Service Profiling in Business to Business Web Services

Bas Jansen

Thesis for a Master of Science degree in Electrical Engineering from the University of Twente, Enschede, The Netherlands

Graduation Committee:

Dr. ir. M. van Sinderen (University of Twente)

Dr. J. Pereira Filho (University of Twente)

Dr. L. Klostermann (Ericsson)

Ir. E. van der Velden (Ericsson)

Ing. E. Boersma (Ericsson)

Enschede, The Netherlands

December 2003

Abstract

It is a general trend that telecommunication and information technology are converging. This development enables new business opportunities for telecom operators: they can give application developers access to telecom capabilities by offering services like SMS and MMS messaging, call control, location based services, etc.

Web services are foreseen to be the dominant technology for business to business interactions over the Internet. Therefore the research starts with a state of the art overview on the large number of web services (related) specifications and standards, development and standardization, and web services initiatives in the telecom area.

When a business offers a service to another business, that service is usually a process involving other services where each fulfills a part of the overall business and technical requirements. Making services using such a process is called service profiling.

The main purpose of this research is to identify a suitable technique that can be used for service profiling. Four different technologies (BPEL, WSCI, Axis and a proprietary Java solution) are compared based on a set of functional and non-functional criteria. BPEL was found to be the most suitable and promising technique.

A prototype of a service profiling environment using BPEL was built. That prototype showed that BPEL is indeed suitable for service profiling, but it also showed some limitations when using BPEL for this purpose. Based on that experience, a number of improvements for BPEL are suggested to overcome the limitations. It is possible that these limitations will be resolved in a future BPEL version.

This report ends with the recommendation for Ericsson to watch BPEL closely as it is still in a premature phase (not suitable for telecom grade applications) but a very promising technique for service profiling in business to business web services, not only from a technical but also from a business perspective. Also, a market study needs to show if operators are willing to adopt this new approach in service offering and exact requirements need to be specified.

Preface

This thesis describes the results of a Master of Science assignment performed under the supervision of the Architecture of Distributed Systems group of the faculty of Electrical Engineering, Mathematics and Computer Science at the University of Twente. This assignment has been carried out from March to December 2003 at Ericsson Telecommunicatie BV in Rijen, the Netherlands.

I would like to thank Jan van der Meer, Lucas Klostermann, Erik van der Velden and Eltjo Boersma for giving me the opportunity to do this assignment at Ericsson and for their support and feedback during the assignment. Furthermore I would like to thank Marten van Sinderen and José Gonçalves Pereira Filho for their help and suggestions throughout the assignment.

Also, I want to thank everyone at Ericsson for making my stay in Rijen a very pleasant and informative experience, my fellow founders of Monito Online Applicaties for not having to spend too much time on work for Monito, and my girlfriend Judith Schevers for supporting me in this busy time.

Enschede, The Netherlands, December 2003.

Bas Jansen.

Table of contents

1 Introduction

This chapter presents the motivation and objectives of this thesis, a general introduction to service profiling and web services, and a brief overview of the structure of this report.

1.1 Motivation

Telephone and IT are converging. Nowadays it is common to send and receive SMS messages from a computer or website, to read your e-mail or browse web sites on your mobile phone or to set your phone preferences via a website. But the most interesting applications are still to come with the introduction of location based services, context aware applications, audio and video capabilities in mobile phones, and so on.

Because of these exciting new applications, telecom operators are looking into ways to 'open up the network' to give application developers access to telecom capabilities like SMS and MMS messaging, call control, etc. Most of the interfaces to these telecom capabilities today are based on telecom oriented (proprietary) APIs (Application Programming Interfaces). Standardization of these interfaces takes place in for example Parlay/OSA (Open Service Access or Open Service Architecture) [parlay].

Exposing telecom capabilities has two main drives: encourage innovation in the application area and enable new business scenarios for telecom operators and ASPs (Application Service Providers). Some efforts are mainly focused on the first point (e.g. JAIN), other efforts (especially Parlay and 3GPP OSA [3gpp]) focus on a combination of the two.

By providing application developers with APIs and protocols the number of developers that are able to develop applications increases dramatically, compared to traditional IN (Intelligent Networking) development. By exposing telecom capabilities, it becomes possible to develop applications mixing data communication (datacom) and telecommunication (telecom) features.

By exposing capabilities, but also provide support for controlled access to the capabilities, operators are able to sell access to both their network capabilities and their subscribers to external parties like ASPs. In addition, operators can allow others to provide part of the service portfolio for their subscribers, and can provide additional network capabilities to enterprise applications.

The last few years the main players in the IT industry have put a lot of effort on the development of XML web services, a platform independent technology for applications to discover and interact with other applications over the internet using XML messages. It is foreseen that these web services will eventually become the dominating technology for business to business (B2B) interactions.

This gives the telecom operators the opportunity to expose their assets via a web service interface, thus becoming players in the IT application market. The telecom industry is taking this avenue by among other things standardization efforts like Parlay/X web services and OMA (Open Mobile Alliance) [oma] web services.

In a market where an operator is a web service provider, it is important for operators that they are able to offer customized services, tailored to both their own and their customers' needs.

An offered service usually consists of a set of services that are invoked in a specific order. A service offering for sending SMS messages may consist of a logging service, charging service, privacy check service, and of course an SMS message sending service. In this example, the SMS message sending is the base service, while the other services are called auxiliary services. Linking these base and auxiliary services together is called service profiling. The service profiling process is the document or script that describes how these services are linked together. Using service profiling a (web) service provider can offer services to service consumers that are tailored to both business and technical requirements of the service provider and consumer.

1.2 Objectives

The purpose of this assignment is to analyze what technique(s) can best be used for service profiling in a telecom environment. The scope is limited to services implemented on the J2EE platform with an XML web service interface. Four different techniques will be reviewed and compared. These are:

- BPEL, the Business Process Execution Language for web services, possibly combined with additional web service standards like WS-Security and WS-Transaction
- WSCI, the Web Services Choreography Interface, possibly combined with additional web service standards like WS-Security and WS-Transaction
- Axis, the open source web service implementation on J2EE of the Apache project, provides a kind of service profiling solution through a 'handler' solution
- A proprietary Java (J2EE platform) solution

Furthermore, limitations will be identified and improvements to overcome those limitations will be suggested.

1.3 Approach

The four different techniques will be assessed and compared with respect to a number of functional and non-functional criteria. The functional criteria state that a service-profiling environment should include support for:

- Choreography to dynamically combine base and auxiliary services
- Orchestration to execute the base and auxiliary services in a specific order because of possible data or control dependencies
- Transactions, specifically atomic transactions (following the 'all-or-nothing' principle) for a reliable and robust service offering
- Service lifecycle management for easy offering of a new variant of an existing service, introducing new services or revoking service offerings

- Online choreography and orchestration so actual execution path of the process behind the service is determined the moment the service is used (runtime) and not when the service is deployed (deploy time)

- The service consumer must not be aware of the whole service profiling process (i.e. the interface to the service profiling process should be a web service itself)

The non-functional criteria are:

- Performance

- Availability of existing solutions

- Industry support for used standards

These criteria will be further explained in chapter 8 where the techniques are compared using these criteria.

As a proof of concept, a limited prototype of the most promising technique will be implemented. A number of test scenarios will be deployed to see if that technique is indeed suitable for service profiling.

Since the service profiling will take place in a telecom environment with high demands with respect to performance and availability, the prototype will be examined whether it supports SLA (Service Level Agreements) enforcement, dynamic updates and service lifecycle management.

1.4 Structure

Chapters 2 and 3 provide general information about web services and service profiling. Chapter 2 contains an overview and state-of-the-art of the web services area, with a focus on web services in the telecom industry.

Chapters 4 to 7 discuss the different technologies that are compared, while chapter 8 covers a summary of the comparisons. Chapter 9 contains a description of the prototype that is implemented with the best technique according to chapter 8.

Based on the experience gained during the research, chapter 10 will identify the limitations of the technique used for the prototype and suggest improvements.

Finally, chapter 11 contains the conclusions regarding service profiling and some recommendations for Ericsson in this area.

2 Web services

This chapter gives a state-of-the-art overview of web services. First, the basic standards that are the basis of all web services are introduced. Then a number of additional web service standards and (proposed) specifications are discussed. These additional standards cover aspects of web services in the area of business processes, security, messaging, reliability, etc. Section 2.4 describes the development and standardization of web services, followed by the discussion of a number of related (upcoming) standards. Section 2.6 provides an overview of web services initiatives in the telecommunications industry.

2.1 What is a web service?

The W3C Web Services Architecture Working Group defines a web service as follows:

A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.
http://www.w3.org/TR/ws-gloss/
W3C Web Services Architecture Working Group, November 2002

So web services are basically a way for applications to discover and interact with other applications over the Internet using XML.

2.2 Basic standards

A web service architecture consists of three primary functions: discovery, description and transport. For each of these functions there is an XML based standard. Web services are described by the Web Services Description Language (WSDL), discovered though Universal Description, Discovery and Integration (UDDI) and transported using the Simple Object Access Protocol (SOAP).

2.2.1 WSDL

The Web Services Description Language (WSDL) is an XML-based language for describing web services. A WSDL document defines services as collections of network endpoints, or ports, and describes the message interactions. A WSDL document consists of the following elements to describe a service:

- Types, data type definitions to describe the messages exchanged
- Messages, an abstract definition of the exchanged data messages
- Operations, specifying the input and/or output messages
- Port types, a named set of abstract operations

- Bindings, defines message format and protocol bindings for a particular port type
- Ports, for associating a binding to an actual service endpoint address
- Services, a set of ports.

Specification	Version and status	Editor(s)	Standardization
WSDL	1.1, W3C Note, March 15, 2001	Ariba, IBM, Microsoft	W3C
	1.2, W3C Working Draft, November 10, 2003	Sun, Microsoft, IBM, Canon	W3C

Table 2-1: Overview of WSDL specifications

2.2.2 UDDI

Universal Description, Discovery and Integration (UDDI) [uddi] is an XML based registry where web service providers can register their web service and web service consumers can search for suitable web services. UDDI registries can act as a global directory for web services.

A listing in a UDDI registry consists of three elements. At the highest level there are White Pages, which contain basic information about the providing company and its services. Next are Yellow Pages, which organize services by industry, service type or geographical location. Finally there are Green Pages, which include the technical mechanics (for example, a link to the WSDL) about how to find and execute a Web service [govatos].

UDDI was developed by IBM, Microsoft and Ariba, and is now under supervision by a technical committee of the Organization for the Advancement of Structured Information Standards (OASIS). Version 3 is the most current version although version 2 is still most widely used.

Specification	Version and status	Editor(s)	Standardization
UDDI	2, OASIS Open Standard, July 19, 2002	IBM, BEA, Microsoft, HP, Sun, Verisign, others	OASIS
	3, OASIS Committee Specification, July 19, 2002	IBM, Microsoft, HP, Oracle, SAP, Sun, Verisign, others	OASIS
	3.01, OASIS Committee Specification, October 14, 2003	IBM, Microsoft, SAP, France Telecom, Oracle, others	OASIS

Table 2-2: Overview of UDDI specifications

2.2.3 SOAP

The Simple Object Access Protocol (SOAP) [soap] describes the XML documents and processing model that are used in the message exchange between web services and web service users. A SOAP message consists of an envelope containing a header and a body part. SOAP messages can be transported over a variety of transport protocols. However, HTTP over TCP/IP is most widely used.

A SOAP sender sends SOAP messages to an ultimate SOAP receiver via zero or more SOAP intermediaries. A SOAP node acts in one or more roles when processing a SOAP message. SOAP header blocks can be targeted to be processed by a node with a specific role.

