Middleware Support for BPEL Workflows in the AO4BPEL Engine

Anis Charfi, Mira Mezini

Software Technology Group
Darmstadt University of Technology
{charfi,mezini}@informatik.tu-darmstadt.de

Abstract. This paper focuses on middleware concerns in BPEL workflows. When looking at those workflows from the implementation perspective, we observe that they have several BPEL-specific middleware requirements, which are not supported by current WS-* specifications and by most BPEL engines available to date. This demo paper will show the AO4BPEL Engine, which implements a container framework that allows the specification and enforcement of middleware requirements in BPEL processes. A *deployment descriptor* is used to specify the quality of service requirements of BPEL activities. A light-weight and aspect-based *process container* is used to enforce those requirements by calling dedicated *middleware Web Services*. We implemented those middleware Web Services by extending open source implementations of WS-* specifications for security, reliable messaging, and transactions.

1 Introduction

In BPEL [11], a composite Web Service is implemented by means of a workflow process, which consists of a set of interactions between the composition and the partner Web Services along with the flow of control and data around those interactions. Whilst the functional side of the composition is specified by the BPEL process, it is unclear how to handle non-functional middleware concerns such as security, reliable messaging, and transactions.

The BPEL specification does not address middleware issues and leaves that for good reasons to BPEL implementations, which should support these "deployment issues" somehow. We observe however, that current BPEL engines do no provide appropriate middleware support for BPEL processes. In addition, it is widely assumed that WS-* specifications such as WS-Security [18] and WS-ReliableMessaging [4] are sufficent to cope with the middleware requirements of BPEL processes, for example by attaching appropriate policies [5] to the WSDL of the composite Web Service or its partners. We argue that WS-* specifications support only some of BPEL middleware requirements, namely those which have corresponding operations and messages in WSDL. There are other BPEL-specific middleware requirements that cannot be mapped to WSDL and SOAP as they require knowledge about the process, its activities, and BPEL semantics. These requirements are not supported by WS-* specifications.

In [7], we presented a container framework, which addresses the problems of *specification* and *enforcement* of middleware requirements in BPEL processes. The framework introduces a *deployment descriptor*, which specifies the middleware requirements of the process activities and a *process container*, which intercepts the execution of the activities at well-defined points and plugs in calls for dedicated *middleware Web Services* to enforce those requirements. In this paper, we present an implementation of that framework on top of the AO4BPEL engine, which is an aspect-aware engine based on IBM's BPWS4J [15]. We will also present some BPEL middleware Web Services that we developed by extending open source implementations of WS-* specifications.

2 Middleware Requirements in BPEL Workflows

We distinguish *simple requirements*, which correspond to BPEL messaging activities and could be supported somehow by using existing WS-* specifications and *complex requirements*, which are specific to BPEL. We will elaborate on reliable messaging, security, and transactions.

2.1 Reliable Messaging

A simple requirement of BPEL messaging activities is the reliable delivery of their respective SOAP messages with guaranteed delivery assurances (e.g., without message loss and/or duplication). Consider for instance two messaging activities such as *reply* or one-way *invoke* that target different partners and are nested in a *sequence*. The corresponding messages should be delivered in the order, in which the activities appear in the *sequence*, i.e, the SOAP message of the first activity should be received by the respective partner before the SOAP message of the second activity. Current reliable messaging specifications [17, 4] do not support this complex requirement because it involves more than two end points (the process and the two partners). This requirement is about the ordered message delivery in multi-party BPEL interactions [10].

2.2 Security

Messaging activities have simple requirements such as integrity, confidentiality, and authentication, which are supported by WS-Security [18]. Complex requirements arise when we consider issues such as secure conversations, trust, and federation in the context of BPEL processes. For example, we assume that we have a *sequence*, which contains many messaging activities targeting the same partner. From a performance point of view, it is inefficient to secure each messaging activity individually as in WS-Security. Instead, using a security context (according to WS-SecureConversation [14]) for all interactions with that partner would improve the performance significantly.

2.3 Transaction

Composite BPEL activities might require transaction semantics e.g., in a *sequence* with two *invoke* activities it might be necessary that either both invocations succeed or both must be undone. It is also essential that the process and the partners decide together the outcome of the transaction, which is called *external coordination* [19]. This feature is not supported in BPEL. Moreover, a way is needed to flexibly configure the transactional requirements of activities according to the application semantics e.g., a *sequence* might need to be executed as an *atomic transaction* or as a *compensation-based transaction*. WS-* transaction specifications [12, 13] can support the transactional requirements of BPEL activities if the BPEL engine is integrated properly with the Web Service transaction middleware.

