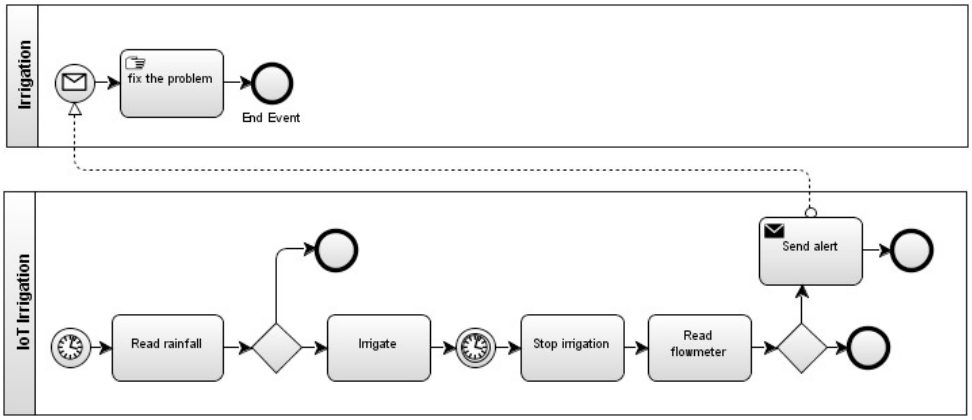

Figure 1 - BPMN use case scenario

To model IoT devices behaviour, we select a subset of BPMN, avoiding the use of two different meta models. The selection of the subset considers two main factors:

1. to model IoT devices behaviour, business modellers do not need all BPMN elements, as stated by Caracas [3]. This way, we include in our subset the BPMN elements that Caracas use to model the IoT programming patterns, and
2. as Callas already considers general IoT devices limitations, it does not support, for instance, parallel tasks, and it is a block-structured language, unlike BPMN that supports unstructured control flow, allowing gateways to loop back and forward. Figure 2 illustrates an example of a flow control that we do not support in IoT behaviour 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 (service 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.

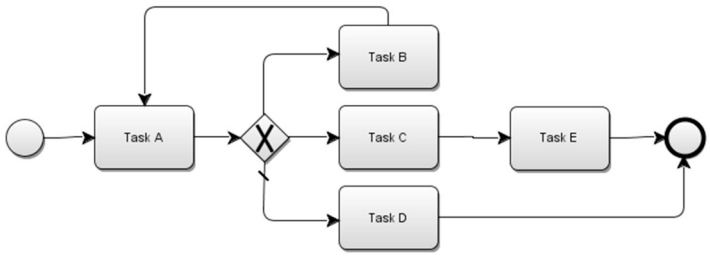

Figure 2 - Example of a non-block control flow structure

We use script tasks to handle the interaction with the hardware of IoT devices. For instance, in our use case example, IoT devices have rainfall sensors, and we need to model the data acquisition. For that, we use the script task

Read Rainfall that corresponds to an external invocation in Callas. They match the underlying IoT devices hardware and their definition includes enough information to derive its signature, i.e., the task name, its return type, and the type of its parameters.

To specify the IoT devices that will execute processes, we use the BPMN resource class, whose parameters are applied in queries at runtime for resource assignment. This way, we also avoid integrating information about IoT devices in a non-standard way into BPMN models. Figure 3 presents a simplified example based on our use case scenario, that can be reused in other business processes.

```
<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>irrigationDevice<formalExpression>
   </resourceParameterBinding>
</performer>
```

Figure 3 - Example of using the BPMN resource class to perform the binding with IoT devices

3.2. Translating BPMN to Callas code

Unlike the approaches of the related work that translate IoT devices behaviour to platform-specific code, we translate it into Callas bytecode [15]. By choosing the Callas sensor programming language, we take advantage of some of its characteristics and functionalities. 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, guaranties statically that well-typed programs are free from certain runtime errors (type-safety) and that its virtual machine is sound, that is, its semantics corresponds to that of the Callas language. It includes some domain-specific sensor 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.† Other interactions with IoT devices hardware are performed by calling language external functions. 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 remotely deploy executable code, that we use to install the code in IoT devices, remotely. We also consider this feature as the first step to support ad-hoc changes [21] in IoT business process parts.