The SOAP Processing Model defines a distributed, stateless processing model for SOAP messages. When a node receives a SOAP message, it first has to determine what roles apply. Then all mandatory header blocks targeted at the node are identified and checked if the nodes understand them. Then the SOAP header blocks and, only in case the node is the ultimate receiver, the SOAP body are processed. If the node is not the ultimate receiver, the SOAP message is sent further down the SOAP message path.

Specification	Version and status	Editor(s)	Standardization
SOAP	1.1, W3C Note, May 8, 2000	IBM, Microsoft, Lotus, DevelopMentor, UserLand	W3C
	1.2, W3C Recommendation, June 24, 2003	Microsoft, Sun, IBM, Canon	W3C

Table 2-3: Overview of SOAP specifications

2.3 Additional web service standards

Although the basic web service specifications UDDI, WSDL and SOAP provide the basic means for web service description, discovery, invocation and message transport, a lot of issues like security, reliability, transactions, etc. are not addressed in those specifications. Therefore additional web service specifications have been defined and are still being developed today. These additional specifications will be discussed in the next sections.

2.3.1 Business processes

There are two main upcoming standards for linking several web services together (choreography) and execute them in a specific order because of possible data or control dependencies (orchestration): the Business Process Execution Language for Web Services (BPEL4WS, often pronounced as "beepel") [bpel1.1] and the Web Service Choreography Interface Language (WSCI, pronounced as "whiskey") [wsci].

The first BPEL4WS development was done by Microsoft, IBM and BEA Systems, joined later on by Siebel Systems and SAP. For a long time this standard was not submitted to a standardization body because Microsoft had made no decision whether they wanted to offer the standard on a royalty-free basis [berlind]. But recently, the OASIS Web Services Business Process Execution Language (WSBPEL) Technical Committee was formed to continue the work on the BPEL4WS specification. For more information, see the chapter about BPEL.

WSCI has been submitted to W3C a lot earlier to form the W3C Web Services Choreography Working Group. Because of the submission of BPEL4WS to OASIS, the W3C Working Group is inviting BPEL TC members to its meetings to coordinate efforts. For more information, see the chapter about WSCI.

Specification	Version and status	Editor(s)	Standardization
BPEL4WS	1.0, initial public draft, July 31, 2002	Microsoft, IBM, BEA	
	1.1, second public draft, March 31, 2003	Microsoft, IBM, BEA, Siebel Systems, SAP	OASIS
WSCI	1.0, W3C Note, August 8, 2002	BEA, Intalio, SAP, Sun, W3C	W3C

Table 2-4: Overview of business process related web service specifications

2.3.2 Security

Security is an important aspect of web services, since services will be exposed over the (public) Internet. The Web Services Security (WS-Security) [wssecurity] specification adds security features to web services by extending SOAP messages with standard XML security technologies such as XML encryption and XML digital signatures. The specification does not specify any implementation specifics such as PKI or Kerberos. WS-Security provides message integrity, message confidentiality and single message authentication and forms the foundation for other WS-Security standards (refer to figure 2-1). The Web Services Security Addendum (WS-Security Addendum) describes clarifications, enhancements, best practices, and errata of the WS-Security specification.

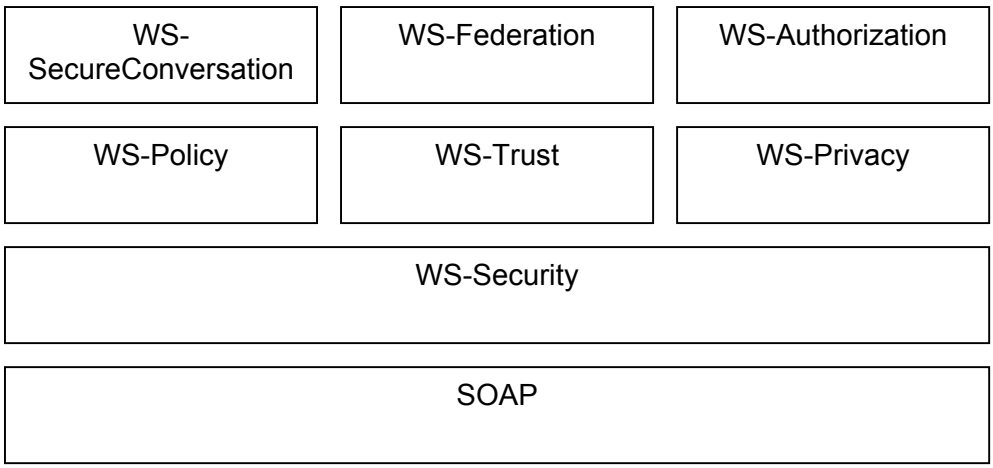

Figure 2-1: WS-Security architecture [della-libera]

The main additional specifications of the WS-Security architecture are listed below. Since WS-Security is a composable architecture, not all specifications need to be used by all web services. Note that not all specifications have been publicly published already [wssecurity].

- WS-SecureConversation describes how to manage and authenticate message exchanges, establish and share security contexts, and derive session keys from security contexts.

- The Web Services Policy Framework (WS-Policy) [wspolicy] provides a model and syntax to express policies (service requirements, preferences and capabilities) of a web service. The Web Services Policy Assertions Language (WS-PolicyAssertions) specifies a set of common message policy assertions that can be specified within a policy, while the Web Service Policy Attachment (WS-PolicyAttachment) specification defines how to associate policy expressions with WSDL type definitions and UDDI entries.

- WS-SecurityPolicy is an addendum to WS-Security and indicates the policy assertions for WS-Policy that apply to WS-Security.

- WS-Federation defines mechanisms that are used to enable identity, attribute, authentication, and authorization federation across different trust realms. The WS-Federation specification defines the model and framework for federation. Additional profiles define in detail how different

requestors apply the model. Presently, two profiles have been defined: *active* (SOAP enabled applications) and *passive* (web browsers etc.) requestors.

- WS-Trust defines a model for requesting and issuing security tokens and for management of trust relationships.
- WS-Authorization will define how access policies for web services are specified and managed.
- WS-Privacy will provide a model for how a privacy language (like P3P) can be used with WS-Policy and WS-Security and how privacy statements can be evaluated using WS-Trust.

The Web Services Security Profile for XML-based Tokens (WS-Security Tokens) describes how to use XML-based tokens such as the Security Assertion Markup Language (SAML) or the eXtensible rights Markup Language (XrML) with the WS-Security specification.

Specification	Version and status	Editor(s)	Standardization
WS-Security	Working Draft 17, August 27, 2003	IBM, Microsoft, Verisign, Sun, others	OASIS
WS-Security Addendum	August 18, 2002	IBM, Microsoft, Verisign	
WS-Security Tokens	August 28, 2002	IBM, Microsoft, Verisign	
WS-Federation	July 8, 2003	IBM, Microsoft, Verisign, RSA, BEA	
WS-Trust	1.0, initial public draft, December 18, 2002	Microsoft, IBM, Verisign, RSA	
WS-SecureConversation	1.0, initial public draft, December 18, 2002	Microsoft, IBM, Verisign, RSA	
WS-SecurityPolicy	1.0, initial public draft, December 18, 2002	Microsoft, IBM, Verisign, RSA	
WS-Policy	1.1, initial public draft, May 28, 2003	Microsoft, IBM, BEA, SAP	
WS-PolicyAssertions	1.1, initial public draft, May 28, 2003	Microsoft, IBM, BEA	
WS-PolicyAttachment	1.1, initial public draft, May 28, 2003	Microsoft, IBM, BEA, SAP	

Table 2-5: Overview of security related web service specifications

2.3.3 Messaging

There are several specifications that add extra features to SOAP messaging like routing, addressing, meta data and attachments.

The Web Services Routing Protocol (WS-Routing) [wsrouting] adds extra headers to the envelope of a SOAP message that can be used to specify the next node in a message path. Several WS-Routing headers together can specify the complete message and return path of a SOAP message. These paths do not need to be known when the message is sent: SOAP routers along the message path can add extra WS-Routing headers. The Web Services Referral Protocol (WS-Referral) [wsreferral] is a protocol that can be used to configure routing tables in SOAP routers.

Web Services Addressing (WS-Addressing) [wsaddressing] introduces SOAP headers to address web service endpoints and to secure end-to-end endpoint identification in messages in a transport-neutral way. WS-Addressing also introduces a message ID and a message correlation ID. The Web Service Callback Protocol (WS-Callback) [wscallback] defines SOAP headers to specify a callback address for asynchronous responses to a SOAP request. SOAP-Conversation elaborates on that principle and enables a subscribe/notify scenario: the subscribe request contains a callback location and a conversation ID. All notifications are sent to the callback location with the conversation ID included in the SOAP header for correlating the message with the original subscription.

Web Services Message Data (WS-MessageData) [wsmsgdata] allows the addition of meta-data about the message. The current version (0.91) specifies two specific types of meta-data: a message ID that relates to the current message and a message ID that relates to another message.

SOAP Messages with Attachments (SOAP-Attachments or SwA) [soapattach] describes the encapsulation of a SOAP message with attachments in a MIME (Multipurpose Internet Mail Extensions) message. BEA, Microsoft and others are working on a new version of SOAP-Attachments [soapattach2] because the current version is under-specified with respect to the XML Infoset and with respect to the processing model of SOAP. The new version also introduces a method to include base64-encoded data within the SOAP envelope. This has the advantage that standard XML processing techniques can still be used, as is not the case with MIME or DIME encapsulation.

Web Service Attachments (WS-Attachments) [wsattach] describes the encapsulation of a SOAP message and zero or more attachments into a DIME (Direct Internet Message Encapsulation) [dime] message. DIME is more efficient than MIME with respect to message processing because of binary headers and a record size field in the header.

Specification	Version and status	Editor(s)	Standardization
WS-Referral	October 23, 2001	Microsoft	
WS-Routing	October 23, 2001	Microsoft	
WS-Addressing	March 13, 2003	Microsoft, IBM, BEA	
WS-Callback	0.91, proposal, February 26, 2003	BEA	
SOAP-Conversation	1.0, June 13, 2002	BEA	
WS-MessageData	0.91, proposal, February 26, 2003	BEA	
SOAP-Attachments	1.0, W3C Note, December 11, 2000	HP, Microsoft	W3C
	0.61, April 1, 2003	BEA, Microsoft, AT&T, SAP, Canon	
WS-Attachments	internet draft, June 17, 2002	Microsoft, IBM	IETF
DIME	Internet draft, June 17, 2002	Microsoft, IBM	IETF

Table 2-6: Overview of messaging related web service specifications

2.3.4 Reliability

There are three initiatives for a reliable messaging standard for web services [chappell]. All specifications or specification sets provide the same methods for reliable transport: acknowledgements, retransmissions, message ordering and duplication detection.

- Web Service Reliability (WS-Reliability) [wsreliability]. Shortly after the announcement of this specification by Sonic, Sun, NEC, Fujitsu, Oracle, SAP, webMethods and many others, OASIS formed a Technical Committee (TC). The WS-Reliability specification does not cover all issues and was purely intended to be a starting point for the TC.

- Web Services Reliable Messaging Protocol (WS-ReliableMessaging) [wsreliablemsg] and WS-Addressing. These specifications by Microsoft, IBM, BEA and Tibco are very similar to WS-Reliability.

- Web Services Acknowledgement Protocol (WS-Acknowledgement) [wsack], WS-Callback and WS-MessageData. This set of specifications by BEA defines SOAP-based reliable messaging.

Apart from the SOAP-based specifications, IBM proposed Reliable HTTP (HTTPR) [httpr] as layer on top of the HTTP protocol for reliable message transport over the Internet.

