Ad-hoc Changes in IoT-aware Business Processes

Dulce Domingos*, Francisco Martins*, Ricardo Martinho†, and Mário Silva*
*Lasige & Department of Informatics, Faculty of Sciences, University of Lisboa
Edifício C6, Piso 3, Campo Grande
1749-016 Lisboa, Portugal
E-mails:{dulce,fmartins,mjs}@di.fc.ul.pt
†School of Technology and Management, Polytechnic Institute of Leiria
Campus 2, Morro do Lena, Alto do Vieiro
2411-901 Leiria, Portugal
E-mail: rmartin@estg.ipleiria.pt

***Abstract*—The Internet of Things makes it possible to adapt the behaviour of business processes in response to real-time context updates. In addition, physical items can run and validate parts of the business processes and optimise their execution, while reducing message transmissions. State-of-the-art event-driven, service-oriented architecture approaches contribute to enabling inter-organisational collaboration and interoperability of heterogeneous hardware, but their applicability is limited to pre-planned, well-structured processes. We take a step forward by supporting ad-hoc changes within business processes, considering changes in the state of the Things; likewise, whenever needed, the software controlling the behaviour of sensors may be dynamically reconfigured as a result of changes in the functional specifications of business processes.**

***Index Terms*—IoT, business processes, ad-hoc changes, smart objects, reprogramming behaviour, web services**

I. Introduction

The Internet of Things (IoT) is the vision for bridging the gap between the physical world and its representation within the digital world. Radio-Frequency IDentification (RFID), as well as Near Field Communications (NFC), and Wireless Sensor and Actuator Networks (WSAN) are recognized as the atomic components that link both worlds. Notably, a new sort of sensors, named smart objects, come equipped not only with wireless communication, memory, and context awareness, but also with computation capabilities [1, 2].

The IoT will have an impact on the execution of business processes, profiting from the context information that the IoT provides to optimising their execution and the capability of reacting to emergent real-time situations [1]. Additionally, smart objects can also perform parts of the business process logic, whenever central control is not required. This helps reducing the amount of central processing and data exchanged between a central system and physical objects [3, 4]. Manufacturing, supply chain management, energy, health and automotive are some of the most promising application areas of the IoT [4].

To provide information and functionalities of physical objects to business processes, the Web Services technology offers a promising approach, since it facilitates interoperability and encapsulates the heterogeneity and specificities of physical objects [3, 5, 6].

The Business Process Execution Language for Web Services (WS-BPEL) has emerged as the de-facto standard for implementing processes based on (web) services [7].

Several proposals follow a synchronous request and response paradigm, using the IoT information as decision context in predefined points of business processes [8–14].

In what concerns event-driven approaches, they support more reactive business processes. They use languages based on event-condition-action (ECA) rules [15] or the Web Services Choreography Description Language (WS-CDL) [16, 17].

However, despite we can use these approaches to define how exceptional or unusual conditions should be handled, we still have to define them before execution. If everything is predetermined before execution, processes are not able to fully benefit from the IoT [3]. To react to real-time IoT information, we need to support ad-hoc changes to business processes. For instance, let us consider a process that needs to re-schedule its deliveries because it has been notified that the temperature inside a cargo container has reached some threshold. This cannot be achieved by terminating or cancelling the process, but by changing its behaviour, while it is still executing.

In this paper, we begin by identify main changing requirements in IoT-aware business processes, considering the application scenario of dangerous goods transportation. Then, we address fundamental issues for supporting ad-hoc changes in IoT-aware business processes. We provide mechanisms to change execution flows while ensuring their correctness, by defining change primitives and compliance criterion.

In addition, we also support changes in the behaviour of physical objects by reprogramming them dynamically and remotely, considering they execute parts of business processes. For that purpose, we introduce Callas, a sensor programming language for which we implemented System Development Kits (SDKs) for two Wireless Sensor Networks (WSN) platforms.