2.4 Discussion

The problem of middleware requirements in BPEL is two-fold. On the one hand, appropriate means are needed to specify the requirements of BPEL activities. On the other hand, appropriate infrastructure is necessary to enforce those requirements during process execution. Even for simple requirements, most current BPEL engine do not provide a way to say "this invoke or that reply must be secure". Moreover, the integration of BPEL engines with existing WS-* middleware is still problematic.

3 The AO4BPEL Engine and the Container Framework

Our framework has three main components: a deployment descriptor, a process container, and a set of middleware Web Services [7].

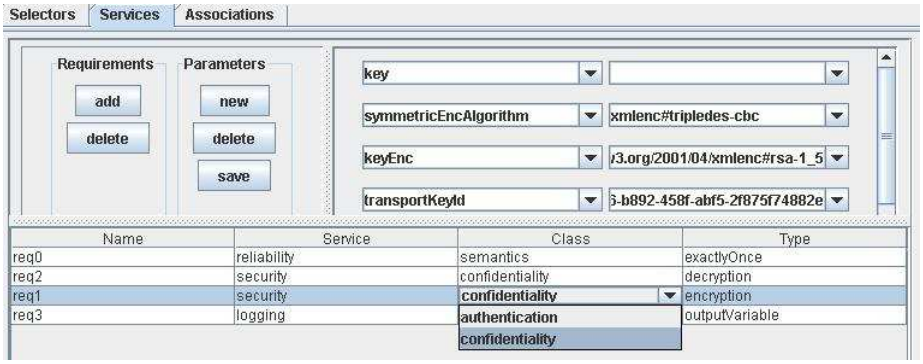

Fig. 1. Deployment Descriptor GUI Tool

3.1 The Deployment Descriptor

An XML-based deployment descriptor specifies declaratively the middleware requirements of the BPEL activities and the parameters that are needed to accomplish those requirements. Listing 1. shows an excerpt of a deployment descriptor for a bank transfer process, which calls the operations *credit* and *debit* on two partner Web Services.

The deployment descriptor defines one or more activity *selectors*, which are XPath expressions to identify the activities that will be associated with some requirement. The attribute *selectorid* attribute of the *requirement* element is used for this association. The *service* elements group requirements that belong to a specific middleware service.

```
<bpel-dd xmlns="http://www.st.informatik.tu-darmstadt.de/bpel-dd">
  <selectors>
    <selector id="0" name="credit" type="activity">/process//invoke[@operation="credit"]</selector>
    <selector id="1" name="debit" type="activity">/process//invoke[@operation="debit"]</selector>
  </selectors>
  <services>
    <service name="reliability">...
      <requirement name="req0" class="semantics" type="exactlyOnce" selectorid="0"/>...
    </service>
    <service name="security">...
      <requirement name="req2" class="confidentiality" type="decryption" selectorid="1"/>
       <parameters>
        <parameter name="symmetricEncAlgorithm">xmlenc#tripledes-cbc</parameter>
        <parameter name="keyEnc">http://www.w3.org/2001/04/xmlenc#rsa-1_5</parameter>
        <parameter name="transportKeyId">16c73ab6-b892-458f-abf5-2f875f74882e</parameter>...
       </parameters>
       </requirement>...
    </service>
  </services>
</bpel-dd>
```

Listing 1. The deployment descriptor

The deployment descriptor is the only component of the framework that the BPEL programmer needs to know about. He/she could write it manually or use the GUI tool shown in Fig. 1 to generate it.

One major advantage of the deployment descriptor against policies [16] is that the deployment descriptor allows the specification of the necessary parameters to enforce a given requirement. With policies, which are too declarative, this is not possible. For instance, in a policy is not possible to specify the user name and password that should be used to support authentication.

At process deployment time, the deployment descriptor file has to be specified in addition to the BPEL file as shown in Figure 2.

3.2 The Process Container

The process container is an implementation concept that BPEL programmers do not need to know about. It intercepts the execution of BPEL activities at well-defined points and calls dedicated middleware Web Services that provide the necessary functionality to enforce the middleware requirements.