Specification	Version and status	Editor(s)	Standardization
WS-Reliability	0.83, Working Draft, November 18, 2003	Sonic, Sun, NEC, Fujitsu, Oracle, Hitachi	OASIS
WS-ReliableMessaging	March 13, 2003	Microsoft, IBM, BEA, Tibco	
WS-Acknowledgement	0.91, proposal, February 26, 2003	BEA	
HTTPR	1.1, December 3, 2001	IBM	

Table 2-7: Overview of reliability related web service specifications

2.3.5 Transactions

Web Services Coordination (WS-Coordination) [wscoordination] describes a framework for how individual web services can interact in order to accomplish a task. The framework includes a context for the coordination and the exchanged messages that are needed in order for transactions to complete successfully as part of an overall business process defined in the Business Process Execution Language (BPEL4WS).

Web Service Transaction (WS-Transaction) [wstransaction] defines two coordination types that are used in the coordination framework described in WS-Coordination: an Atomic Transaction (AT) is used for the coordination of a set of activities of short duration following the 'all-or-nothing' principle. This means that all activities, or no activities at all are carried out. If an activity fails, tasks that have already been completed can automatically be undone. A Business Activity (BA) usually takes more time than an AT, making it impossible to do a rollback in case of a failure. In a BA, business logic is used to handle exceptions. The combination of AT and BA protocols support a variety of business processes, including those found in the Business Process Execution Language (BPEL4WS) specification.

In September 2003, Microsoft, IBM and BEA published a revised version of the complete Web Services Transaction Framework [wstransaction2]. The new WS-Coordination specification from September 2003 replaces the August 2002 version. The Web Services Atomic Transaction (WS-AtomicTransaction) and the (soon to be published) Web Services Business Activity (WS-BusinessActivity) replace the WS-Transaction specification.

An alternative to the WS-Coordination and WS-Transaction couple there is the Business Transaction Protocol (BTP) [btp], which is a more general and less complicated protocol. Moreover, OASIS already standardized BTP. Overviews and comparisons can be found in [coverpages] and [furniss].

The OASIS Web Services Composite Application Framework Technical Committee (WS-CAF TC) was formed in September 2003 to further develop and standardize the Web Services Composite Application Framework (WS-CAF).

WS-CAF (backed by Sun, Oracle, and others) is the counterpart of Microsoft and IBM's WS-Coordination and WS-Transaction specifications. The framework is aimed at solving the problem of coordinating multiple web services and consists of three specifications. WS-CAF is supposed to be compatible with existing specifications like BPEL, WS-Transaction and BTP [taft]. The authors of WS-Coordination and WS-Transaction have been contacted to contribute their specifications to the TC.

The Web Service Context (WS-CTX) defines a framework for context management that enabled web services to share a common context to share information about a common end result. The Web Service Coordination Framework (WS-CF) notifies web services involved in a transaction of a certain outcome. Web Service Transaction Management (WS-TXM) enables web services to negotiate about a common outcome of a transaction.

Specification	Version and status	Editor(s)	Standardization
WS-Coordination	August 2002	Microsoft, IBM, BEA	
	September 2003	Microsoft, IBM, BEA	
WS-Transaction	August 2002	Microsoft, IBM, BEA	
WS-AtomicTransaction	September 2003	Microsoft, IBM, BEA	
WS-BusinessActivity	To be published		
BTP	1.0, OASIS Committee Specification, June 3, 2002	Choreology, Sun, BEA, Oracle, others	OASIS
WS-CTX	1.0 draft, July 28, 2003	Sun, Oracle, Iona, Arjuna, Fujitsu	OASIS
WS-CF	1.0 draft, July 28, 2003	Sun, Oracle, Iona, Arjuna, Fujitsu	OASIS
WS-TXM	1.0 draft, July 28, 2003	Sun, Oracle, Iona, Arjuna, Fujitsu	OASIS

Table 2-8: Overview of transaction related web service specifications

2.3.6 User interface

The Web Services User Interface (WSUI) [wsui] specification uses a simple XML schema to provide web services with a user interface using XSLT style sheets to create for example HTML or WML views.

The Web Services eXperience Language (WSXL) [wsxl] defines a framework for enabling businesses to offer one web service through multiple channels. Each service offering can be customized using WSXL components. WSXL describes three types of basic components: data components, presentation components and control components. Each of the components can be configured to customize the output by means of an Adaptation Description in the Adaptation Description Language (ADL). One web service and one or more components can be choreographed together using code, or for example the choreography mechanisms of BPEL4WS, since each component has a web service interface.

Both the WSUI and WSXL specifications form the input of the Web Services for Interactive Applications (WSIA) OASIS Technical Committee [wsia]. The WSIA and Web Services for Remote Portlets (WSRP) [wsrp] OASIS groups are working closely together since WSIA and WSRP are both standards for visual, user-facing web services components. WSIA intends to define a general interface to display web service components in any type of web application, while WSRP intends to define the specific interface for the case when that web application is a portal. Both groups intend to define a common interface so WSIA components can be used in portals and WSRP components can be used in WSIA applications [freedman].

Specification	Version and status	Editor(s)	Standardization
WSUI	1.0, working draft, July 26, 2002	Epicentric (now Vignette)	
WSXL	2, IBM Note, April 10, 2002	IBM	
WSIA		IBM	OASIS
WSRP	1.0, OASIS Standard, September 3, 2003	IBM, Vignette, Novell, Netegrity Oracle, Crossweave, WebCollage	OASIS

Table 2-9: Overview of user interface related web service specifications

2.3.7 Other

The Web Service Inspection Language (WSIL or WS-Inspection) offers a way to inspect what services are available on a specific site or server. WS-Inspection offers a much more simple way of publishing web services than in a central UDDI registry, but also has less functionality.

The OASIS Web Services Distributed Management (WSDM) Technical Committee will work on developing a standard for the management of web services, for example in a B2B situation where trusted business partners will want the ability to manage each other's web services. Computer Associates, IBM, and Talking Blocks have submitted their Web Services Manageability 1.0 (WS-Manageability) specification that consists of three documents [wsmanage]: Web Services Manageability Concepts, Specification and Representation. The specification introduces general concepts of a manageability model, manageability implementation patterns and discovery considerations.

Specification	Version and status	Editor(s)	Standardization
WS-Inspection	1.0	Microsoft, IBM	
WSDM		Novell, IBM	OASIS
WS-Manageability	1.0, September 10, 2003	Talking Blocks, Computer Associates, IBM	OASIS

Table 2-10: Overview of other web service specifications

2.4 Development and standardization

A lot of companies are involved in the development of web services, but four names are represented in almost every aspect of web services: IBM, Microsoft, BEA Systems and Sun Microsystems. Other companies like Verisign, RSA, Oracle, SAP, HP and many others are also involved, but only in one or just a few web service areas. Since so many companies are involved, standardization and interoperability are very important issues.

Three non-profit organizations have formed technical committees or working groups to coordinate the development and standardization of web services: the World Wide Web Consortium (W3C), the Internet Engineering Task Force (IETF) and OASIS. The Web Services Interoperability Organization (WS-I) addresses interoperability issues between web services.

2.4.1 W3C

The web services activities of the W3C are structured into four Working Groups, apart from a Coordination Group: the Web Services Architecture Working Group defines the overall architecture, the XML Protocol Working Group defines the SOAP and SOAP with attachments protocols, the Web Services Description Working Group defines the WSDL specification and the Web Services Choreography Working Group works on the WSCI specification.

2.4.2 IETF

The two web services related standards submitted to IETF by IBM and Microsoft (DIME and WS-Attachments) were individual submissions and not part of any working group or activity at the IETF.

2.4.3 OASIS

OASIS is a non-profit global consortium that drives the development, convergence and adoption of e-business standards and has over 600 corporate and individual members from all over the world. The development and standardization work within OASIS is structured into Technical Committees (TCs) which each deal with their (part of a) specific standard.

2.4.4 WS-I

With so many companies and bodies involved in the development and standardization of web services, and the large number and versions of web services standards, it will be difficult to guarantee interoperability across platforms, applications and programming languages. That is why over 170 companies formed the Web Services Interoperability Organization (WS-I), lead by Microsoft and IBM. The deliverables of the WS-I are:

- A set of profiles containing a list of web service specifications including version numbers together with guidelines how the specifications should be used.
- Testing tools that can be used to monitor and analyze the interactions with a web service to create an interoperability report.

- Use cases and usage scenarios capturing the business and technical requirements of the use of web services.

- Sample applications that are implementations of the use cases and usage scenarios and conform to a given set of profiles. These sample applications are implemented on multiple platforms using different languages.

Other interoperability testing organizations mainly focus on one particular specification while WS-I tests conformance on a higher level for a profile or a set of specifications. For example, the SOAP interoperability tests of the Soapbuilders organization are focused on platform tool interoperability to ensure that tools created by platform vendors create interoperable web services. The WS-I is more focused on the guidance of the users of those tools (the web service implementers) to create interoperable and conformant web services.

In August 2003 WS-I approved Basic Profile 1.0 that covers the SOAP 1.1, WSDL 1.1, UDDI 2.0, XML 1.0 and XML Schema specifications [basicprofile].

2.5 Related standards

XML web services are not the only way to exchange information and conduct electronic business over the Internet using XML messages. XML-RPC and ebXML are two other important standards. In the area of security, SAML, XACML and the Liberty Alliance Project are important initiatives.

2.5.1 XML-RPC

XML-RPC [xmlrpc] is a very simple specification for Remote Procedure Calls (RPC), defined by Userland in 1998. It defines request, response and fault XML messages that are exchanged using HTTP POST. Because the standard has been around since 1998 there are quite a few implementations available in several programming languages. The specification does not include any discovery or description methods, nor does it discuss security issues.

2.5.2 ebXML

The mission of the Electronic Business using eXtensible Markup Language (ebXML) [ebxml] working group established by UN/CEFACT and OASIS is:

ebXML enables enterprises of any size, in any location to meet and conduct business through the exchange of XML-based messages.
http://www.ebxml.org
ebXML Working Group

The ebXML architecture contains an ebXML Registry where businesses can register their Collaboration Protocol Profile (CPP). A CPP describes a company's ebXML capabilities, constraints, implementation details and supported business scenarios. If some company discovers business scenarios of another company they would like to engage in, both companies form a Collaboration Protocol Agreement (CPA) containing the mutually agreed upon business scenarios and specific agreements. If the CPA is accepted the companies are ready to engage in electronic business transactions using ebXML.

ebXML focuses on the business processes between two enterprises where web services have a more general context. Not only technical but also business process aspects of electronic business transactions are described by the documentation and specifications.

The basic functions of ebXML have an overlap with XML web service functions but generally have something extra. An ebXML web service is described in CPP, whereas an XML web service is described in WSDL. A CPP contains the same information as a WSDL, but also some other parameters like the role of an organization in the context of a particular service, error handling and failure scenarios. A service can be published and discovered using an ebXML registry or a UDDI registry. An ebXML registry provides more information with respect to business profiles, processes and documents. The messages in an ebXML transaction are transported using the secure and reliable ebXML Messaging Service. Although this service uses SOAP over HTTP, it adds features like CPA management [irani].

2.5.3 SAML

The Security Assertions Markup Language (SAML) [saml] is an XML framework for exchanging authentication and authorization information and is being developed by the OASIS Security Services Technical Committee.

The information items expressed in SAML are assertions about a subject (person, computer) that has an identity in some security domain. These assertions can contain information about authentication, authorization and attributes of a subject. SAML authorities issue assertions: authentication authorities, policy decision points and attribute authorities.

SAML defines a protocol for the communication between SAML authorities and clients. Clients can request an assertion from a SAML authority. This authority can use various sources of information (like Radius, LDAP, or other SAML assertions) to form the response to the client. SAML can be used over many different transport methods but currently only the binding to SOAP over HTTP is defined.