The remainder of this paper is structured as follows: in Section II, we start by presenting our application scenario. After defining some simplified WS-BPEL examples, we use them to identify the main changing requirements of IoT-aware business processes. Section III surveys related work. In Section IV, we present our approach to change sensors behaviour and in Section V we describe how we can perform ad-hoc changes

978-1-4244-7414-1/10/$26.00 ©2010 IEEE

in IoT-aware business processes in a correct way. Finally, we draw our conclusions and provide an overview of future work.

II. Dangerous Goods Transportation Scenario

The European Agreement concerning the International Carriage of Dangerous Goods by Road, commonly known as ADR,[1] governs transnational transport of hazardous materials. The ADR was put forward to ensure the safety of people (either directly involved, such as consigners or carriers, or indirectly involved, such as emergency staff in case of an emergency situation), properties, and the environment. This way, it aims at minimizing the risk of incidents and guarantying an effective response. In this section, we describe how sensors can help to comply with ADR requirements. After presenting some examples with the WS-BPEL language, we use them to identify the main challenges we have to overcome to support ad-hoc changes in IoT-aware business process.

A. Using WSN in Dangerous Goods Transportation

In the following, we present some applications of WSN that may help to comply with ADR requirements:

- *Tracking and tracing vehicles*: these systems aim at monitoring vehicle routes with real-time knowledge of their position. The vehicle sensors are configured with the routes and only have to inform the central system about deviations. Periodically, a sensor obtains the vehicle position with a satellite location system, or cellular network systems triangulation, and sends it to a back-end system for central processing. The MITRA project [18] researched the use of these systems to monitor dangerous goods transportation, while Dias and Fontana [19] use them to guarantee cargo security.
  These systems can reduce the amount of exchanged data and central processing, whenever the vehicle routes is known in advance. However, to deal with unexpected events, such as changes in traffic conditions, they need to reconfigure the initial route, while vehicles are on the road.
- *Monitoring the storage of dangerous goods*: as an example, the EU research Project CoBIs [20] already uses a sensor network to monitor the storage of chemical containers. It ensures the following rules: (1) only a certain amount of chemical may share one place, (2) storage of specific combinations of chemicals is prohibited, (3) some substances must be kept in a special storage area. When rules are broken, sensors send alerts to a back-end system, and raise visual alerts for local notification. However, current services nodes only provide support for changing the amount stated by rule (1).
- *Monitoring transport conditions*: these systems use sensors to monitor containers conditions, such as, temperature, pressure, tampering, or door opening. For example, in [3], the authors use sensors to monitor fruit quality changes during transport. Sensors pre-process their data and submit only substantial changes to the freight owner. They dynamically load software for the transport instructions, by using the JADE framework.

[1]From the French abbreviation *Accord européen relatif au transport international des marchandises dangereuses par route.*

```
<process name="dangerousGoodsTransportation" ...>
  <flow ...>
    <while ...>
      <sequence ...>
        <invoke name="getTemperatureInvoke"
          partnerLink="sensorTemperaturePartnerServiceLink"
          operation="getTemperature"
          outputVariable="Temperature"/>
          <if ...>
            <condition>
              $Temperature &gt; 12
            </condition>
              <sequence ...>
                <invoke name="notifyCentralMonitor" .../>
                <invoke name="notifyVehicle" .../>
                 ...
              </sequence>
          </if>
          <wait>
            <for>'P15M'</for>
          </wait>
        </sequence>
      </while>

      ... <!-- main execution flow of the process -->

  </flow>
</process>
```

Fig. 1. A WS-BPEL sketch to centrally monitor the temperature of a cargo container

All the above examples emphasise the benefits of using sensors to provide real-time information and execute parts of the business logic. In addition, they also justify the need for remotely changing sensor logic in real-time.

We now illustrate, in WS-BPEL, how sensor information and functionality can be integrated into business processes.

B. BPEL Examples

Web services present two main interaction patterns types:

- *request-response*: clients send a request and receive a response synchronously.
- *subscribe-notification*: clients register to a web service to receive event notifications.

These interaction patterns are used to get sensors information synchronously or asynchronously. Both implicitly cause the execution of more or less complex programs in sensors. For instance, we may query the current temperature or the average temperature of the sensor network. Alternatively, we subscribe an event that will notify us when the temperature reaches some threshold.