We implemented a light-weight and aspect-based process container using a set of AO4BPEL aspects [6] that are automatically generated from the deployment descriptor. Automatic aspect generation is possible because the advices used

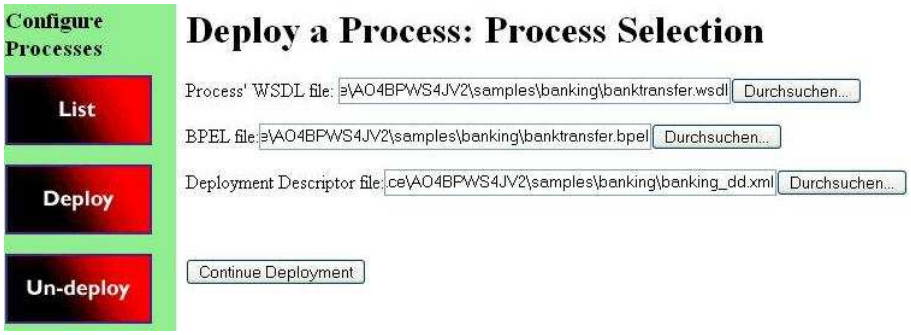

Fig. 2. Process Deployment in AO4BPEL

for integrating middleware Web Services follow well-defined patterns. E.g., for reliable messaging, there are recurring advice patterns for sending a message with exactly-once semantics, etc.

AO4BPEL [6, 9] is an aspect-oriented extension to BPEL. AO4BPEL supports *process-level join points* and *interpretation-level join points*. The former capture the execution of an activity. The latter capture internal points during the interpretation of an activity e.g., the point where a SOAP message of an *invoke* activity has been created. In AO4BPEL, an aspect defines one or more pointcuts and advices. A *pointcut* is a construct for selecting a set of join points e.g., to intercept some activities that have a common requirement. XPath is used as pointcut language in AO4BPEL. The *advice* is a BPEL activity that specifies some crosscutting functionality and can be used for example to enforce a requirement by calling a middleware Web Service.

After deploying the BPEL process (cf. Fig. 1), container aspects are automatically generated and deployed. We can see these aspects by switching to the process-list view of the AO4BPEL engine shown in Fig. 3.

Listing 2 shows the second aspect in the aspects list of Fig. 3. This aspect declares the reliable messaging service (RM) as partner and two variables for the input and output parameters of the call to *sendWithExactlyOnceSemantics*.

The pointcut of this aspect intercepts the *invoke* activity that calls the operation *credit* at the point where the SOAP request message has been created (this is the semantics of the advice type *around soapmessageout*). The advice contains an *assign* activity, which sets the input parameters of the call to the operation *sendWithExactlyOnceSemantics* of the RM Service. The SOAP message corresponding to the current join point (i.e., the *invoke* activity that calls the operation *credit*) is accessed by means of the special AO4BPEL context collection variable *soapmessage*. The second *assign* activity is used to pass the response message for the current join point activity from the RM Service to the BPEL process by using the special AO4BPEL context collection variable *newsoapmessage*. This is necessary because the request message for calling the operation *credit* was sent by the RM service on behalf of the BPEL process.

Fig. 3. The List of Container Aspects in AO4BPEL

```
<aspect name="credit_semantics_exactlyOnce">
  <partners><partner name="rmService" partnerLinkType="rms:RMService"/></partners>
  <variables>
    <variable messageType="rms:sendWithExactlyOnceSemanticsRequest" name="inputMessage"/>
    <variable messageType="rms:sendWithExactlyOnceSemanticsResponse" name="outputMessage"/>
  </variables>
  <pointcut name="creditExactlyOnce">/process//invoke[@operation="credit"]</pointcut>
  <advice type="around soapmessageout">
    <bpws:sequence>
      <bpws:assign>
        <bpws:copy><bpws:from part="message" variable="soapmessage"/>
                   <bpws:to part="message" variable="inputMessage"/></bpws:copy>
        <bpws:copy><bpws:from part="isInonly" variable="ThisJPActivity"/>
                   <bpws:to part="inonly" variable="inputMessage"/></bpws:copy>
        <bpws:copy><bpws:from part="partnerEndpoint" variable="ThisJPActivity"/>
                   <bpws:to part="endpoint" variable="inputMessage"/></bpws:copy>
      </bpws:assign>
      <bpws:invoke name="rmInvoke" operation="sendWithExactlyOnceSemantics" partner="rmService"
              inputVariable="inputMessage" outputVariable="outputMessage" portType="rms:RMService"/>
      <bpws:assign>
        <bpws:copy><bpws:from part="sendWithExactlyOnceSemanticsReturn" variable="outputMessage"/>
        <bpws:to part="newmessage" variable="newsoapmessage"/></bpws:copy>
      </bpws:assign>
    </bpws:sequence>
  </advice>
</aspect>
```

Listing 2. A container aspect for reliable messaging

3.3 The Middleware Web Services

The middleware Web Services are not part of the AO4BPEL engine. They are based on open source implementations of WS-* specifications.