Specification	Version and status	Editor(s)	Standardization
SAML	1.0, OASIS Open Standard, November 5, 2002	VeriSign, Sun, others	OASIS
	1.1, Oasis Standard, September 2, 2003	Sun, Netegrity, RSA Security, others	OASIS

Table 2-11: Overview of SAML specifications

2.5.4 XACML

The eXtensible Access Control Markup Language (XACML) [xacml] is designed to express access control policies for information access over the Internet. The authorization decision model is shared by SAML and XACML, based on the ISO IETF model. The model includes several entities [anderson]:

- Attribute Authorities that provide information about subjects, resources, etc.

- Authentication Authorities that state which individuals have authenticated and how they are authenticated
- Policy Administration Points (PAP) that create policies
- Policy Enforcement Points (PEP) that generate Authorization Decision Requests, send these requests to a PDP and enforce the decision of the PDP
- Policy Decision Points (PDP) that evaluate the policies in the context of a specific Authorization Decision Request and return an Authorization Decision.

XACML defines the language to express Authorization Decision Requests, Authorization Decisions and the policies created by the PAP. XACML and SAML are closely related: SAML assertions can be used in a XACML Authorization Decision Request to describe how a subject was authenticated and what attributes he has. The XACML OASIS TC recently has submitted a proposal to the SAML OASIS TC to include native XACML Request and Response contexts in SAML 2.0.

Specification	Version and status	Editor(s)	Standardization
XACML	1.0, OASIS Open Standard, February 18, 2003	Overxeer, Entrust, Sun, IBM, BEA, others	OASIS

Table 2-12: Overview of XACML specifications

2.5.5 Liberty Alliance Project

The Liberty Alliance Project [liberty] originally was an initiative of Sun Microsystems but has grown to an alliance of over 160 companies from all over the world, including educational, governmental and financial institutions, service providers, technology firms and wireless providers.

The objective of the project is to define an open standard for federated network identity and identity-based services that enable simplified sign-on to multiple domains or websites and support and promote permission-based attribute sharing to enable a user's choice and control over the use and disclosure of his/her personal identification. To accomplish this the alliance is divided into three collaborating expert groups:

- The Business & Marketing Expert Group takes care of public relations, market requirements and business templates for business adoption of the specifications.
- The Technology Expert Group creates the specifications en drives sample implementations and interoperability testing.
- The Public Policy Expert Group ensures that the Liberty specifications comply with laws and regulations and develops privacy best practice guidelines.

The specifications are released in several phases. Phase 1 (also called the Identity Federation Framework, ID-FF) is a set of specifications that provide an architecture for federated network identity (account sharing) and single sign on. Phase 2 (Identity Web Services Framework, ID-WSF) allows groups of trusted parties to link with other groups and will provide end users with the ability to control how their identity information is shared (permissions-based attribute sharing). Phase 3 (Identity Services Interface Specifications (ID-SIS) will build services on top of the phase 2 ID-WSF [fontana].

Liberty Alliance collaborates with existing standards groups like the OASIS Security Services TC responsible for the SAML specifications. In Liberty version 1.1 (phase 1) they extended SAML 1.0 to include additional security features for identity management. With the publication of the Liberty phase 2 draft specifications the phase 1 documents were submitted to OASIS to serve as input for SAML 2.0.

Another well-known identity management system is Microsoft's .NET Passport. Although Liberty in the beginning was set up by Sun to provide an alternative to Passport, efforts are now made to make sure Liberty and Passport can coexist and work together [roberts].

2.6 Web services in telecom

There are several organizations working on web services for telecom related applications.

2.6.1 Parlay Group

The Parlay Group [parlay] is a group of IT companies, software developers, network device vendors and operators, application service providers, etc. The goal of the Parlay Group is to define a set of open, technology independent application programming interfaces (APIs) for multi-vendor interoperability and rapid development of applications.

The Parlay Web Services Working Group is working on interface and infrastructure definitions for using web services within a telecom environment. The Parlay X Working Group defines highly abstracted web service interfaces for Parlay OSA (Open Service Access) services, thus exposing telecom capabilities through web services that are very simple to use by the IT community. These services include call control, SMS and multimedia message sending, payment, account management, user status and user location.

The Parlay Group has formed a Joint Working Group (JWG) together with the European Telecommunication Standard Initiative (ETSI) and the Third Generation Partnership Project (3GPP) to develop and maintain one uniform set of Parlay OSA APIs.

2.6.2 Open Mobile Alliance

The Open Mobile Alliance (OMA) [oma] is a global organization, set up by the mobile industry in June 2002 based on the WAP Forum and the Open Mobile Architecture initiative. Since then, a number of other organizations have integrated into OMA: the Location Interoperability Forum (LIF), SyncML (Synchronization Markup Language), MMS-IOP (Multimedia Messaging Interoperability Process), Wireless Village, Mobile Gaming Interoperability Forum (MGIF) and the Mobile Wireless Internet Forum (MWIF). The principles of OMA are [omaoverview]:

- Products and services are based on open, global standards, protocols and interfaces and are not locked to proprietary technologies.
- The applications layer is bearer agnostic (examples: GSM, GPRS, EDGE, CDMA, UMTS)
- The architecture framework and service enablers are independent of Operating Systems (OS)
- Applications and platforms are interoperable, providing seamless geographic and intergenerational roaming.

The OMA releases specifications in three phases to ensure standardization and interoperability between services, applications and devices [omarelease]:

- **Phase 1**: an approved set of open technical specifications forming an enabler that can be implemented in products and solutions and which can be tested for interoperability
- **Phase 2**: in addition to the open technical specification in phase 2, the enabler has successfully passed interoperability tests
- **Phase 3**: finally the OMA Interoperability Release is released when the enabler has passed end-to-end interoperability tests

The OMA Mobile Web Services Group develops a specification to enable the offering of services within the OMA framework to third parties using a web service interface. These services can be for example messaging (SMS, MMS), location services or accounting.

OMA is working on a number of open specifications for mobile services and works closely together with existing standards organizations and groups such as IETF, 3GPP, 3GPP2, W3C and JCP. OMA is trying to align its work with other initiatives like Parlay and Liberty.

2.6.3 3GPP

The Third Generation Partnership Project (3GPP) [3gpp] is a collaboration between a number of telecommunications standards bodies: ARIB, CWTS, ETSI, T1, TTA and TTC. The original scope of 3GPP was to produce a worldwide uniform standard for third generation mobile networks. This scope was later broadened to include the evolution of the second-generation GSM networks: GPRS and EDGE.

The 3GPP Multimedia Messaging Service [mms] architecture contains an interface for sending and receiving MMS's using SOAP (with attachments) over HTTP. This interface is called the MM7 reference point. Although this interface uses SOAP, there is no WSDL description defined, so one could argue if this is a real web service.

2.6.4 Ecma International

Ecma International is a not-for-profit industry association of technology developers, vendors and users that develops standards for information and communication technology (ICT) and consumer electronics.

The Ecma-348 standard (Web Services Description Language (WSDL) for CSTA Phase III, [ecma]) describes a WSDL for the XML messages defined in Ecma-323, XML Protocol for Computer Supported Telecommunications Applications (CSTA) Phase III.

With this CSTA WSDL developers can easily build applications that manage voice, instant messaging, SMS, paging and e-mail, in the same way, no matter what type of infrastructure they have.

2.7 Discussion

In the area of web services, very little is already standardized. The basic standards UDDI, WSDL and SOAP are well defined, while most of the additional specifications are still in the public draft stage, or even not even that far.

With so many companies working on web services, interoperability and overlap are some of the problems that arise. The problem of interoperability is handled by the WS-I. Overlap can be seen in a number of areas: business processes (BPEL4WS vs. WSCI), binary attachments in SOAP (SOAP-Attachments vs. DIME), reliable messaging (WS-Reliability vs. WS-ReliableMessaging vs. WS-Acknowledgement/WS-Callback/WS-MessageData vs. HTTPR) and transactions (WS-Transaction vs. BTP vs. WS-CAF).

Also, until specifications have been submitted to OASIS or another standardization body, intellectual property rights (IPR) and license claims can form a barrier for implementations and can be a threat to the worldwide adoption of some of the web services specifications.

So the basics are there, but the rest still needs a whole lot of work.

3 Service profiling

This chapter elaborates on the service profiling concept and provides an example for better understanding. Also a general service profiling architecture is introduced.

3.1 Introduction

When one business offers a service to another business, that service usually consists of an interface and a process. The interface describes how to communicate, and the process describes what is actually being done when the service is used. That process contains both business and technical aspects, like how the service is paid for and what should be done in case of a failure. Such a process is called a service profiling process. An offered service will therefore usually be composed of a choreography of several other services, together fulfilling the business and technical needs.

A service profiling process links several base and auxiliary services together. A base service is a service that implements the actual functionality of the offered service, while auxiliary services usually provide the business and technical requirements like charging, logging, privacy checks, etc.

Once a service provider has set up a library of base and auxiliary services, offering new services is only a matter of designing a new service profiling process and deploying the process on a service profiling platform.

Because the service profiling process is executed the moment the service is used, it is possible to adapt the service to some context the moment it is used, for example time of day, user status, user location, etc. These services are said to be context aware.

This assignment is limited to web services, but there are a lot of advantages when using web services. It is a standardized format and web service invocations can easily be routed over the Internet. That way it is very easy to offer a service that contains base or auxiliary services that the service provider itself does not offer. In this way a telecom operator can offer a service where a user can request a map with points of interest depending on his or her location, without the operator needing to buy a complete points of interest database or a database with maps. Already a large number of web services are available on the Internet.

3.2 Example

We will now introduce the example service profiling process SendSMS that will be used later on to build a prototype.

Consider a company that wants to send advertisement SMS messages to consumers. If consumers do not want the SMS messages, they can sign an opt-out form with their operator. If a message is sent, the operator has to check if the receiver of the SMS message has signed an opt-out form. Moreover, if the SMS could not be sent for some reason, the company does not have to pay for that SMS. The flowchart for such a service would look something like figure 3-1.

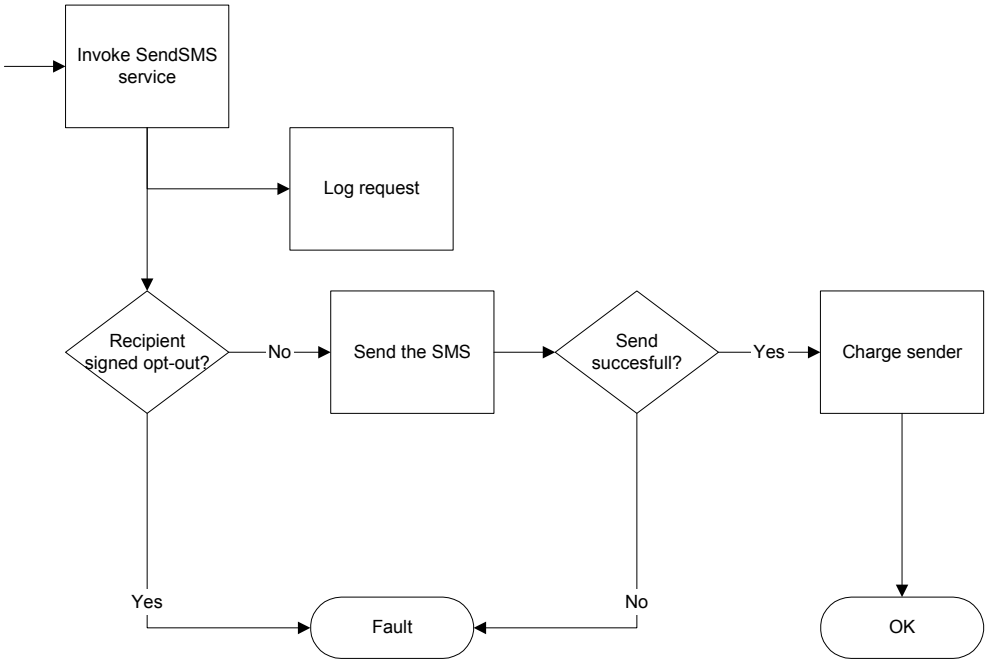

Figure 3-1: Flowchart of an example SendSMS service

The flowchart actually describes the service profiling process. In this example the base service is easily identified: "Send the SMS". Whether or not the sending of the SMS was successful can be handled by an exception handling mechanism, so that leaves us with the following auxiliary services: "Log request", "Receiver signed opt-out?" and "Charge sender". As you can see in the flowchart, the logging service can be invoked in parallel with the opt-out check to speed up the response time.