Figure 1 presents a sketch of a WS-BPEL process that centrally monitors the temperature of a cargo container.

A WS-BPEL process definition includes partner links that define the relationships with other business partners, declarations of process data, handlers for various purposes, and the activities. Basic activities perform their intended purpose, such as receiving a message from a partner, or manipulating

```
<process name="dangerousGoodsTransportation" ...>
 <flow ...>
  <invoke name="notifyTemperatureInvoke"
    partnerLink="sensorTemperaturePartnerServiceLink"
    operation="notifyTemperature"
    inputVariable="ThresholdValue" />

  <eventHandlers>
    <onEvent partnerLink="handleTemperaturePartnerServiceLink"
      operation="handleTemperature" ...>
      <scope>
          <sequence ...>
             <invoke name="notifyCentralMonitor" .../>
             <invoke name="notifyVehicle" .../>
              ...
          </sequence>
      </scope>
    </onEvent>
  </eventHandlers>

    ... <!-- main execution flow of the process -->

 </flow>
</process>
```

Fig. 2. A WS-BPEL example to monitor temperature on the sensor

data. Structured activities contain other activities and define the business logic control flow.

In this example, we use the basic activity invoke to call the web service getTemperature. The structured activities we use are:

- flow: specifies the parallel execution of the monitorization and the main execution flow of the process;
- while: specifies the loop to invoke the getTemperature every fifteen minutes;
- sequence: specifies the sequential executions of activities when the temperature reaches the value 12; and
- if−else: evaluates whether the temperature reaches the value 12.

In Figure 2, we illustrate an alternative way to monitor the temperature of the container. Here we transfer the logic of testing the value of the temperature to the sensor. This way, the business process only performs two main operations:

- *invoke* notifyTemperature *to subscribe the service*: this web service receives an argument with the temperature threshold value; and
- *handle the event reception with an* onEvent *element within an event handler*: this way, the process can receive events in parallel to its "normal" flow. The message event represents the web service method handleTemperature exposed by the process. When the process receives the event, *i.e.*, when the method handleTemperature is invoked, it executes the sequence activity inside the scope of the onEvent handler.

C. Changing Requirements

Considering sensors behaviour, we identify two types of changes:

- *changing configuration values*: for instance, the threshold of notifyTemperature; and
- *changing software code*: for instance, reprogramming the way sensors monitor the temperature to compare it with a minimum threshold as well.

We can change configuration values by re-subscribing notifyTemperature with the new threshold value, or by providing a web service to perform this functionality. However, performing the second type of change is more complex, because we need to generate the code to reprogram sensors, and install it dynamically, taking into account the limitations and the heterogeneity of sensors.

In addition, we may need to change the execution flow of business processes. We distinguish the following types of change:

- *changing or reordering activities*: for example, a delivery process may need to re-schedule its cargo after being notified that the temperature inside the container raised above a pre-defined threshold; this cannot be achieved by terminating or cancelling the process. Instead, we change its behaviour: we can reorder the execution of activities or we can perform more substantial changes, such as replacing some activities;
- *changing subscriptions*: we may need to subscribe or unsubscribe some events.

While performing ad-hoc changes, we need to ensure the correctness of process executions, considering the state of the execution flow. Indeed, we cannot add a new activity in a sequence of activities, before one already executed.

Finally, to meet these requirements within the context of IoT-aware business processes, we must take into account that: (i) business processes executions are distributed among physical objects, (ii) the parts of business logic that physical objects execute are not controlled directly by the central systems, since they are encapsulated as web services, and (iii) we have to minimize information exchanges due to physical objects limitations.

III. Related Work

Web Services technology offers a promising approach to provide IoT information and functionalities to business processes [3]. However, limitations of physical objects in terms of computation power, battery supply, and communication range have to be overcome. Priyantha et al. propose solutions to deal with the overhead of the structured data formats used in web services [21], and Leguay et al. propose a simpler protocol and software architecture that reproduces the architectural concepts and information exchanges of a regular SOA implementation [5].

