Internet of Things Aware WS-BPEL Business Process

Dulce Domingos, Francisco Martins and Carlos Cândido
LaSIGE - Faculty of Science, Lisbon University, Campo Grande, Lisbon, Portugal

Keywords: Internet of Things, Business Process, WS-BPEL.

Abstract: Business process can benefit from the deployment of the Internet of Things, since they can use sensor context information to optimize their execution and to react to emergent situations in real-time. Nowadays, the WS-BPEL is the de-facto standard language used to define processes through the composition of web services. However, as this language is based on the service orchestration paradigm, it complicates the way process can use context information. This paper presents a WS-BPEL extension that includes context variables. These variables maintain sensor values. Their updates are done automatically by using the publish/subscribe paradigm. In addition, to support more reactive behaviours, this extension also includes the when-then construct. To realize this extension, we use the language transformation mechanisms, so it can be as much independent as possible from the process execution engine.

1 INTRODUCTION

The Internet of Things (IoT) aims at bridging the gap between the physical world and its representation within information systems. It will definitely have a deep impact on business processes, since they can use the context information IoT provides to optimize their execution and to adapt to environment changes, in real time.

The WS-BPEL (OASIS, 2007) is a OASIS standard used to define processes as web services compositions, i.e., orchestrations of web services. To facilitate the IoT integration into business processes, recent works provide the information and functionalities of physical objects as web services. These web services can be implemented directly in sensors or through middleware (Zeng et al., 2011). The service-oriented approach has the advantage of facilitating interoperability as well as it also encapsulates heterogeneity and specificities of physical objects. As the WS-BPEL is based on service orchestration, the sensor information that web services provide can be easily integrated into processes by using a synchronous request/reply paradigm.

However, with this paradigm, if a process needs to have updated information about environment changes, it has to get sensors information periodically, overloading the process execution engine and increasing the number of exchanged messages between the sensor network and the execution engine.

The work we present in this paper aims at simplifying the use of updated sensor context information in business processes and at defining more reactive processes. We define a WS-BPEL extension to include context variables, which are update asynchronously through the publisher/subscriber paradigm, according to the WS-Notification standard. We support context variable with a language transformation. This option has the advantage of being independent from the process execution engine.

In addition, our WS-BPEL extension also include the *when-then* construct. The *when* condition can include context variables. We also support this construct with a language transformation. However, our implementation depends on an auxiliary service that detects the modification of variable values. In our prototype, we use Apache ODE *listeners* (Apache ODE, 2013).

We present a motivating scenario in the next section. Section 3 resumes the standards we use in this work. We describe related work in section 4 and our WS-BPEL extension in section 5. The last section includes conclusions and future work.

2 MOTIVATING SCENARIO

Our motivating scenario is based on a common business application: the transportation of goods.

A distribution company receives a container of

Domingos D., Martins F. and Cândido C..
Internet of Things Aware WS-BPEL Business Process.
DOI: 10.5220/0004449905050512
In *Proceedings of the 15th International Conference on Enterprise Information Systems* (ICEIS-2013), pages 505-512
ISBN: 978-989-8565-60-0
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

strawberries on a pier discharge. After unloading, the container is placed on a truck and transported to the distribution center, from where they are distributed to retail stores. The container has temperature sensors and they are used to monitor the strawberries state. During transportation, the temperature inside the container raises above the value that guarantees strawberries quality. If the company maintains the destination of strawberries, they will reach consumers in bad conditions. To avoid this waste, when they get the information about the raise of temperature, they change the container destination and it is forwarded to a closer distribution center.

In this use case scenario, we illustrate the benefits of monitoring context information, i.e., the temperature of the container, and consequently, the possibility to change the process to react to context modifications.

To include this behaviour into a WS-BPEL process definition, the modeler has to define the interaction with sensors, deviating his focus from the main process logic.

3 CONCEPTS

In this section we present an overview of WS-BPEL. Before that, we present the other standards we use in this work.

3.1 Web Services Addressing

Web Services Addressing (WS-Addressing) (Box and et al., 2004) defines transport-neutral mechanisms that allow web services to communicate addressing information. This specification defines elements to uniquely identify web services endpoints using Extensible Markup Language (XML). One of the elements is the *EndPointReference* (EPR), whose field *address* represents the web service address. Beyond this field, an EPR offers optional fields (such as the *ReferenceParameters*), for instance, to distinguish EPRs with the same *address*.