3.3 Architecture

The general service profiling architecture in figure 3-2 shows how the different entities in the service profiling process (base service, auxiliary services, service profiling process, web service consumer) are linked together.

The architecture shows two domains: the Internet and the provider domain. In between is a security gateway that handles authentication and authorization, so the provider domain can be seen as a secure environment in which the service profiling process executes. The Web Service Consumer (WSC) is the person or company that is using (consuming) the service. Since the base or auxiliary service do not need to be in the provider domain, one auxiliary service is placed in the Internet domain in this picture. The service profiling process is the central entity that links everything together.

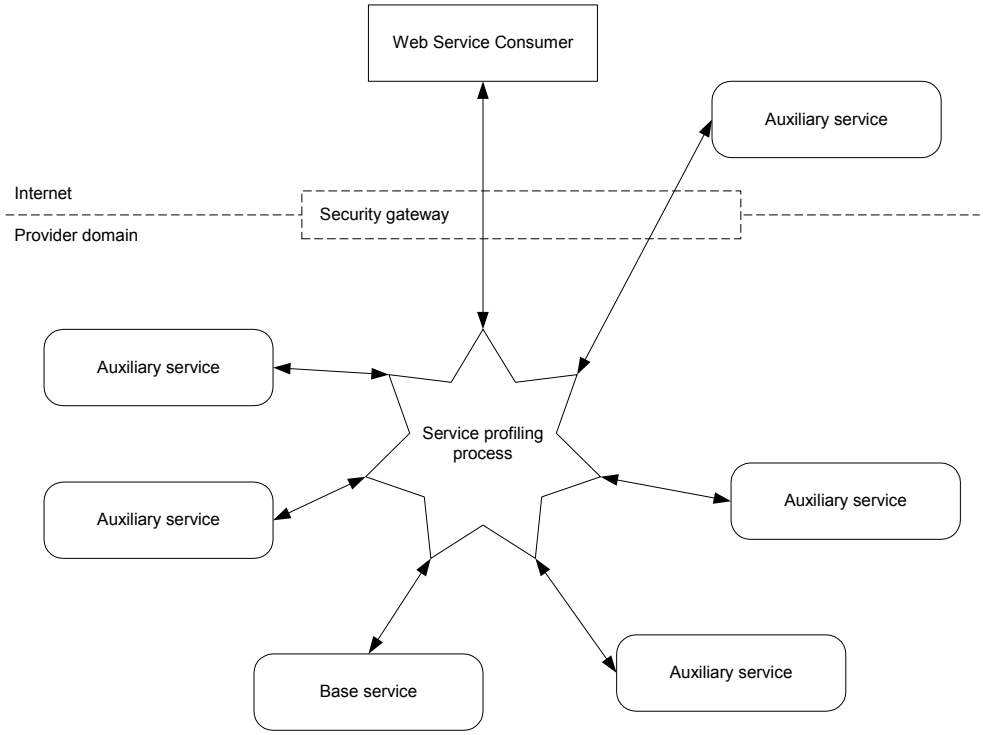

Figure 3-2: General service profiling architecture

All arrows represent web service invocations (usually SOAP over HTTP) while the rounded rectangles represent web services. The interface between the service profiling process and all other services and the WSC are web service interfaces, so the only thing the WSC sees is a web service interface and is not aware of the whole service profiling process.

4 BPEL

This chapter contains an introduction to BPEL and a discussion about the suitability for service profiling.

4.1 Introduction

The Business Process Execution Language for Web Services (BPEL4WS, or BPEL in short) is an XML-based "notation for specifying business process behavior based on Web Services" [bpel11]. A BPEL document describes a business process in the form of a series of web service invocations (choreography) that are executed in a specific order because of possible data or control dependencies (orchestration). BPEL provides support for error handling, data handling and message correlation. Because all invoked services are web services, other web service specifications or standards can provide additional features (like WS-Security for enhanced security).

BPEL originally started as an initiative from IBM, Microsoft and BEA. In August 2002, the first public draft specification of the “Business Process Execution Language for Web Services (BPEL4WS)" was released. The specification was based on the Web Services Flow Language (WSFL) from IBM and XLANG from Microsoft.

WSFL is "an XML language for the description of Web Services compositions" [wsfl] published by IBM in May 2001 to serve as an input to standardization initiatives in the corresponding area. The specification describes two types of compositions: usage patterns to describe business processes and interaction patterns as an overview of all partner interactions. XLANG is "a notation for the specification of message exchange behavior among participating web services" [xlang] published by Microsoft in May 2001.

On April 16, 2003, OASIS announces the forming of the new Web Service Business Process Execution Language TC to continue work on the business process language published in the BPEL4WS 1.0 specification. Just two weeks before, IBM, Microsoft, BEA, SAP and Siebel published a new version of the BPEL specification (version 1.1, dated March 31, 2003), incorporating numerous corrections and clarifications based on the feedback received on the 1.0 version. This BPEL specification was intended to be submitted by the authors at the first meeting of the TC. Instead of that version, an updated BPEL 1.1 specification dated May 5, 2003 was submitted as a starting point for the TC.

Several implementations exist today. IBM offers an open source BPEL engine running on a J2EE application server called BPWS4J [bpws4j]. The Collaxa BPEL Server is a commercial product based on the JBoss application server and the Axis SOAP implementation [collaxa]. The next version of Microsoft BizTalk application integration server will also include BPEL support and is based on the Microsoft .NET platform.

4.2 How it works

The BPEL specification depends on a number of other XML and web service related standards:

- WSDL 1.1 for describing the choreographed and exposed web services and for mapping abstract port types to real service endpoint addresses.
- XML Schema 1.0 for data types etc.
- XPath 1.0 as XML query language. It is possible to use other XML query languages, but support for XPath 1.0 is mandatory for BPEL compliance.

4.2.1 Partners and partner links

A BPEL process definition begins with a list of partner links and variables. Then the actual process follows as a series of web service invocations and operations on variables.

Conversations between the BPEL process and a web service of a partner are always done through a partner link. The web service consumer is also seen as a partner. A partner link is of a partner link type, which describes the conversational relationship between two services by defining "roles" played by each of the services in the conversation and specifying the WSDL port type for each service to receive messages.

The process can have more than one role. Each role specifies exactly one WSDL port type. It is possible to have multiple roles on one partner link. Multiple partner links to one partner is also possible.

In a process, web services are referenced in an abstract way by a partner link, WSDL port type and WSDL operation. Each invocation has a name and specifies an input and/or output variable.

4.2.2 Variables

There are two types of variables in a BPEL process: message properties and variables.

A message property can hold information about both the context of a message (headers) or the data in the message (body) and can both be part of business protocols (for example, correlation tokens) or infrastructure protocols (for security, transaction, reliable messaging, etc.). Message properties are defined using XML Schema simple type definitions. Message properties are linked to a part of a message using a property alias, specified as an XPath query.

Variables are used in a process to keep track of some state in the process or to contain messages that are to be sent to or are received from partners. The type of a variable may be a WSDL message, an XML Schema simple type or an XML Schema element.

4.2.3 Correlation

Sometimes multiple web service invocations need to be correlated so that the right instance of the service is invoked, or asynchronous messages are routed to the right instance of the business process. Usually, correlation tokens (like a customer ID, order ID or transaction ID) are inserted into a message. Since there may be different correlation tokens in one process, a message may contain more than one correlation token. Correlation sets specify correlated groups of operations within a service instance. A set of correlation tokens is defined as a set of properties shared by all messages in the correlated group. The tokens are automatically inserted into the header or body of the SOAP message, if not already present.

4.2.4 Web service activities

There are three basic activities for web service operations: *invoke*, *receive* and *reply*. An invoke activity performs a WSDL operation on a WSDL port type over a partner link using an input and possibly an output message. The receive activity waits for an operation to be received at a port type over a partner link. After the receive activity took place a reply may be sent using the same operation, port type and partner link.

The assign activity can be used to copy (a part of the) data from one variable to another or insert new data using expressions.

Other activities are the *throw*, *wait* and *empty* activities. If an activity fails for some reason, a fault can be caught and handled. The throw activity can be used to explicitly signal a fault. The wait activity is for waiting until a certain point in time or for a certain period, and an empty activity can be used to suppress errors for example.

In an activity, a compensation handler can be specified. A compensation handler is an activity that compensates for another activity. These handlers are mainly used in fault handling for cleaning up and a roll back of transactions.

4.2.5 Structured activities

Structured activities prescribe the order in which a collection of activities takes place. There are five different activities: *sequence*, *switch*, *while*, *pick* and *flow*.

Sequential control is provided by sequence, switch and while. A sequence activity contains a list of activities that are performed sequentially. If the last activity of the sequence has completed, the sequence is also complete. A switch activity consists of a list of conditional branches, and an optional 'otherwise' branch. Only the first branch whose condition is true is executed, or if no branch has a true condition the 'otherwise' branch is executed. All activities contained in a while activity are repeatedly executed until the while condition is false.

The pick activity is a set of events (either the reception of a message or an 'alarm' based on a timer) where only the event that occurs first is executed. If that one event is completed, the whole pick activity is complete.

Concurrency and synchronization between activities is provided by the flow activity. All activities in a flow activity are executed at the same time (concurrent). A flow completes when all activities in that flow have completed (either successful or not). Synchronization between concurrent activities is done by links. A link has a source activity and a target activity where the target activity can only be executed when the source activity is complete.

The source of a link can also specify a transition condition that has to evaluate to true before the target activity can start. Likewise, the target of a link can specify a join condition that has to evaluate to true and may contain an expression with variables and has to take the link status into account.

An activity that has one or more synchronization dependencies (i.e. is the target in one or more links) only starts if the transition conditions and join conditions of all incoming links evaluate to 'true'

4.2.6 Fault handling

The behavior context for an activity is provided by a *scope*. A scope can provide fault handlers, event handlers, a compensation handler, data variables and correlation sets.

A compensation handler is an activity that compensates for the activity within the scope if for example a fault is caught and the result of the activity needs to be undone. A compensation handler can be invoked explicitly by using the *compensate* activity (for example in a fault handler), or implicitly if the compensation handler of an enclosing scope is executed.

Fault handlers are executed when an exception is thrown, by a *throw* activity or by a failure in an invoke activity for example. A fault handler contains *catch* handlers that can catch a specific fault, or all faults if no fault is specified. The *catchAll* handler catches any fault that is not caught by a more specific catch handler. A catch handler may invoke a compensation handler, invoke some web service, rethrow the fault to a higher-level scope, terminate the process, or just do nothing.

4.2.7 Event handlers

Two types of event handlers can be associated with a scope: *onMessage* handlers that fire on the reception of a specific message or *onAlarm* handlers that fire after a certain time after the scope is activated or at a certain point in time. The event handlers are active as long as the scope they belong to is active. Multiple messages can be handled by the event handlers concurrently and multiple times. An alarm handler can only be activated once and is disabled afterwards. An event cannot instantiate a process instance, but can instantiate a correlation set.