Figure 4 presents the Callas source code that implements the IoT Irrigation pool behaviour described in Figure 3. The program starts by declaring two module types: Nil, an empty module type used to represent void function returns; and the Sensor module that defines the message signatures that flow on IoT devices. All sensors in the network implement this same interface. The Sensor module implementation spans from line 9 to line 20. Each function is implemented using the def construct. The startWSNIrrigation 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 invoking the irrigate external function. The system irrigates for 20 minutes, stops by calling the stopIrrigation

† Currently the CVM 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/.

function, and checks whether the water valve is sealed. It sends and alerts to the central process in case there is a water leak (send waterLeakAlertfunction).

The main program (lines 22–27) loads the module into the device's memory and installs a timer to call startWSNIrrigation every day.

```
defmodule Nil:                                                          1
  pass                                                                  2
                                                                        3
defmodule Sensor:                                                       4
  Nil startWSNIrrigation ()                                             5
  Nil waterLeakAlert ()                                                 6
                                                                        7
# declare module Sensor                                                 8
module m of Sensor:                                                     9
  def startWSNIrrigation (self):                                        10
    isRaining = extern readRainfall ()                                  11
    if not isRaining :                                                  12
      extern irrigate ()                                                13
      sleep (20∗60∗1000)                                                14
      extern stopIrrigation ()                                          15
      curFlowmeter = extern readFlowmeter ()                            16
      if curFlowmeter > 0                                               17
        send waterLeakAlert ()                                          18
  def waterLeakAlert (self):                                            19
    pass                                                                20
                                                                        21
mem = load                          # load the sensor memory            22
newMem = mem || m                   # update with Sampler module m      23
store newMem                        # replace the sensor memory         24
                                                                        25
# invoke tasks every day                                                26
startWSNIrrigation () every 24∗60∗60∗1000                               27
```

Figure 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; and (3) translation into Callas.

For the parsing part, we take the BPMN XML file and create 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 behaviour.

The second phase traverses the AST multiple times for validations. As for additional syntactic 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 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 server and the IoT

devices). Upon message reception, the Callas virtual machine takes the responsibility of invoking the correct message handler function; (b) timers are directly supported by Callas. For each timer in the model we define a correspondent timer in Callas that invokes a function encoding the timer behaviour; (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.

- 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. Function `readRainfall` is an example of a script tasks (*vide* Figure 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 gateway is ignored during the translation process, since its behaviour is captured by the Callas sequential composition statement. It is just used during 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 modellers in the BPMN design tool context.

Figure 4 contains the result from translating the IoT `Irrigation pool` modelled in Figure 2.

3.3. 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:

1. 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;
2. remove IoT pools from the BPMN model file, since this behaviour is going to be performed by IoT devices, instead of the jBPM server;
3. 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
4. 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 then can deliver messages using the jBPM RESTful API.

3.4. Prototype

In our prototype, we use the Eclipse IDE [9] and the jBPM [20], a BPMN server from JBoss. Our irrigation use case is composed of two 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.

4. 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 modellers to define IoT behaviour within business process and with the same level of abstraction, by only using standard BPMN. By translating BPMN, the IoT behaviour 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 provided by IoT devices. Modellers 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.

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.

