Middleware Services for Web Service Compositions

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

ABSTRACT

WS-* specifications cover a variety of issues ranging from security and reliability to transaction support in web services. However, these specifications do not address web service compositions. On the other hand, BPEL as the future standard web service composition language allows the specification of the functional part of the composition as a business process but it fails short in expressing non-functional properties such as security, reliability and persistence. In this paper, we propose an approach for the transparent integration of technical concerns in web service compositions. Our approach is driven by the analogy between web services and software components and is inspired from server-side component models such as Enterprise Java Beans. The main components of our framework are the *process container*, the *middleware services* and the *deployment descriptor*.

Categories and Subject Descriptors

H.3.5 [**Information Systems**]: Information Storage and Retrieval- Online Information Services: Web-based Services

General Terms

Design, Languages

Keywords

Middleware, Web Service Composition, BPEL

1. INTRODUCTION

The Web Services protocol stack has evolved from the basic specifications (WSDL, SOAP, UDDI) into a comprehensive stack with specifications addressing security, reliability, transaction, orchestration, etc. For orchestration, process-based composition languages like BPEL define the web service composition as a workflow of partner interactions. Whilst focusing on the functional side (control and data flow) of the composition, BPEL does not support the expression of non-functional properties. E.g., a requirement could be that an interaction with a partner (via the messaging activities `invoke`, `reply`, or `receive`) has to be reliable (exactly-once semantics), secure (authentication), transacted (part of an enclosing transaction).

WWW 2005, May 10–14, 2005, Chiba, Japan.
ACM 1-59593-051-5/05/0005.

While there exist specifications for at least some of these technical services, the link that make these services available to composition specifications is still missing. One could say that we just have to extend BPEL with new constructs to support non-functional concerns but this would result in a very complex language. We argue that the problem of technical concerns in BPEL is rather due to lack of an advanced *deployment* concept in web service compositions equivalent to deployment concepts as they are found in the world of enterprise component models, such as Enterprise Java Beans (EJB) [1] or CORBA Component Model (CCM) [2]. In these models, the components are freed from technical code and middleware services are provided to the component transparently by the *container* they live in, based on some metadata provided by the developer and deployer of the component. We propose a similar approach for composite web services motivated by the observation that web services are distributed components over the Web. We envision BPEL orchestration engines to evolve into *web application servers*.

The following sections talk about requirements and design of a framework for the integration of technical concerns in BPEL compositions. Then, we present some implementation ideas toward light-weight process containers.

2. REQUIREMENTS

For illustration, we consider an online banking scenario where a bank transfer is implemented as a web service composition involving two banks. Each bank web service provides two operations `credit` and `debit`. The corresponding process main activity is a `sequence` with two `invoke` activities (one calls the `credit` and the other calls the `debit`). We go through the requirements of this composition:

- *Transaction*: the `sequence` of the two invokes in the bank transfer process must run as a transaction i.e., either both `invoke` activities succeed or the transaction must rollback. The transaction coordinators of both partner web services must agree on the outcome of the transaction.
- *Security*: interactions with partners (e.g., bank)via messaging activities require authentication and authorization. The respective messages should not be tampered on their way to the partner (integrity) and as sensitive data is being exchanged encryption is necessary (confidentiality).
- *Reliability*: we want to ensure that each of the `invoke` messages is received by the partner reliably (although

the Internet is not a reliable medium) and with a guaranteed call semantics (e.g., *exactly-once*).

- *Persistence*: BPEL introduces variables as containers for process data. However, after the termination of the process instance all data is lost. In many B2B scenarios it is required that data is stored persistently.

3. DESIGN

The web services framework (with BPEL) can be seen as a component model for assembling complex web services using other web services. There is a high similarity between web service compositions and enterprise component models e.g., support for hierarchical composition, specification of required and provided interfaces, separation of interface and implementation. Enterprise component models introduce the concepts of a *container* as the runtime environment of components providing infrastructural services, *annotations* as the specification of component requirements for technical services, and the *application server* as the provider of middleware services. In EJB or CCM the client never interacts directly with the native component implementation, but rather via the container, which intercepts calls and plugs infrastructural services into the application.