4.3 IPR and licensing

When the original developers of the BPEL specification (IBM, Microsoft, BEA Systems, SAP AG and Siebel Systems) submitted BPEL to Oasis, they made the following statement concerning intellectual property rights when implementing the BPEL specification:

Each Author commits to grant a non sub-licenseable, non-transferable license to third parties, under royalty-free and other reasonable and non-discriminatory terms and conditions, to certain of their respective patent claims that such Author deems necessary to implement required portions of the BPEL Specification, provided a reciprocal license is granted. http://www.oasis-open.org/committees/wsbpel/ipr.php

BEA, IBM and Microsoft have submitted additional statements to OASIS. BEA states to have no patent rights in BPEL, but will provide the above mentioned license if necessary. IBM believes that they hold several published and unpublished patents that may be essential to compliant implementations of the BPEL specification, and will provide the above-mentioned license.

Microsoft defines additional terms for providing a royalty-free license: object code versions of a BPEL implementation may only be distributed *"…incorporated into Company Products and solely for the purpose of complying with BPEL4WS."* [bpellicense], while the source code may only be distributed if the following notice will be prominently displayed in all copies and in the license agreement of that source code:

"This source code may incorporate intellectual property owned by Microsoft Corporation. Our provision of this source code does not include any licenses or any other rights to you under any Microsoft intellectual property. If you would like a license from Microsoft (e.g. rebrand, redistribute), you need to contact Microsoft directly." [bpellicense].

4.4 Implementations

I found two BPEL implementations on the J2EE platform that are available for download on the Internet: the Collaxa BPEL Server and IBM implementation BPWS4J. There are also already several graphical editors available to design BPEL processes: plug-ins for Eclipse (IBM, BP Wizard Software) and LTSA, VisualScript XML and Collaxa BPEL Designer.

4.5 Architecture

The architecture for the service profiling process would not be very different from the general architecture in chapter 3. The BPEL engine can just replace the general 'service profiling process'. All arrows represent web service invocations.

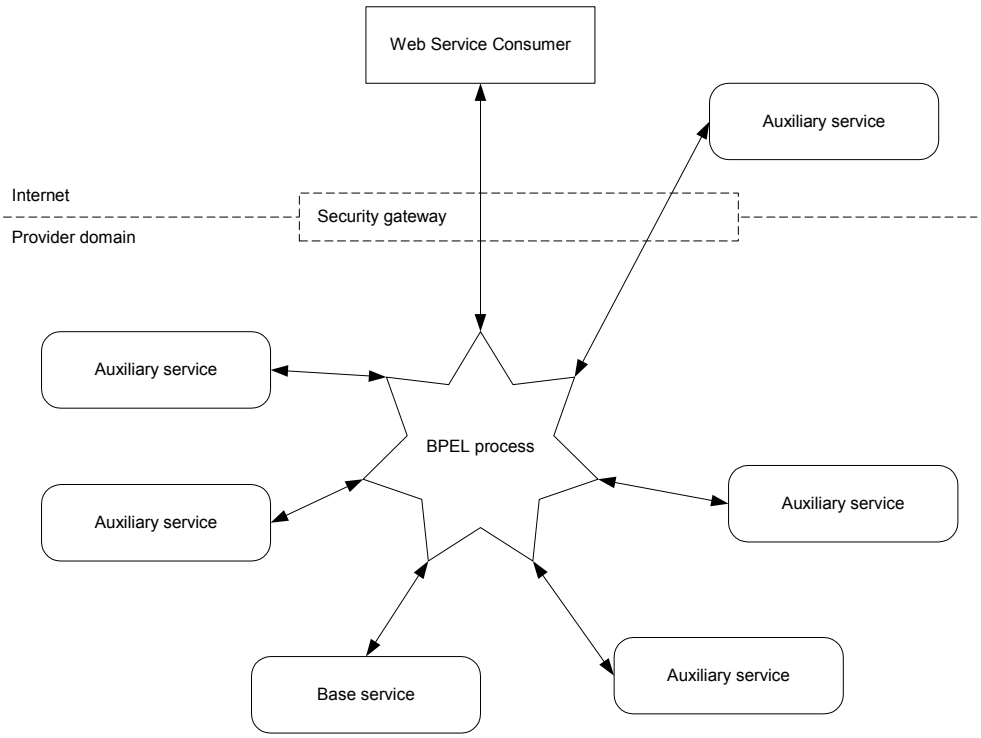

Figure 4-1: Service profiling architecture using BPEL

4.6 Discussion

The structured activities in BPEL are very well suited for (online) choreography and orchestration. Transactions within the process are supported through the use of fault and compensation handlers. The service consumer is totally unaware of the process, since the process itself has a web service interface described in standard WSDL.

The BPEL implementations in the Collaxa BPEL Server and IBM BPWS4J offer tools for easy process deployment and undeployment. Graphical editors greatly ease the creation of BPEL processes.

Performance could be an issue, since all auxiliary services must have a web service interface, thus introducing extra XML and SOAP encoding overhead. Moreover, BPEL was designed with relatively long lasting transactions in mind, and is thus not optimized for performance.

BPEL is backed by a great majority of industry leaders, like Microsoft, IBM, Sun Microsystems, BEA, HP, Oracle, SAP, and many others. Although Microsoft has posed additional licensing terms, this should not be a problem, especially when using an existing BPEL implementation.

5 WSCI

This chapter contains an introduction to WSCI and a discussion about the suitability for service profiling.

5.1 Introduction

The Web Service Choreography Interface (WSCI) is an "XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services" [wsci]. WSCI is an extension to WSDL. While WSDL describes the static interface of a web service to perform one operation on that service, WSCI can be used to describe the dynamic interface of a web service by describing the relationship between multiple operations in the context of multiple message exchanges.

BEA Systems, Intalio, SAP and Sun Microsystems released the WSCI specification in June 2002. In August 2002 the specification was submitted to W3C as a royalty-free specification, where it got the "W3C Note" status. This specification served as the one of the principal input documents for the W3C Web Services Choreography Working Group.

5.2 How it works

WSCI describes an *interface* that contains one or more *processes*. A web service may expose multiple interfaces for supporting multiple scenarios or multiple views on the same scenario (for example for different actors).

5.2.1 Variables

Properties are the variables of WSCI. Properties are name-value pairs that can contain entire messages, a reference to part of a message, or just some value. All incoming and outgoing messages are automatically defined as properties. Properties have a global scope, unless they are defined as local properties to some scope.

The (top-level) selector element defines how property values are extracted from incoming messages using an XPath query. If no XPath query is specified and no value is specified, the whole message is assigned to the property.

5.2.2 Context and correlation

A *context* is the environment in which an activity is executed. Several activities can share one context. A context contains a set of local properties and processes that are available to the activity, and information about exception handling and transactions.

For defining correlations, the correlation element is used. Each correlation consists of a unique name, a list of properties used to correlate messages, and optionally an *extends* attribute if the correlation is the extension of some base correlation. The *correlate* element is used to associate an action with the correlation definition, using the unique name. This element is also used to specify when a correlation has to be instantiated.

5.2.3 Exceptions and transactions

The kinds of exceptions that can be caught are the receipt of a particular message that is considered an exception in that context, the receipt of a WSDL fault message, a fault generated by the service itself or a timeout. If an exception is caught, the exception handler is executed and only the current context terminates, not the entire process. Uncaught exceptions are raised in the parent context.

If a context is associated with a transaction, the activities within that context are executed in an all-or-nothing manner. Compensation activities can be declared to roll back an activity if the transaction has completed successfully, but needs to be undone. If a transaction contains other (sub) transactions, those sub transactions are rolled back first (recursively) in reverse order of completion. WSCI supports two types of transactions: atomic transactions that are of short duration and require resource locking, and open transactions where the transaction progresses from one consistent state to another.

5.2.4 Processes

A process is a (portion of) behavior that can be reused in another process by instantiating it via the receipt of a message, explicit calling, or from within the service implementation (not shown in the interface). There are two types of processes: top-level processes that are defined at the interface level and can be referenced from everywhere within the interface, and nested processes that are defined within a complex activity and can only be referenced from within that activity.

The behavior of a process is described as a set of choreographed activities. This may either be atomic (WSDL) activities, or complex activities recursively composed of multiple activities. For describing choreography of activities WSCI supports sequential and parallel execution, looping and conditional execution.

Processes can be reused by using a *call* or a *spawn* activity. A call activity instantiates a process and waits for it to complete, a spawn activity instantiates a process and completes immediately. The *join* activity waits for all instances of a spawned process to complete.

5.2.5 Activities

A process contains a list of activities. There are two types of activities: atomic activities, or complex activities that consist of multiple atomic and/or complex activities. An atomic activity is the *action* activity that performs a WSDL operation. Below are the complex activities that are used for choreography: they define which activities are executed and in what order.

- *All*: performs all activities within this complex activity in a non-sequential order, possibly in parallel.
- *Choice*: performs only one activity set based on the first event triggered. Possible events are the receipt of a message, a timeout or a fault.
- *Foreach*: performs all activities in the activity set repeatedly, once for each item in the selected list.

- *Sequence*: performs all activities in the activity set in sequential order.
- *Switch*: selects one activity set based on the evaluation of one or more conditions.
- *Until*: performs all activities in the activity set repeatedly (one or more times) until the condition evaluates to false.
- *While*: same as until, but zero or more times.
- *Delay*: delays execution for a certain period or until a certain point in time.
- *Empty*: does nothing.
- *Fault*: triggers a fault in the current context.

Before complex activities can execute, a context has to be initiated.

5.2.6 Global Model

A WSCI interface describes the view of the overall message exchange as seen from only one participant. The WSCI *Global Model* makes it possible to describe a multi-participant view of the overall message exchange. The Global Model consists of a list of interfaces of the participating services and links between operations on those services.

5.3 IPR and licensing

When the W3C members BEA Systems, BPMI.org, Commerce One, Fujitsu Limited, Intalio, IONA, Oracle Corporation, SAP AG, SeeBeyond Technology Corporation and Sun Microsystems submitted the WSCI specification to the W3C, they all included an IPR statement in the submission request [wsci-ipr]. Each company declared to grant a royalty free license to any essential claims necessary to implement WSCI.

5.4 Architecture

The architecture is somewhat different from the general architecture in chapter 3. The process definition is not the central entity anymore, but is described as a WSCI interface. A WSCI proxy executes the actual process and serves as an abstraction for the web service consumer that does not need to have WSCI support in this case.

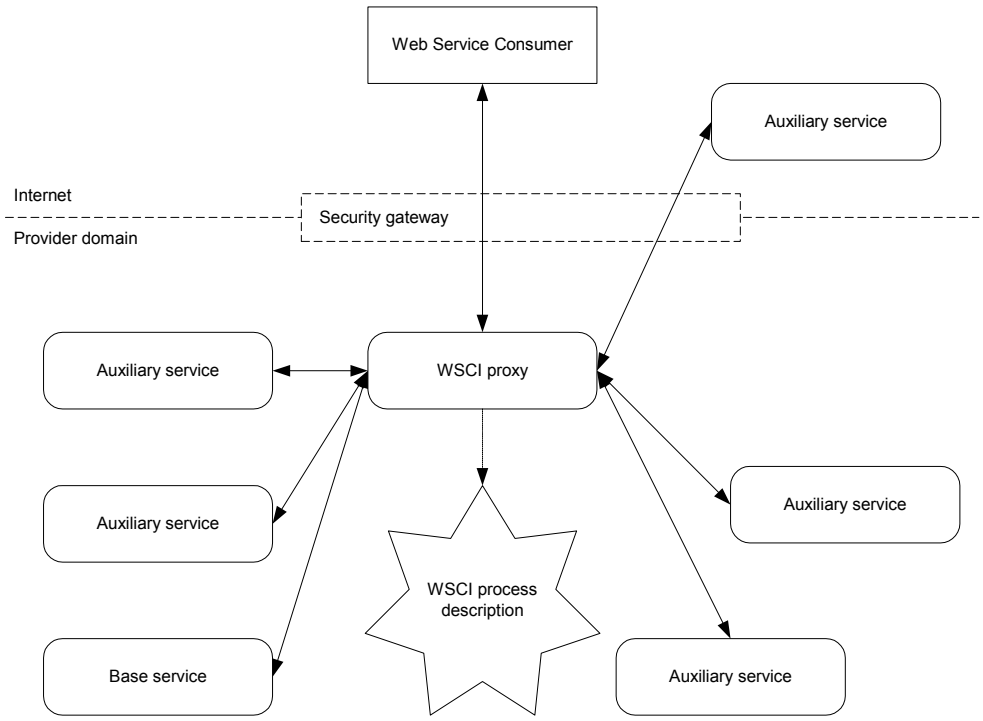

Figure 5-1: Service profiling architecture using WSCI

5.5 Discussion

The concept of activities in WSCI makes (online) choreography and orchestration possible. Transactions are supported through the use of exception handlers and compensation activities.