3.2 Web Services Description Language

Web Service Description Language (WSDL) (Christensen et al., 2001) is a XML-based language used to describe the functionalities that web services offer. A WSDL web service description includes its name, address (EPR), the service binding (defines the protocols), operations available, exchanged messages, and possible faults. It contains all the information needed to contact a web service.

3.3 Web Services Notification

Web Services Notification (WSN) (OASIS, 2006) defines a set of specifications that aim at defining how web services interact using notifications or events. The communication uses the publisher/subscriber paradigm, where an entity can publish information to others without having to know them in advance. The specifications provide WSDL interfaces. We point out the interfaces for two services: the publisher service (*NotificationProducer*) and the service that receives notifications (*NotificationConsumer*). They specify the minimum operations that each service must provide: the publisher service has to provide the *subscribe* and *get-CurrentMessage* operations, and the consumer service has to provide the operation to receive notifications, the *notify* operation. When invoking the *subscribe* operation, the subscriber must define, among other things, the EPR to where the notifications should be sent.

3.4 Web Services Business Process Execution Language

Web Services Business Process Execution Language (WS-BPEL) (OASIS, 2007) is the OASIS standard executable language for defining business processes through web services orchestration. A business process definition includes two elements: a WSDL file that describes the business process functionalities (web services) with their messages data structures, services addresses, among others, and a WS-BPEL file that defines the business process logic.

The WS-BPEL includes different types of activities, such as flow control activities (*If, While, Scope, Flow*), communication activities (*Receive, Reply, Invoke*), assign values activities (*Assign*), fault handlers (*Throw, Rethrow*), to name a few. We can declare variables of any primitive type, complex type (consisting of several primitive data types), and messages. Message variables are used almost exclusively in communication activities. Variables can be global or local, if declared within a *Scope*.

Processes in WS-BPEL export and import functionalities by using web services. Web services are modeled as *partnerLinks*. Every *partnerLink* is characterized by a *partnerLinkType*, which is defined in the WSDL definition. A *partnerLinkType* specifies the role and the type of a partner. An input communication activity is associated with the *MyRole* and an output communication activity is associated with the *PartnerRole*.

In order to distinguish process instances, WS-BPEL provides the *Correlation* mechanism. A *corre-*

lationSet is defined by 1) the primitive data type that will be used and 2) the rule set (one per message type). The *correlationSet* is associated with communication activities. Each *correlationSet* can only be initialized once and, if we use it in an *Invoke*, we have to define when the *Correlation* is established: in the sending operation, in response, or in both. The *CorrelationSets* property defines, through XPATH, the message elements exchanged by processes that identifies each conversation (i.e., each process instance).

The WS-BPEL standard supports extensibility, by allowing namespace-qualified attributes to appear in any WS-BPEL element and by allowing elements from other namespaces to appear within WS-BPEL defined elements. In addition, WS-BPEL provides two explicit extension constructs: *extensionAssignOperation* and *extensionActivity*.

All extensions used in a process must be declared. This statement is made by inserting into the *Extensions* construct language the namespaces associated with the extensions and the *MustUnderstand* attribute with the value *yes* or *no*, which states whether the process execution engine has to support the extension.

There are two different options to realize an extension (Kopp et al., 2011). With a ”BPEL Language Transformation”, extension constructs are translated into standard BPEL constructs. The generated standard BPEL code can be deployed on a process execution engine that ignores the extension. With the other option, the ”BPEL runtime engine”, the extension is realized by changing the process execution engine in order to support the additional functionalities.

3.5 XSL Transformations

Extensible Stylesheet Language Transformations (XSLT) (Clark et al., 2007) is a specification that defines the syntax and semantics of a language to transform and render XML documents. XSLT is designed for use as part of the Extensible Stylesheet Language (XSL), which is a style language for XML. XSL includes an XML vocabulary to specify formatting and uses XSLT to describe the document transformation.

4 RELATED WORK

In our work, we use context with the same meaning as George et al. (George and Ward, 2008; George, 2008). These authors define context as an environment state, which is external to the process, whose value can change independently of the process lifecycle, and can influence process execution.

Traditionally, context information is obtained according to a synchronous request/response paradigm, and business processes use it in predefined points. They use context information to: (i) determine the services that compose processes (Yu and Su, 2009), (ii) choose between multiple implementations for a specific service (Ranganathan and McFaddin, 2004), or (iii) determine whether a service should participate in future compositions (Karastoyanova et al., 2005).