To distribute parts of the business logic among physical objects, some proposals use software agents. Despite the possibility of offering agent functionalities as web services and the advantages of encapsulating hardware heterogeneity and remote reprogramming of physical objects, agent frameworks present limitations. The mobile software agents transmission overhead and the runtime environment overhead, commonly based on Java, contribute to lower performance [3].

To use IoT information within business processes, many proposals follow a synchronous request and response paradigm, where IoT information is used in predefined points of business processes. They use the information that the IoT provides to: (i) define state transition constraints [8–10], (ii) determine the services that compose processes [11], (iii) choose between multiple implementations for a specific service [12], or (iv) determine whether a service should participate in future compositions [13, 14]. In particular, the language Context4BPEL proposes an extension to WS-BPEL to support context-aware process definitions [10].

Event-driven approaches follow two main directions: (i) they define languages based on ECA rules and use these rules to generate the WS-BPEL composition of web services automatically and dynamically according to events [15], or (ii) they use the WS-CDL language to define business processes and generate the compliant WS-BPEL compositions to execute [17].

However, to take advantage of IoT information, business processes need to react to new situations in real time, changing their execution flow [3]. Traditionally, systems that support ad-hoc changes address the problem of assuring the correctness of process executions, despite run-time changes. This is achieved by providing a set of operations with a constrained use to guarantee that the process definition continues syntactically correct and that the process execution is compliant with its definition [22, 23].

Reichert and Rinderle apply to WS-BPEL existing approaches supporting adaptive business processes, focusing on fundamental issues and exclude more advanced WS-BPEL process patterns [22].

More recent works address ad-hoc changes in distributed workflow management systems, where they have to coordinate each part of an execution flow that may span many servers. In the ADEPT project, the authors propose methods for modifying the process execution flow and transferring it to respective servers, considering the state of each part of the execution flow [24].

However, none of these approaches considers that we may not directly control some parts of processes, since they are distributed among physical objects and encapsulated as web services.

IV. Changing Sensors Behaviour

This section introduces Callas [25], a programming language for sensor networks that we adopted for modelling IoT-aware business processes. Callas has two main properties that justify our choice: it is self-reconfigurable (*i.e.*, it may be reprogrammed dynamically without physical intervention on the sensors), and it is type-safe. We then present a Callas example illustrating its support for reprogramming sensors using web services.

A. The Callas Programming Language

Callas offers constructs to define sensor computations, communications, code mobility, and code updates. The language is based on a process calculus that establishes the foundations for developing programming languages and run-time systems for sensor networks.

```
defmodule Sampler:
  Nil getTemp()
  Nil notifyTemp()
  double threshold()

# declare module sampler
module s of Sampler:
  def getTemp(self):
    curTemp = extern senseTemp()
    send log(curTemp)

  def threshold(self):
    15.0

  def notifyTemp(self):
    curTemp = extern senseTemp()
    curThreshold = threshold()
    if curTemp >= curThreshold:
      send alert(curTemp)

mem = load          # load the sensor memory
newMem = mem || s   # update function sample
store newMem        # replace the sensor memory
# invoke sample() every ten minutes, for one week
notifyTemp() every 10*60*24*7
```

Fig. 3. A program for querying/notifying a temperature in a WSN.

Sensor network applications are built by plugging together components called modules. Dynamic reprogramming is supported by modules that can be exchanged between sensors and then installed locally upon reception on a sensor.

The programming language, along with its run-time system, presents two fundamental static properties: the type-safety of the language, and the soundness of its run-time system. These properties ensure the absence of an important class of run-time errors in sensor network applications, shortening both development time and debugging time. The type-safety property ensures that well-typed programs will not raise run-time protocol errors. A compiler for a type-safe programming language statically type-checks code and prematurely detects run-time protocol errors in the usage of modules, before the application is deployed over the network. The soundness property ensures that the underlying run-time system preserves the semantics of the programming language. This is achieved by implementing the run-time system based on an abstract specification (*e.g.*, a virtual machine).