The WSCI interface concept makes that the web service consumer is not unaware of the process. One possible solution is to use a "WSCI proxy" that inspects the WSCI interface of the base service and executes the process.

Service lifecycle management can be done through regular web service deployment tools, since the process description can be embedded in the WSDL description of a web service. Variations of one service can be made by creating more than one WSDL document per (base) service.

Performance could be an issue here since all auxiliary services need to have a web service interface, just like BPEL.

Although the specification can be implemented royalty free, there are no (publicly available) WSCI implementations yet. Also, the standard lacks backing from several large IT companies like Microsoft and IBM.

6 Axis

This chapter describes the details of Axis and will discuss the suitability for service profiling.

6.1 Introduction

The Apache Software Foundation (http://www.apache.org/) provides support to a range of open-source software projects. Apache Axis [axis] is one of those projects, in which volunteers work on an open-source implementation of the SOAP 1.1 (and a large part of the 1.2) specification submitted to W3C [soap]. Axis is an application that runs on a java application server or servlet engine. Axis developers prefer the Jakarta Tomcat server (the official reference implementation for the java servlet and java server pages technologies), but Axis runs perfectly well on other J2EE application servers like IBM Web Sphere, JBoss, Sun One, WebLogic, etc.

Axis comes with an integration guide that describes how to integrate Axis into your own Java project. Axis is used in the Collaxa BPEL Server, Apple's WebObjects, Borland Enterprise Server, Borland JBuilder, JBoss Application Server, IBM's Web Services Toolkit, Macromedia's ColdFusion MX, and many others.

6.2 How it works

Axis makes methods of standard Java objects accessible via SOAP without additional programming. Axis generates WSDL automatically and takes care of the SOAP encoding and decoding. Axis can act both as a client and a server. If Axis is used as a server, there are two ways to deploy a web service:

- Rename a .java file to .jws and drop it in the Axis directory on the application server. All methods are accessible via SOAP and the WSDL is automatically generated. This way of deploying web services is very simple but also not very configurable and does not support Java packages. Also, you need the source code of the deployed service.
- Axis has a Web Service Deployment Descriptor (WSDD) file format that can be used to specify which methods are allowed to be exposed as web services and to specify handlers or chains (see below).

There are also two ways to use Axis to invoke a web service:

- Use the WSDL2Java tool to generate Java code that allows a programmer to call web services as if it were local Java methods.
- Directly use the web service client classes of Axis.

When a request arrives at Axis, a *Message Context* is created and the message is placed in that context. Then the message will follow a message path that consists of a request flow, the processing of the request, and possibly a response flow. These flows contain *Handlers*. Handlers are Java objects that can modify the message and it's message context. A *Chain* is a special handler that contains a sequence of handlers. Chains are constructed offline and cannot be altered when they are deployed. Handlers and chains can be defined to have one instance that handles all messages (singleton scope) or to have an instance for every request.

If a fault occurs, all handlers prior to the one that raised the fault are invoked in reverse order to handle the fault.

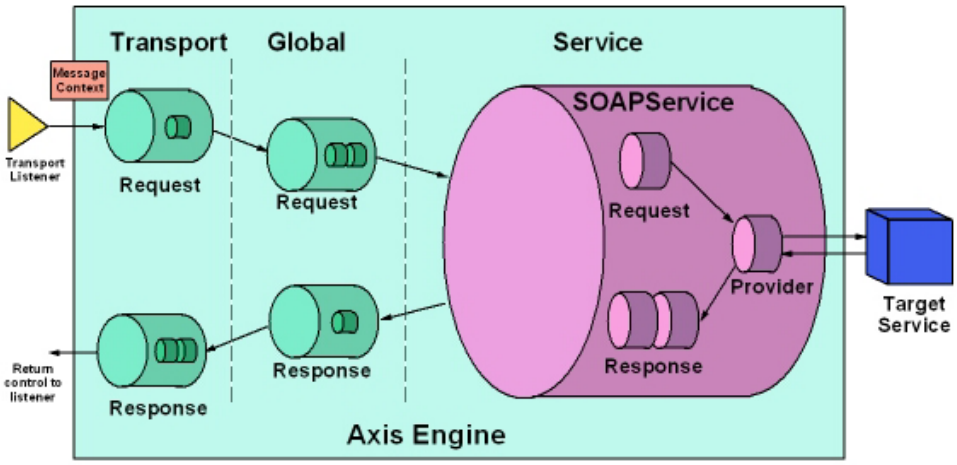

Figure 6-1: Message path in Axis as a server (Axis Architecture Guide [axis])

If Axis is used as a server, the transport request and global request chains are invoked if a request arrives. One of these chains contains a handler that sets the service handler property in the message context. This allows the Axis engine to select the right service for the request. A service consists of a request and response chain, and a provider. The provider is a handler responsible for invoking the actual back end logic of the service. There are several providers available for different service styles offering increased automation, from a messaging service that only gives access to the raw XML data of the exchanged messages to the RPC service that uses SOAP RPC conventions and performs SOAP encoding and XML-Java data binding.

The response message follows the response message path from the provider via the service, global and transport chains.

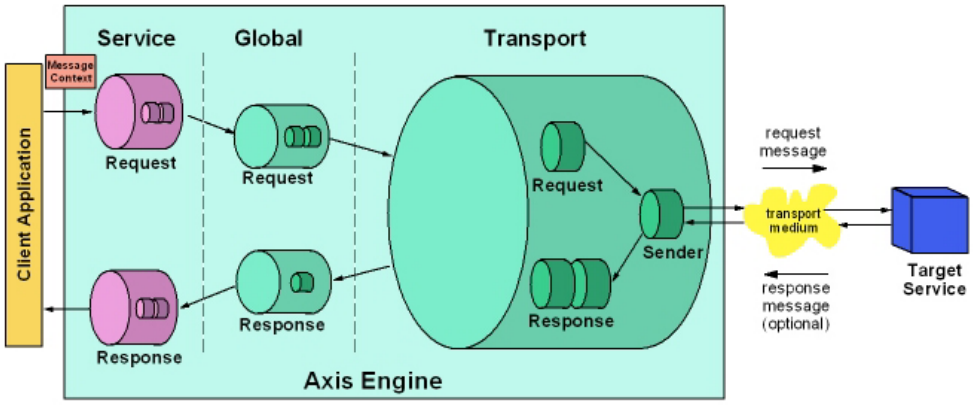

Figure 6-2: Message path in Axis as a client (Axis Architecture Guide [axis])

The message path when Axis is acting as a client is almost the same, only the order of the chains is reversed: first the service chain, then the global chain and then the transport chain are invoked for the request. The response message follows the same path backwards.

6.3 IPR and licensing

Axis is open source and is released under the Apache Software License, Version 1.1 [axislicense]. This license states that copyright notices must be retained and that documentation must include the acknowledgement "*This product includes software developed by the Apache Software Foundation (http://www.apache.org/).*". Furthermore, the *Apache* and *Xerces* names may not be used for promotions or as a part of the name of the software.

6.4 Architecture

The service profiling architecture is totally different from the general architecture in chapter 3 when using Axis handlers. There is no central process engine, and services are only executed in chains. Only interactions outside of the web service provider are web service interactions (solid lines), all others are Java based interactions (dotted lines). The process runs in a Java environment, not in a web services environment. This means that all service invocations are Java function calls rather than web service invocations. If there is an auxiliary service in the chain that has a web service interface, then a service proxy that takes care of the web service invocation (via the Axis client classes) needs to be used.

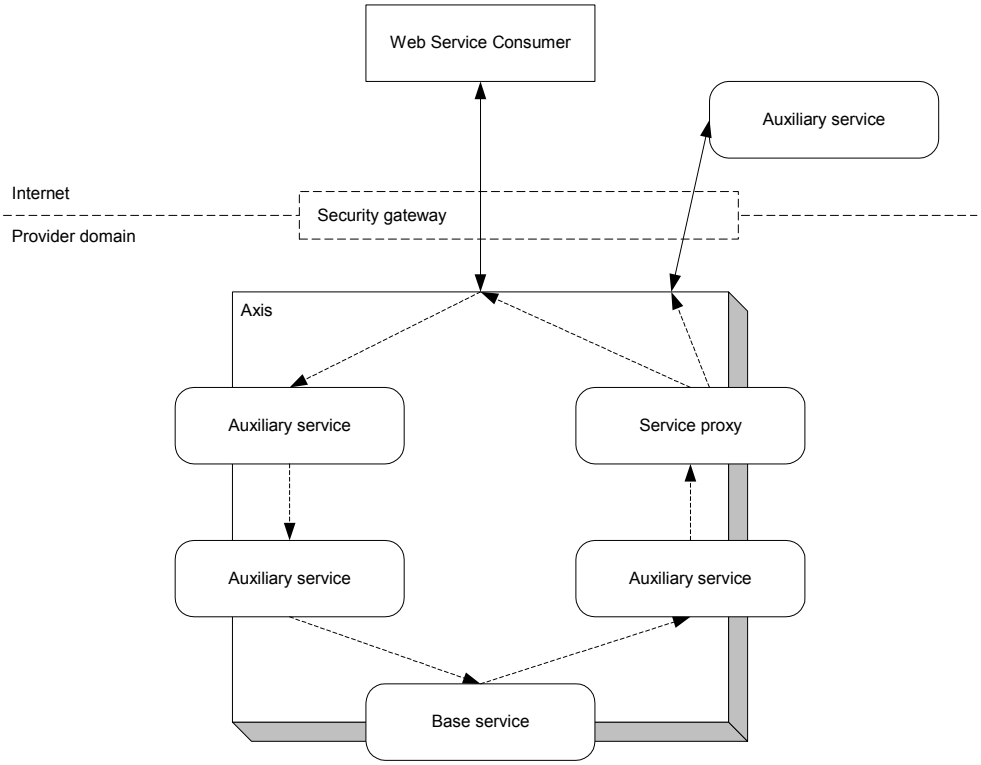

Figure 6-3: Service profiling architecture using Axis handlers

6.5 Discussion

The concept of a chain of handlers in Axis makes choreography possible. Orchestration is not possible because data or control dependencies cannot be realized since services in the chain have a predefined order. This also prevents online choreography. Transactions are supported to a very limited extent because fault messages travel through the chain backwards, which makes a rollback possible.

Deploying and undeploying services is easy with the deployment tools provided with Axis.

The web service consumer is unaware of the chain of handlers since the chain is configured when the web service is deployed. Because all communication between handlers are pure Java function calls, performance is not an issue.

Since Axis is already being used in a number of commercial and non-commercial products, there is a large industry support for Axis.

7 Java

This chapter will introduce a proprietary Java solution and discuss the suitability for service profiling.