In (Wieland et al., 2007), the authors propose an extension to WS-BPEL, named Context4BPEL, in order to explicitly model how context influences workflows. The Context4BPEL is defined according to the WS-BPEL extension mechanisms. This extension includes mechanisms to: (1) manage context events to allow the asynchronous reception of events; (2) query context data, and (3) evaluate transition conditions based on context data. However, Context4BPEL is realized as a ”BPEL Runtime Extension” and consequently also needs that the process execution engine supports it. In addition, the context information management depends on the Nexus platform.

In (Wieland et al., 2009), the authors propose a WS-BPEL extension that includes reference variables. With this kind of variables, the services can exchange pointers to variables instead of their values. Pointers are represented with EPRs. According to the value of an attribute of the extension, references are evaluated (1) upon activation of *Scope*, (2) before variables are used, (3) periodically, or (4) through an event sent from a external service. This extension is realized as a ”BPEL Language Transformation”, replacing references with WS-BPEL variables, inserting links to partners and interaction activities. The type (4) of references evaluation is similar to our work: the transformation adds a constructor *onEvent* to the process definition. However, these authors do not state how the external service addresses the event to the *onEvent* web service. Additionally, references evaluation depends on the RRS Service (Reference Resolution Service), a specific service available on the platform the authors propose.

George et al. (George and Ward, 2008; George, 2008) also propose a solution based on context variables. These authors extend WS-BPEL by adding new attributes to variables. However, the process definition must also contain explicitly the *Invoke* operation to realize the subscription. To realize the extension, they use a ”BPEL Runtime Extension” option, by changing the process execution engine, and they distinguish process instances through the Muse Apache platform (Apache Muse, 2013).

5 IoT WS-BPEL EXTENSION

In this section we present how we extend the WS-BPEL with context variables and with the *when-then* construct. We also describe how we realize the extension and we present an overview of our prototype.

5.1 Context Variables - Language Extension

Our main goal is to simplify the access to context information within WS-BPEL processes through the new concept of context variable. This way, each variable represent the current value of a specific sensor. As the language already provides a constructor for variables, we decided to define context variables by adding new attributes to the constructor. The new attributes represent the minimum information necessary to realize the subscribe operation according to the WS-Notifications standard, and, consequently, the minimum information required to identify a sensor through a web service.

The following example illustrates the definition of a context variable named *tempVar*, which maintains the value of a sensor. The web service that provides this value is identified by the attribute *publisherEPR* with the topic *Temperature*.

Example of a variable context definition:

```
<variables>
...
<variable name="tempVar" Type="xsd:anyURI"
  iotx:topic="Temperature"
  iotx:publisherEPR="http://192.168.1.43:8081/
                          pubService"/>
</variables>
```

In addition, the extension has to be defined in the process. In the following we present the extension definition, and its namespace and prefix.

Example of the extension definition:

```
<bpel:process name="myProcess"
...
xmlns:iotx="http://iot.extensions">
<extensions>
  <extension namespace="http://iot.extensions"
             mustUnderstand="yes"/>
</extensions>
...
</bpel:process>
```

In the next section we describe the realization of this part of our extension.

5.2 Context Variables: Language Transformation

As previously mentioned, WS-BPEL extensions can be realized as a runtime extension or as a model transformation. We realize the context variable part of our WS-BPEL extension with a model transformation. This option has the advantage of making it independent of the process execution engines. However, as the model transformation operation adds new activities, variables, etc., the process that is executed do not match exactly the process the modeler defined.

In the following, we detail the transformation we realize. We use the WS-Notification standard in the communication between processes instances and sensors. This transformation includes editing the WS-BPEL file and creating a WSDL file.

5.2.1 Editing the WS-BPEL File

Firstly, we remove the extension attributes from the context variables. These variables are now WS-BPEL standard variables of type xsd: anyURI.

Next, we change the main *Sequence* of the process by inserting a *Flow* activity after the first *Receive* activity, named *Start*. Inside the *Flow* activity, we define a *Sequence* activity to include the original process definition and a *Sequence* activity for each context variable. This way, *Sequence* activities that receive notifications run in parallel with the *Sequence* activity that has the original process definition.

The *Sequence* activities of each context variable include two main operations: the subscription operation and the reception of notifications.