We have built two Software Development Kits (SDKs) for programming Wireless Sensor Networks with Callas in two different platforms, one for real-life Sun SPOT devices [26], and another for the VisualSense simulator [27]. Since we abstract the run-time system (*e.g.*, sensors hardware) with the virtual machine, the same compiler can be used for both. Each SDK, however, must provide its implementation of the virtual machine.

B. Programming with Callas

The example listed in Figure 3 programs a sensor node that responds to temperature queries (getTemp) and notifies a central node when the temperature raises above a threshold (notifyTemp).

A Callas program is a sequence of type and module declarations, assignments, expressions, and conditionals. It uses Python's line-oriented syntax, where indentation (the number of spaces in the beginning of a line) demarcates syntactic blocks.

The program starts with a type declaration (lines 1–4). It begins with the reserved word **defmodule**, followed the type identifier Sampler, and defines a module with three functions: getTemp, notifyTemp, and threshold.

Lines 7–19 declare a module bound to variable s of type Sampler just introduced. The declaration proceeds by defining the module's functions. Each function defines a signature and a body. For instance, the definition of function getTemp (lines 8–10) starts with the reserved word **def**, succeeded by the name of the function getTemp, and by a comma-separated list of parameters in parenthesis. The first parameter in any function is the module itself, *e.g.,* to allow for recursive calls.

The function body is a sequence of terms. Functions are second-class values, meaning that they cannot be handled as a value, *e.g.*, passed to a module. Notice that, as in Python, when a line ends with a colon the remaining lines form a syntactic group with an increased indentation. The body of function getTemp consists of two terms: the first assigns to variable curTemp the result from an external call to get the ambient temperature from the device; the second term is a network call to function log, passing the value of variable curTemp as an argument. Expression **send** log(curTemp) is an asynchronous call to function log in the neighbouring nodes. This expression yields as a result an empty module (that is the outcome of function getTemp). The definition of functions threshold and notifyTemp should be straightforward to follow.

The memory content of a sensor may be replaced dynamically throughout the lifetime of the device. For accessing the memory of a device we use expressions **load** and **store**. Lines 21–23 load the code of the device and save it into variable mem (line 21), assign to variable newMem a new module, by composing the modules in variables mem and s (line 22), and store the new module in memory (line 23).

Expression nem || s merges the functions of both modules nem and s into a new module, giving precedence to the functions of module s in case of name clashes (*i.e.*, the same function name appearing on both modules).

We program a timed-event in line 25 that invokes function notifyTemp every ten minutes for a week. This function triggers an alert event if the temperature raises the threshold.

C. Reprogramming Sensors via Web Services

The Callas language, as well as its run-time environment, allows for dynamic reprogramming of modules, meaning that while the sensor application is deployed, we can change its code without recovering the physical devices and reprogram them one by one.

This feature allows us to adequate the sensor's behaviour according to the changes occurred in the business process.

To replace sensor code remotely we send it as a module and install it in the sensor, upon reception. For instance, if we want the change the temperature threshold value, say to 18 degrees, we simply deploy a new threshold function. The code for the base station is the following:

```
module s1 of Threshold:
  def threshold(self):
    18.0

send deploy(s1)
```

assuming that Threshold is the type of a module with the threshold function. The code for the deploy function running in the sensor receives the module and updates its memory. We sketch it as part of a system module of the sensor:

```
module system of System:
  ...
  def deploy(this, module):
    curMem = load
    updatedMem = curMem || module
    store updatedMem
  ...
```

More interesting is to change the algorithm of the notifyTemp function. Let us change it to send an alert if the temperature is not in the range threshold−5 and threshold. The new code is sketched below:

```
module s2 of NotifyTemp:
  def notifyTemp(self):
    curTemp = extern senseTemp()
    curThreshold = threshold()
    if curTemp < curThreshold−5 or curTemp > curThreshold:
      send alert(curTemp)

send deploy(s2)
```

The interface with sensors, either for querying data or for code update, can be triggered from various sources such as a web service. In this case, we define a web service proxy to translate web service invocations to Callas function invocations.