3.1 Overview of the Framework

In analogy to the enterprise component world, we define a framework for the integration of middleware services in BPEL compositions. Its main components are the *process container*, the *middleware services* and the *deployment descriptor*. The process container plays the role of intermediary between the process and the infrastructural services so that it plugs in support for non-functional concerns. This support is provided by dedicated middleware services, which are in turn exposed as web services. The configuration of the non-functional requirements of BPEL activities is defined in an XML-based deployment descriptor, which specifies e.g., which `invoke` activities require security checks or must be encrypted, which variables are persistent, etc. The overall architecture is sketched in Figure 1.

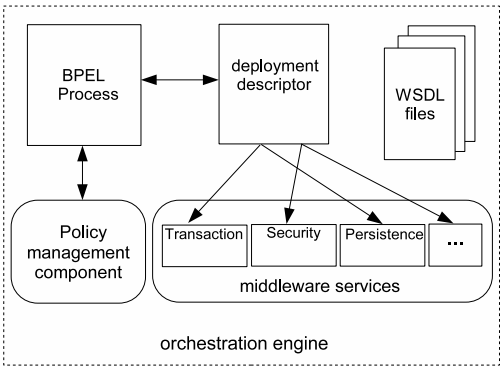

Figure 1: overview of the framework

3.2 Middleware Services for BPEL

Some technical concerns require the process and its partners to have certain capabilities e.g., support for distributed transaction or certain encryption algorithm. Capabilities, requirements, and preferences are described in WS-Policy, which is a declarative general model and syntax for policy expressions and policy assertions. Policies are managed by a special component, which checks whether the policies of the composition and its partner services are compatible. We propose an extensible set of middleware services:

- *BPEL Transaction Service* It is a transaction processing middleware that allows the specification of transaction boundaries, coordination types (WS-AtomicTransaction, WS-BusinessActivity), and protocols. It is used to create shared transaction contexts, register activities in a context, coordinate distributed web services, etc.
- *BPEL Security Service* enforces the security requirements of BPEL interactions. Internally, it reuses implementations of WS-Security, a specification that describes enhancements to SOAP to provide message integrity, message confidentiality, and single message authentication.
- *BPEL Reliability Service* enforces the reliability requirements of BPEL interactions. Internally, it uses mechanisms such as message identifiers, acknowledgment, sequencing, etc and can be based on implementations of WS-Reliability.
- *BPEL Persistence Service* enables the persistent storage of process variables as specified in the deployment descriptor (mapping from variables or parts of them to a database or file system).

4. IMPLEMENTATION

The drawback of the EJB or CORBA component models is that the set of supported middleware services is predefined and cannot be extended easily. Most EJB application servers lack openness i.e., the implementation of certain services cannot be replaced and the user cannot define new services. To solve this problem, several research works [3, 4] propose light-weight containers where the container is substituted by a set of aspects. We also follow an aspect-oriented [5] approach to implement BPEL process containers. The composition and the middleware services are integrated by means of AO4BPEL [6] aspects. AO4BPEL is an aspect-oriented extension to BPEL allowing for more flexibility and modularity. At deployment time, the process container is generated i.e., a set of aspects is created and activated according to the configuration found in the deployment descriptor. The aspects intercept points in the process execution and plug in calls to middleware services.

5. REFERENCES

[1] Linda G. DeMichiel. Enterprise JavaBeans Specification, Version 3.0. Technical report, 2004.
[2] Object Management Group. Corba components 3.0. Specification formal/02-06-65, OMG, June 2002.
[3] Roman Pichler, Klaus Ostermann, and Mira Mezini. On aspectualizing component models. March 2003.
[4] JBoss Inc. JBoss aop. http://www.jboss.org, 2004.
[5] Aspect-oriented sofware development. http://www.aosd.net.
[6] A. Charfi and M. Mezini. Aspect-Oriented Web service Composition with AO4BPEL. In *Proc. ECOWS 2004*, volume 3250 of *LNCS*, 2004.