Subscription Operation. The subscription operation is done with an *Invoke* activity. This *Invoke* activity calls the publisher EPR defined in the context variable. As the subscription operation is a two-way operation, we define two variables: the *inputVariable* and the *outputVariable*. Before the *Invoke* activity, we use an *Assign* activity to initialize the message the *Invoke* sends to the publisher. We format this message according to the WS-notification standard. The message is initialized with the topic declared in the context variable and the EPR to where the publisher sends notifications. The EPR is generated by concatenating the process name with the name of the context variable. The *output* variable is initialized in the *Invoke* response. As we initialize the *Correlation* in the *Invoke* response, we declare its initialization in the response. Below we show the subscription message and the *Invoke* activity.

Example of subscription message (Subscribe):

```
<wsnt:Subscribe ...>
 <wsnt:ConsumerReference>
  <wsa:Address ... >
   http://192.168.1.71:8080/ode/processes/
                            myProcesstempVar
  </wsa:Address>
 </wsnt:ConsumerReference>
 <wsnt:Filter>
  <wsnt:TopicExpression ... > Temperature
  </wsnt:TopicExpression>
 </wsnt:Filter>
</wsnt:Subscribe>
```

Invoke to perform the subscription:

```
<bpel:invoke name="Invoke"
          partnerLink="pubSubPartnerLink"
          operation="Subscribe"
          portType="wsntw:NotificationProducer"
          inputVariable="subscribeRequest"
          outputVariable="subscribeResponse">
  <bpel:correlations>
    <bpel:correlation set="notifyCorrelationSet"
          initiate="yes"
          pattern="response" />
  </bpel:correlations>
</bpel:invoke>
```

Operation to Receive Notifications. Processes receive notifications through a *Receive* activity (i.e., an inbound message activity). This activity uses a variable to save the notifications and the *Correlation*. In the following we illustrate the *Receive* activity.

Operation to receive notifications:

```
<bpel:receive name="Receive"
       partnerLink="pubSubPartnerLink"
       operation="Notify"
       portType="wsntw:NotificationConsumer"
       variable="NotificationMsg">
  <bpel:correlations>
    <bpel:correlation set="notifyCorrelationSet"
       initiate="no" />
  </bpel:correlations>
</bpel:receive>
```

Finally, after the *Receive* activity, we add an *Assign* activity to copy the value of the notification message to the context variable, as we present in the following.

Assign operation that updates the context variable:

```
<bpel:assign validate="no" name="updateVar">
  <bpel:copy>
    <bpel:from part="Notify"
             variable="NotificationMsg">
      <bpel:query queryLanguage="urn:oasis:
         names:tc:wsbpel:2.0:sublang:xpath1.0">
         wsnt:NotificationMessage/wsnt:Message
      </bpel:query>
    </bpel:from>
    <bpel:to variable="tempVar" />
  </bpel:copy>
</bpel:assign>
```

All process instances use the same port to receive notifications. To distinguish instances, we use the *Correlation*. In the following we describe how we use the *Correlation* to guarantee that each subscriber (process instance) receives its notifications.

As each context variable maps to a different subscription, we define a *correlationSet* for each context variable. We use the *Correlation* with two messages (*SubscriptionResponse* and *Notify*). Thus we define two rules and we use them in all the correlationSets. The rules states that, for each message, the correlations use the field *ReferenceParameters* of the element *SubscriptionReference*. The data type has to be the same as the field *ReferenceParameters*, i.e., *anyURI* (any type). We use the *Correlation* in the *Invoke* response, where the *correlationSet* is initialized, and in the *Receive*.

5.2.2 Adding WSDL Files

Each context variable is related with two services: the subscribe service and the service to where notifications are sent (the consumer service). We define these services in an additional WSDL file, which the transformed WS-BPEL process imports. This way, we avoid modifying the original WSDL file. We get the address of the subscribe service directly from the definition of the context variable, and we generate the address of the consumer service by concatenating the process name with the name of the context variable. We import the operations of each service from the WSDL files of the WS-Notification standard. This WSDL file also includes the *Correlation* properties we use in the WS-BPEL transformation.

5.3 When-then: Language Extension

We define the when-then as a top level construct. It is not an activity. As event handlers, when-then constructs are installed in parallel with each other. This construct has a condition and an activity, which is executed once, when the condition becomes true. In the following we present the when-then syntax and an example. In this example, we illustrate the use of a when-then construct that executes a *Sequence* with two empty activities when the value of *tempVar* is greater than 35.