V. Ad-hoc Changes to Business Processes

To support ad-hoc changes we need to guarantee the correctness of process executions. A process execution is correct if its definition is syntactically correct, and if it is compliant with its definition, *i.e.*, if it has been executed according to its definition up to the current moment [23].

We find two approaches for supporting ad-hoc changes, one that performs ad-hoc changes in two steps and another in only one step, providing in the last case a set of change operations with pre-conditions to enforce compliance.

When performing ad-hoc changes in two steps, users first change the process definition and then migrate the process execution to this new definition. To support the change of process definitions, systems provide a complete and minimal set of primitives that allows changing the various aspects of the process definition, such as inserting and deleting activities. Completeness refers to the provision of a set of operations that do not restrict the user to specify the required changes. This completeness should be met with a minimal set of primitives. Typically, these primitives guarantee that the new version of the definition is syntactically correct [23].

Before migrating a process execution to the changed process definition, systems have to check its compliance in order to

guarantee that the execution is still valid under the changed definition, preventing run-time errors. A simple but inefficient approach for assessing compliance is checking the process execution history for compatibility. More efficient approaches define migration conditions for each of the change operations, and determine whether migration is possible by considering the migration conditions of all the change operations used to derive the new process definition [22].

In the following, we identify the complete and minimal set of primitives to support ad-hoc changes of IoT-aware business processes, and the compliance criterion that accounts for these process requirements.

A. Change Primitives

We focus our work in the parts of business processes that interact with sensors. We support all the changes we need to perform with two primitives: remove and add a basic activity. With these two primitives, we support the change operations as follows:

- *Remove and add a synchronous request to a web service*: these two operations are the same as the remove and add of a basic activity;
- *Remove and add a subscription*: same as above. We point out that we can change the argument value of a subscription with a composite operation: we first remove the current subscription, and then we add a new one with the new value;
- *Remove and add a sensor functionality*: as we describe in Section IV, we support these operations with web services.

B. Compliance Criterion

An execution flow is compliant with a changed process definition D if it could have been executed according to D as well, and would consequently have produced the same effects on the flow state and variables. This is always the case when the execution flow is still in a stage where changes do not conflict, *i.e.*, the execution flow has not advanced beyond the steps where the old and the changed definitions differ. Otherwise, we need to check the history of the execution flow. In that case, the execution flow is compliant with process definition D if its history describes a valid flow of process definition D.

Reichert and Rinderle already relax this criterion to avoid excluding execution flows from being migrated when some already executed loop iteration is not compliant with the changed process definition. They (logically) discard from the execution history those history entries produced within previous loop iterations [22].

In the following, we explain why we also need a less restrictive correctness criterion for IoT-aware business processes. We consider each primitive operation individually:

- *Remove and add a synchronous request to a web service*: this relaxed correctness criterion is still too restrictive if we consider executions flows that derive from an event reception. Similarly to loop iterations, let us consider a flow instance of the example illustrated in Figure 2, which already has received the event once and has executed the corresponding steps to handle it. Suppose that it receives this event for the second time and needs to change the set of steps it uses to handle the event, for instance, by adding a new activity. This change would not be allowed, because the execution history of the first event handling is not compliant with the changed process definition;
- *Remove and add a subscription*: to remove an event subscription, we have to consider two different cases. If the process has not received a notification of the event, the change may be applied straightforwardly. Indeed, without this subscription the execution flow history remains compliant. Otherwise, taking into account the above criterion, this change would not be allowed, considering the history entries of the flow executed to handle the received event. The add operation of an event subscription raises another problem: take the example illustrated in Figure 1. If we want to add a subscription to receive a notification when the temperature gets lower below a pre-defined threshold, we do not have information on whether this event could have already occurred, thus we can not determine if the execution flow would have been the same with this subscription. Indeed, we do not save history information about values that sensors read and that not useful to processes, simply because sensors do not send them to processes;
- *Remove and add a sensor functionality*: the problem with the add operation of an event subscription is similar to the one we observe with the primitives to remove and add sensor functionalities. Indeed, with this kind of change, we can not determine whether an execution flow would produce the same effects on flow state and variables.