The reliable messaging service [10] provides operations that are called by the container to enforce a delivery assurance for messaging activities (e.g., exactly-once) and to support the in-order delivery of messages even between more than two endpoints. Our implementation of this service is based on Apache Sandesha [1], which was extended to support mutli-party reliable messaging [10].

The security service [8] provides two port types: one for *secure messaging* according to WS-Security and one for *secure conversations* according to WS-SecureConversation with operations such as *createContext*, *encryptWithContext*, etc. The implementation of this service is based on Apache WSS4J [2].

The transaction service provides operations that are called by the container to enforce atomic transactions [12] such as *begin(transid)*, *participate(transid, soap)*, and *commit(transid)*. The implementation of this service is based on Apache Kandula [3], an implementation of WS-Coordination and WS-AtomicTransaction.

4 The Demo

In this demo, we will use a travel agency scenario with several BPEL processes that compose the Web Services of airline companies and hotel chains. We will deploy these processes on the AO4BPEL engine and specify a deployment descriptor file that defines the middleware requirements of the process activities.

Once the process is deployed, the audience will see how AO4BPEL container aspects will be generated automatically and activated. Then, we will start some instances of the deployed processes and use a tool to monitor the SOAP messages that are exchanged between the processes and their partners. Thus, we verify that the middleware Web Services are called correctly by the process container and the requirements of the different activities are fulfilled.

5 Conclusion

In this paper, we presented a user-friendly implementation of a container framework for the specification and enforcement of the middleware requirements of BPEL processes. The framework was inspired from enterprise component models and it can be reused with other BPEL engines. The engine has to provide a process container that is able to intercept the execution of BPEL activities and call the middleware Web Services. The deployment descriptor and the middleware Web Services could be reused without any changes.

The AO4BPEL engine presented here provides support for many BPEL-specific middleware requirements and paves the way toward a new breed of BPEL engines that we devise *application servers for BPEL*.

References

1. Apache. Sandehsa 1.0, July 2005.
2. Apache. Wss4j, March 2005.
3. Apache. Kandula 0.2, May 2006.
4. C. Ferris and D. Langworthy (Eds.). Web Services Reliable Messaging Protocol (WS-ReliableMessaging), February 2005.
5. C. Sharp (Eds.). Web Services Policy Attachment (WS-PolicyAttachment), September 2004.
6. Anis Charfi and Mira Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. In *Proceedings of the European Conference on Web Services (ECOWS)*, volume 3250 of *LNCS*, pages 168–182. Springer, September 2004.
7. Anis Charfi and Mira Mezini. An Aspect-based Process Container for BPEL. In *Proceedings of the 1st Workshop on Aspect-Oriented Middleware Development (AOMD)*, November 2005.

8. Anis Charfi and Mira Mezini. Using Aspects for Security Engineering of Web Service Compositions. In *Proceedings of the IEEE International Conference on Web Services (ICWS), Volume I*, pages 59–66. IEEE Computer Society, July 2005.
9. Anis Charfi and Mira Mezini. AO4BPEL: An Aspect-Oriented Extension to BPEL. *World Wide Web Journal: Recent Advances on Web Services (special issue), to appear*, 2006.
10. Anis Charfi, Benjamin Schmeling, and Mira Mezini. Reliable messaging in bpel processes. In *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS), to appear*, September 2006.
11. F. Curbera, Y. Goland, J. Klein, et al. Business Process Execution Language for Web Services (BPEL4WS) Version 1.1, May 2003.
12. D. Langworthy (Eds.). Web Services Atomic Transaction (WS-AtomicTransaction), November 2004.
13. D. Langworthy (Eds.). Web Services Business Activity (WS-BusinessActivity), November 2004.
14. M. Gudgin and A. Nadalin (Eds.). Web Service Secure Conversation Language (WS-SecureConversation) Version 1.0, February 2005.
15. IBM. The BPEL4WS Java Run Time, August 2002.
16. J. Schlimmer (Eds.). Web Services Policy Framework (WS-Policy)., September 2004.
17. OASIS. Web Services Reliable Messaging TC WS-Reliability 1.1, 15 November 2004.
18. OASIS. Web Services Security: SOAP Message Security Version 1.0, March 2004.
19. Stefan Tai, Rania Khalaf, and Thomas Mikalsen. Composition of coordinated web services. In *Proceeding of ACM/IFIP/USENIX International Middleware Conference (Middleware)*, volume 3231 of *LNCS*, pages 294–310. Springer, October 2004.