The when-then syntax:

```
<iotx: when standard-attributes>
   <bpel:condition expressionLanguage="anyURI"?>
```

```
         bool-expr
   </bpel:condition>
   activity
</iotx:when>
```

A when-then example:

```
<process name="myProcess">
...
    <iotx:when name="testeWhen">
    <bpel:condition>$tempVar > 35</bpel:condition>
    <sequence name="teste">
        <empty name="empty1"> </empty>
        <empty name="empty2"> </empty>
    </sequence>
</iotx:when>
...
</process>
```

5.4 When-then: Language Transformation

We also realize this part of the extension with a language transformation approach. However, we use an auxiliary web service to detect the modification of variable values, avoiding busy waiting. In our implementation, these web services use the *Listeners* of Apache ODE.

In the following we detail the language transformation, which includes editing the WS-BPEL file and creating a WSDL file. Before that we describe the auxiliary web service.

5.4.1 The Auxiliary Web Service

The auxiliary web service monitors the value of when conditions. It provides the *RegisterWhen* operation that clients use to register the conditions they want that this service monitors. This service uses an ODE *Listener* to be informed when the value of a variable has been modified. In this situation, it evaluates conditions, and, when they become true, it sends a *UnlockWhen* to the respective process instance.

5.4.2 Editing the WS-BPEL File

Firstly we remove the when-then construct. To execute the when-then code in parallel with the main process logic, we add a *Flow* activity. We point out that the final WS-BPEL process definition only has an additional *Flow* activity with all the code added to support context variables and when-then constructs. For each when-then, we add:

- two message variables used with web service interactions,
- the *PartnerLink* and its roles,
- the *CorrelationSet*, and
- the *Sequence* activity.

The first operation of the *Sequence* activity is the *Assign*. We use the *Assign* operation to initialize the message we use to invoke the register operation in the auxiliary web service. This message has the condition, the EPR used to receive the message notifying that the when condition becomes true, and the identification of the process instance.

We illustrate this *Assign* operation in the following.

Inicialization of the message used to invoke the web service that monitors the when condition:

```
<assign validate="no"
        name="testeWhenSubscriptionCreation">
 <copy>
  <from>
   <literal xml:space="preserve">
     <when:RegisterWhenRequestElement>
       <when:WhenProcessReference>
         <wsa:Address ...>
           http://localhost:8080/ode/processes/
                             SubscribertesteWhen
         </wsa:Address>
       </when:WhenProcessReference>
       <when:Condition>$tempVar &gt; 35
       </when:Condition>
       <when:InstanceId> -1 </when:InstanceId>
     </when:RegisterWhenRequestElement>
    </literal>
   </from>
   <to variable="testeWhenWhenRequest"
       part="registerWhenRequest"/>
  </copy>
  <copy>
     <from> $ode:pid </from>
     <to variable="testeWhenWhenRequest"
         part="registerWhenRequest">
       <query ...> when:InstanceId </query>
     </to>
  </copy>
</assign>
```

After the *Assign* operation, we add an *Invoke* activity to call the *RegisterWhen* operation and a *Receive* activity. This activity is associated with the *UnlockWhen* operation, and, as it is a blocking activity, it will not continue until the process instance receives a message notifying that the conditions has become true. Both activities use the *CorrelationSet*, which distinguishes processes instances through their identification (InstanceId). Finally, after the *Receive*, the *Sequence* has the original activities of the when-then construct.

5.4.3 Adding WSDL Files

The WSDL files include two web services interfaces:

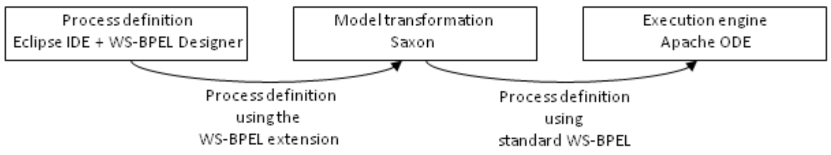
Process definition
Eclipse IDE + WS-BPEL Designer
Model transformation
Saxon
Execution engine
Apache ODE
Process definition using the WS-BPEL extension
Process definition using standard WS-BPEL

Figure 1: The prototype and its tools.

- the auxiliary web service
- the web service that corresponds to the *Receive* activity used to wait for the message notifying that the conditions has become true. The EPR of this web service is generated by concatenating the process name with the name of the when-then construct.

In addition, it includes the messages, the *PartnerLinkType*, and the correlation rules.