References

1. Caracas A. From business process models to pervasive applications: Synchronization and optimization. IEEE International Conference on Pervasive Computing and Communications (PERCOM Workshops), IEEE, 2012; p. 320–325.
2. Caracas A, Bernauer A. Compiling business process models for sensor networks. International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011, p. 1-8.
3. Caracaş A. Modeling, compiling, and efficiently executing business processes on resource-constrained wireless sensor networks. Phd Thesis, Eth Zurich. 2012
4. Casati N, Daniel F, Dantchev G, Eriksson J, Finne N, Karnouskos S, Montero P, Mottola L, Oppermann F, Picco G, Quartulli A, Römer K, Spiess P, Tranquillini S. Towards Business Processes Orchestrating the Physical Enterprise with Wireless Sensor Networks. International Conference on Software Engeneering (ICSE), 2012.
5. Chang C, Srirama S N, Buyya R. Mobile cloud business process management system for the internet of things: a survey. *ACM Computing Surveys* (CSUR), 2016; 49(4), 1-42.
6. Daniel F, Eriksson J, Finne N, Fuchs H, Gaglione A, Karnouskos S, Montero PM, Mottola L, Oppermann FJ, Picco GP, Römer K, Spieß P, Tranquillini S, Voigt T. makeSense: Real-world Business Processes through Wireless Sensor Networks. CONET/UBICITEC, 2013, p.58-72.
7. Domingos D, Martins F, Cândido C. Internet of Things Aware WS-BPEL Business Process. ICEIS (2), 2013, pp. 505–512.
8. Dunkels A, Grönvall B, Voigt T. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I), 2004.
9. Eclipse IDE for Java EE Developers, 2014. URL: http://www.eclipse.org/. Page visited on May 10th, 2017.
10. Gay D, Levis P, von Behren R, Welsh M, Brewer E, Culler D. The nesC Language: A Holistic Approach to Network Embedded Systems. ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). ACM Press, 2003, pp. 1–11.
11. George AA, Ward PAS. An architecture for providing context in WS-BPEL processes. Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds (CASCON). ACM Press, 2008, pp. 22:289–22:303.
12. Glombitza N, Lipphardt M, Werner C, Fischer S. Using graphical process modeling for realizing SOA programming paradigms in sensor networks. Sixth International Conference on Wireless On-Demand Network Systems and Services (WONS). 2009, pp. 61 –70.
13. Haller S, Karnouskos S, Schroth C. The Internet of Things in an Enterprise Context. Future Internet Symposium, FIS 2008, 2008; pp. 14–28.
14. Jedermann R, Lang W. The benefits of embedded intelligence–tasks and applications for ubiquitous computing in logistics. In *The Internet of Things*; Springer, 2008; pp. 105–122.

15. Lopes, L, Martins F: A safe-by-design programming language for wireless sensor networks. *Journal of Systems Architecture - Embedded Systems Design*, 2016; Vol. 63, pp. 16-32
16. Mass J, Chang C, Srirama SN. 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.
17. Meyer S, Ruppen A, Magerkurth C. Internet of Things-Aware Process Modeling: Integrating IoT Devices as Business Process Resources. Advanced Information Systems Engineering. Springer, 2013, pp. 84–98.
18. OASIS. Web Services Business Process Execution Language Version 2.0. Technical report, Organization for the Advancement of Structured Information Standards, 2007.
19. OMG. Business Process Model and Notation (BPMN), Version 2.0. Technical report, Object Management Group, 2011.
20. Redhat. JBoss jBPM. URL: www.jboss.org/jbpm/. Page visited on May 10th, 2017.
21. Schonenberg H, Mans R, Russell N, Mulyar N, van der Aalst WM. Towards a Taxonomy of Process Flexibility. CAiSE Forum, 2008, Vol. 344, pp. 81–84.
22. Sungur CT, Spiess P, Oertel N, Kopp O. Extending BPMN for wireless sensor networks. Conference on Business Informatics (CBI). IEEE, 2013, pp. 109–116.
23. Tranquillini S, Spieß P, Daniel F, Karnouskos S, Casati F, Oertel N, Mottola L, Oppermann FJ, Picco GP, Römer K, Voigt T. Process-based design and integration of wireless sensor network applications. *In Business Process Management*; Springer, 2012; pp. 134–149.
24. Zeng D, Guo S, Cheng Z. The Web of Things: A Survey (Invited Paper). *Journal of Communications* 2011, 6.