Taking into account the previous considerations, we extend the correctness criterion defined by Reichert and Rinderle by (logically) discarding from the execution history not only those history entries produced within previous loop iterations, but also those entries produced by previous event receptions. In addition, we also consider that the execution history does not include history information about sensors behaviour that has not been explicitly asked. This way, we only need migration conditions for the primitives to remove and add a synchronous request to a web service, which remains the ones that Reichert and Rinderle already defined for the primitives of removing and adding activities.

VI. Conclusions and Future Work

Business processes can greatly benefit from the IoT, since physical objects can run software providing real-time information for detecting unanticipated situations, which can trigger immediate reactions including the adoption of new radically different process. The execution of these new processes can involve on-the-fly reprogramming of sensors, which we support through a wireless sensor network framework running programs in Callas. Callas is a sensor-oriented programming language, self-reconfigurable and type-safe.

Our work introduces mechanisms for performing ad-hoc changes in IoT-aware business processes in a correct way. We identified the change primitives that support the change operations needed for modifying the parts of the business process that interact with physical objects. In addition, we defined a compliance criterion that does not exclude unnecessary changes from being performed.

In the near future, we intend to extend our approach to deal with evolutionary changes, *i.e.*, how we can apply changes to a group of run-time execution processes.

To validate our model for creating and executing IoT-aware business processes, we will create a testbed for simulating a realistic sensor-rich environment for tracing and tracking dangerous goods transportations and will measure users and systems responses to unforeseen events in this environment.

ACKNOWLEDGMENT

This work was supported by FCT through project PATI (PTDC/EIA-EIA/103751/2008) and through LASIGE Multi-annual Funding Programme.

REFERENCES

[1] SAP, "Toward a european strategy for the future internet," 2008, SAP White paper.

[2] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. In Press, Corrected Proof, available online since June 2010, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/B6VRG-506RH6G-1/2/9a5e0f620af63be61c934390e9e5d2f1

[3] R. Jedermann and W. Lang, "The benefits of embedded intelligence - tasks and applications for ubiquitous computing in logistics," in *Proceedings of The Internet of Things. First International Conference (IOT 2008)*, ser. Lecture Notes in Computer Science, C. Floerkemeier and et Al., Eds., no. 4952. Springer-Verlag, 2008, pp. 105–122.

[4] S. Haller, S. Karnouskos, and C. Schroth, "The internet of things in an enterprise context," in *Future Internet — FIS 2008: First Future Internet Symposium, FIS 2008 Vienna, Austria, September 29-30, 2008 Revised Selected Papers*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 14–28.

[5] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, "Service oriented architecture for heterogeneous and dynamic sensor networks," in *Proceedings of the Second international Conference on Distributed Event-Based Systems (DEBS'08)*, vol. 332. ACM Press, 2008, pp. 309–312.

[6] M. Aiello and S. Dustdar, "Are our homes ready for services? a domotic infrastructure based on the web service stack," *Pervasive Mobile Computing*, vol. 4, pp. 506–525, 2008.

[7] OASIS, "Web services business process execution language version 2.0," Organization for the Advancement of Structured Information Standards, Tech. Rep., 2007.

[8] J. Han, Y. Cho, E. Kim, and J. Choi, "A ubiquitous workflow service framework," in *Proceedings of the 2006 International Conference on Computational Science and its Application (ICCSA 2006)*, ser. Lecture Notes in Computer Science, no. 3983. Springer-Verlag, 2006, pp. 30–39.

[9] K. Shin, Y. Cho, J. Choi, and C. Yoo, "A workflow language for context-aware services," in *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering*. IEEE Press, 2007, pp. 1227–1232.

[10] F. Tang, M. Guo, M. Dong, M. Li, and H. Guan, "Towards context-aware workflow management for ubiquitous computing," in *Proceedings of the 2008 international Conference on Embedded Software and Systems (ICESS 2008)*. IEEE Press, 2008, pp. 221–228.