5.5 Prototype

We develop the prototype with the following tools:

- Eclipse EE + BPEL Designer plugin - modeling tool (Eclipse IDE, 2013; BPEL Designer Project, 2013),
- Saxon Home Edition - XLST Processor (Saxon Home Edition, 2013),
- Apache ODE - WS-BPEL execution engine(Apache ODE, 2013), and
- Apache Tomcat (Apache TomCat, 2013).

Figure 1 illustrates our toolchain prototype and how it performs model transformations.

6 CONCLUSIONS AND FUTURE WORK

Business processes can benefit significantly from the IoT information. The work we present in this paper aims at simplifying the access to this information within WS-BPEL processes. Through a WS-BPEL extension, processes can include context variables, whose value is updated transparently and asynchronously: the extension is responsible for the operations required to perform the communication between process instances and sensors, allowing process modelers to focus on business logic. We realize the extension through a language transformation mechanisms, allowing it to be as independent as possible from the process execution engine.

Future work includes maturing the prototype to add more validations in the language transformation, to support context variables defined inside *Scopes*, and to provide an extension to the Eclipse WS-BPEL Designer plugin, the tool we use to model processes.

ACKNOWLEDGEMENTS

This work was partially supported by the FCT through the PATI project (PTDC/EIAEIA/103751/2008) and the LaSIGE multi annual funding.

REFERENCES

Apache Muse (2013). URL: http://ws.apache.org/muse/ Page visited on February 10th, 2013.

Apache ODE (2013). URL: http://ode.apache.org/ Page visited on February 10th, 2013.

Apache TomCat (2013). URL: http://tomcat.apache.org/ Page visited on February 10th, 2013.

Box, D. and et al. (2004). Web services addressing (ws-addressing). IBM, W3C. URL: http://www.w3.org/Submission/ws-addressing/.

BPEL Designer Project (2013). URL: http://www.eclipse.org/bpel/ Page visited on February 10th, 2013.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web service definition language. Microsoft, IBM.

Clark, J., Deach, S., and Kay, M. (2007). Xsl transformations. Saxonica, Adobe.

Eclipse IDE (2013). Eclipse IDE for Java EE Developers. URL: http://www.eclipse.org/ Page visited on February 10th, 2013.

George, A. (2008). Providing context in ws-bpel processes. Technical report, Journal of the Electrochemical Society.

George, A. A. and Ward, P. A. S. (2008). An architecture for providing context in ws-bpel processes. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, CASCON '08, pages 22:289–22:303, New York, NY, USA. ACM.

Karastoyanova, D., Houspanossian, A., Cilia, M., Leymann, F., and Buchmann, A. (2005). Extending bpel for run time adaptability. In *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference*, EDOC '05, pages 15–26, Washington, DC, USA. IEEE Computer Society.

Kopp, O., Grlach, K., Karastoyanova, D., Leymann, F., Reiter, M., Schumm, D., Sonntag, M., Strauch, S., Unger, T., Wieland, M., and Khalaf, R. (2011). A classification of bpel extensions. *Journal of Systems Integration*, 2(4):3–28.

OASIS (2006). Web services notification (ws-notification) version 1.3. OASIS. URL: https://www.oasis-open.org/committees/wsn/.

OASIS (2007). Web services business process execution language version 2.0. Organization for the Advancement of Structured Information Standards.

Ranganathan, A. and McFaddin, S. (2004). Using workflows to coordinate web services in pervasive computing environments. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '04, pages 288–, Washington, DC, USA. IEEE Computer Society.

Saxon Home Edition (2013). URL: http://saxon.sourceforge.net/ Page visited on February 10th, 2013.

Wieland, M., Görlach, K., Schumm, D., and Leymann, F. (2009). Towards reference passing in web service and workflow-based applications. In *Proceedings of the 13th IEEE international conference on Enterprise Distributed Object Computing*, EDOC'09, pages 89–98, Piscataway, NJ, USA. IEEE Press.

Wieland, M., Kopp, O., Nicklas, D., and Leymann, F. (2007). Towards context-aware workflows. In *CAISE´07 Proceedings of the Workshops and Doctoral Consortium*. Citeseer.

Yu, L. and Su, S. (2009). Adopting context awareness in service composition. In *Proceedings of the First Asia-Pacific Symposium on Internetware*, Internetware '09, pages 11:1–11:10, New York, NY, USA. ACM.

Zeng, D., Guo, S., and Cheng, Z. (2011). The web of things: A survey (invited paper). *Journal of Communications*, 6(6).