[11] M. Vukovic and P. Robinson, "Adaptive, planning based, web service composition for context awareness," in *Proceedings of the 2nd International Conference on Pervasive Computing (ICPC'2004)*, 2004, pp. 247–252.

[12] A. Ranganathan and S. McFaddin, "Using workflows to coordinate web services in pervasive computing environments," in *Proceedings of the IEEE International Conference on Web Services (ICWS 2004)*. IEEE Press, 2004, pp. 288–295.

[13] D. Karastoyanova, A. Houspanossian, M. Cilia, F. Leymann, and A. Buchmann, "Extending BPEL for run time adaptability," in *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference (EDOC '05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 15–26.

[14] L. Baresi and S. Guinea, "Towards dynamic monitoring of WS-BPEL processes," in *Proceedings of International Conference On Service Oriented Computing (ISCOC 2005)*, ser. Lecture Notes in Computer Science, vol. 3826. Springer-Verlag, 2005, pp. 269–282.

[15] Z. Laliwala, R. Khosla, P. Majumdar, and S. Chaudhary, "Semantic and rules based event-driven dynamic web services composition for automation of business processes," in *Proceedings of the IEEE Services Computing Workshops (SCW 2006)*. IEEE Press, 2006, pp. 175–182.

[16] W3C, "Web Services Choreography Description Language (WS-CDL) Version 1.0," W3C, Tech. Rep., 2005.

[17] J. Mendling and M. Hafner, "From inter-organizational workflows to process execution: Generating BPEL from WS-CDL," in *Proceedings of the On the Move to Meaningful Internet Systems: OTM Workshops (OTM 2005)*, ser. Lecture Notes in Computer Science, R. Meersman, Z. Tari, and P. Herrero, Eds., no. 3762. Springer-Verlag, 2005, pp. 506–515.

[18] E. Planas, E. Pastor, F. Presutto, and J. Tixier, "Results of the MITRA project: monitoring and intervention for the transportation of dangerous goods," *Journal of Hazardous Materials*, vol. 152, no. 2, pp. 516–526, Apr. 2008.

[19] E. M. Dias and C. F. Fontana, "Methodology for the implementation of the safe logistic chain process," in *Proceedings of the 9th WSEAS International Conference on Applications of Computer Engineering*, 2010, pp. 28–34.

[20] P. Spiess, H. Vogt, and H. Juetting, "Integrating sensor networks with business processes," in *Proceedings of the 4th ACM International Conference on Mobile Systems, Applications, and Services (ACM MobiSys 2006), Real-World Sensor Networks Workshop*, 2006.

[21] N. Priyantha, A. Kansal, M. Goraczko, and F. Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys'08)*. ACM Press, 2008, pp. 253–266.

[22] M. Reichert and S. Rinderle, "On design principles for realizing adaptive service flows with BPEL," in *Proceedings of the International Conference on Conceptual Modeling (EMISA 2006)*, ser. Lectures Notes in Informatics. Springer-Verlag, 2006, pp. 133–146.

[23] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Workflow evolution," *Data Knowledge Engineering*, vol. 24, no. 3, pp. 211–238, 1998.

[24] M. Reichert and T. Bauer, "Supporting ad-hoc changes in distributed workflow management systems," in *Proceedings of the Confederated International Conference On the Move to Meaningful Internet Systems (OTM'2007)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 150–168.

[25] F. Martins, L. Lopes, and J. Barros, "Towards safe programming of wireless sensor networks," in *Proceedings of Programming Language Approaches to Concurrency and Communication-cEntric Software (PLACES'09)*, ser. EPTCS, vol. 17, 2010, pp. 49–62.

[26] Sun Microsystems Laboratories, "Project sun SPOT," `http://www.sunspotworld.com/`.

[27] P. Baldwin, S. Kohli, E. A. Lee, X. Liu, and Y. Zhao, "Modelling of Sensor Nets in Ptolemy II," in *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN'04)*. ACM Press, 2004, pp. 359–368.