7.1 Introduction

Instead of using a standard technology, there is always the possibility to make a proprietary solution, in this case on the J2EE platform. Usually, a J2EE application server with access to network capabilities is already present in a telecom network. This way the auxiliary services do not need to have (but can have!) a web service interface. This can have a positive impact on security and performance.

The requirements are already described in chapter 1. Because of its good and widely used SOAP implementation, Axis will be used for handling the incoming and outgoing SOAP messages.

Since we will need a way to express a process description, we need a language to define the choreography. The best way is to use an industry standard scripting language. David Kearns has written an article [kearns] in which different aspects of integration of a scripting language in a Java application is discussed. Possible scripting languages are for example Tcl, Python, JavaScript, or even Java via BeanShell. Performance will be a special point of interest when choosing a particular scripting engine. According to [kearns] performance differences can be up to a factor 100 between different engines.

The creation of a process will still be the task of a programmer, unless a graphical editor is built that provides a drag-and-drop way to create or manage a process.

7.2 How it works

When a SOAP message arrives at Axis, the information is translated to the Java domain and serves as the starting point for the service profiling environment to start a process. This environment serves as the environment in which the process executes by means of the scripting engine. The environment takes care that the invocation of the process script is routed to the right auxiliary services, either directly or via Axis.

Transaction support is limited to the possibility of the scripting language to catch faults and construct compensation procedures.

The environment also has to take care of correlation to make sure messages are routed to the right process instance.

7.3 IPR and licensing

When using a proprietary Java solution, there still are IPR and licensing issues to look into. For IPR and licensing of Axis, please refer to chapter 6.3. For other existing implementations and/or standards that are used in this solution there might be IPR and licensing issues, for example for the scripting engine that is being used.

7.4 Architecture

For the SOAP encoding and decoding, Axis is used since it is a widely used solution. All communication after Axis is done in Java for performance and flexibility reasons. Invocations from the process to (external) web services are routed through Axis for SOAP encoding. The dotted lines in figure 7-1 represent Java invocations, while the solid lines represent SOAP interactions.

The scripting engine with the process description runs in a service profiling environment that takes care of the actual invocations of the auxiliary services and has a management interface for deploying and undeploying services.

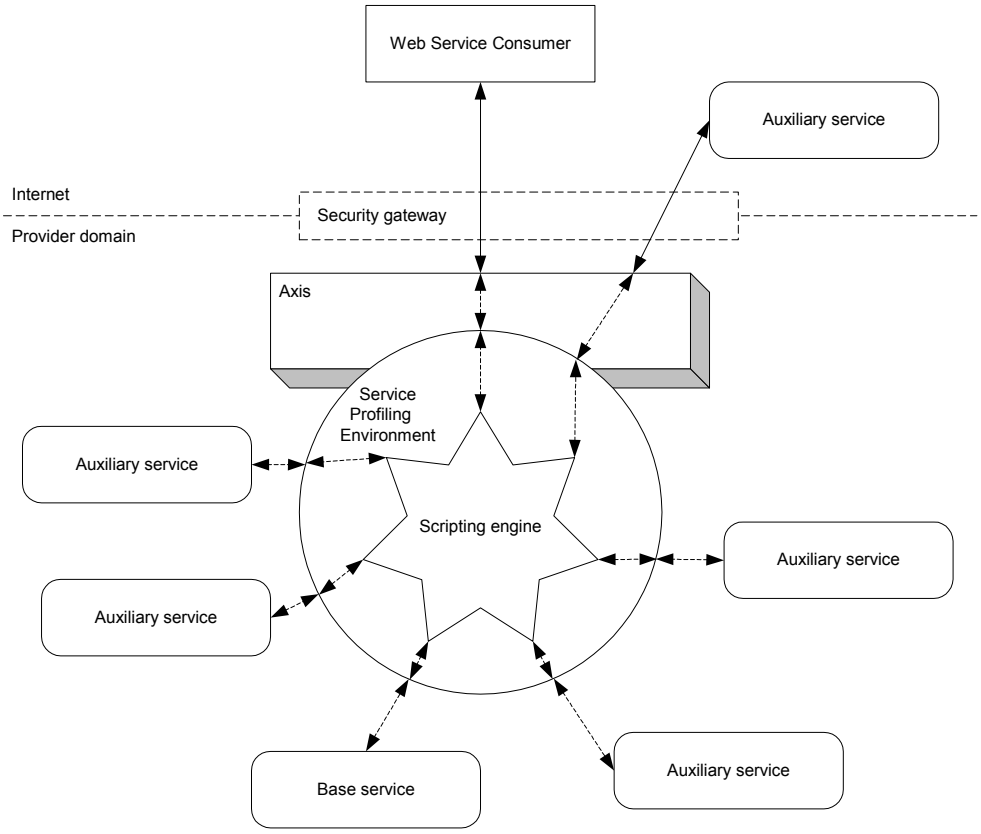

Figure 7-1: Service profiling architecture using a proprietary Java solution

7.5 Discussion

Choreography and orchestration are supported via the scripting engine. This also enables online choreography and orchestration. Transactions are only supported to the level the scripting engine supports exception handling.

A management tool for creating and deploying services has to be custom made. Since the process description is not a standard process description, there are no standard tools available for process design.

Since the invocations of the auxiliary services are Java, there should not be a performance issue there. The performance of the scripting engine will play a vital role in the performance of the overall system.

8 Comparison

This chapter provides a comparison between the four discussed technologies BPEL, WSCI, Axis and Java. The technologies are compared using the criteria mentioned in section 1.3.

8.1 Introduction

In section 1.3, a number of functional and non-functional criteria are defined. These criteria are the aspects the four discussed technologies BPEL (chapter 4), WSCI (chapter 5), Axis (chapter 6) and Java (chapter 7) will be compared on. Each of these last four chapters discusses a technology and points out some of the aspects. This chapter takes each aspect apart and discusses the four technologies per aspect.

When comparing these technologies using the aspects mentioned in chapter 1, not all aspects are equally important. The differences in "weight" per aspect can be expressed into weight factors. A factor of 1 means "average important", 2 means "more than average important" and 3 means "very important". Tables 8-1 and 8-2 give an overview of the weight factors. The reason why a certain aspect has a higher weight factor than others is discussed in the section discussing that particular aspect.

Functional aspect	Weight factor
Choreography	3
Orchestration	1
Transactions	2
Service lifecycle management	1
Online choreography and orchestration	2
Process unawareness	3

Table 8-1: Weight factors for the functional aspects used for comparison

Non-functional aspect	Weight factor
Performance	1
Implementation availability	2
Industry support	3

Table 8-2: Weight factors for the non-functional aspects used for comparison

8.2 Functional aspects

The functional aspects form the basis of service profiling. If support for a certain aspect is not present in a certain technology, that technology does not support the full service profiling concept. However, that does not mean that the technology is not suitable for a certain service profiling application.

8.2.1 Choreography

Support for choreography means that the technology can be used to choreograph several service together. Since this is the basic idea of service profiling, this aspect has a weight factor of 3. All four technologies support choreography.

8.2.2 Orchestration

Orchestration means that the order in which services are invoked can be dependent on certain data or control directives. The execution path in a process might for instance be different depending on the result a service returns or some other information (time, identity of the WSC). But since this might not always be a requirement in an application of service profiling, this aspect only has a weight factor of 1.

BPEL, WSCI and the Java solution all support orchestration because the process is described using a scripting (-like) language that supports conditional actions (if statements, switch blocks, etc.). Axis is the only one that does not support orchestration since all services are invoked one after another in a chain.

8.2.3 Transactions

Transactions are used to recover from faults and exceptions in a predefined way so the service profiling environment remains in a stable state and if necessary the WSC can be provided with a normal fault message. Transactions are also used to negotiate one common outcome using multiple services. For this last aspect, the WS-Transaction specification can be used. Since this aspect is important but not vital, it has a weight factor of 2.

Both BPEL and WSCI have support for fault handlers and transactions. Axis does not fully support fault handlers since if a fault occurs, all handlers prior to the one that raised the fault are invoked in reverse order to handle the fault. The Java solution does have limited exception handling, but no (built in) support for transactions.

8.2.4 Service lifecycle management

Service lifecycle management includes the whole process of deploying a service, phasing out, undeploying and versioning. The details of a service lifecycle are implementation specific, but a general comparison about how to deploy different services using the same base and auxiliary services can be made. Because exact details are implementation specific, this aspect only has weight factor 1.

Designing a service in BPEL simply means writing a BPEL process that uses existing web services or other BPEL processes (since these are also web services). With WSCI this is the same, just write a new WSDL. In Axis a new deployment descriptor with a new chain of services has to be made. The Java solution also requires writing and deploying a new process script.

8.2.5 Online choreography and orchestration

Online choreography and orchestration means that the actual execution path in a process is determined at execution time, not at compile or deployment time. This allows services to be context aware for example. This aspect is not a core aspect of service profiling and might for some applications not even be desirable for performance reasons. That is why this aspect has a weight factor of 2.

Axis is the only technology that does not support online choreography and orchestration, since the chain of services is determined when the service is deployed.

8.2.6 Process unawareness

For easy integration with other services and take advantage of available web services tools, the web service consumer must not be aware of the whole service profiling process behind the service. This is especially important for standardized interfaces like web services to enable interoperability. Since this is an important aspect for both technical and business reasons, this aspect has a weight factor of 3.

All technologies are invocable as a normal web service, except WSCI. But since the web service consumer must not have direct influence in the process, a WSCI proxy that executes the process is probably a good idea.

8.3 Non-functional aspects

The non-functional aspects do not actually say anything about the technical suitability of a certain technique, but if it is wise to use the technique from technical and business perspective.

8.3.1 Performance

The actual performance is dependent on the implementation of course, but a high level comparison can be made. This aspect has a weight factor of 1.

Executing a script requires interpretation of that script. Since that is a performance wise "expensive" task, both the Collaxa BPEL server and IBM BPWS4J compile the script at deployment time to enhance performance. WSCI and the Java solution can also be implemented that way. Axis saves the deployment descriptor that contains the handler configuration as an XML file that can easily be parsed since it only is a chain. Axis and Java have the advantage that pure Java calls can be used in the process. Java calls are usually faster than web service invocations since they do not have the whole web services protocol and conversion overhead. The Collaxa BPEL server also has a feature for Java calls, but that is not part of the BPEL specification.

8.3.2 Implementation availability

If there is an implementation available of a technique not only speeds up the design of a service profiling environment, but is also a proof the technique can actually be implemented and can be used. This aspect has a weight factor of 2.

This report already mentioned two publicly available BPEL implementations: the Collaxa BPEL server and IBM BPWS4J. Since Axis is the SOAP implementation of the Apache group, this is also available. At the time of writing, no WSCI implementations are (publicly) known. And since the Java solution is proprietary, no implementation is available, although there are implementations of scripting environments available.

8.3.3 Industry support

Industry support can be very important for a service profiling environment. Not only may an operator ask for support of certain technology to describe a process, but also inexpensive general-purpose tools for designing and maintaining processes might come to market. Also, operator personnel might already be familiar with the way a process is described. That is why this aspect has a weight of 3.

Industry support for BPEL is huge. Most major IT companies are member of the OASIS WSBPEL technical committee. There are fewer companies that back WSCI, and most of them also back BPEL. Axis is considered as the main SOAP implementation and is used in a variety of products. Industry support for a proprietary solution is of course minimal, unless you are able to make it a de-facto standard. But that is very hard to do.

8.4 Overview

To find out what technique is the "winner" of this comparison, they all get a certain score for each aspect, where a score of 0 means "no support" or "bad", 2 means "full support" or "very good", and 1 is something in between. The total score is calculated by multiplying the score with the weight factor and add up the score for each aspect. Table 8-3 shows the scoreboard for the functional aspects, while table 8-4 shows the same for the non-functional aspects.

Functional aspect	WF	BPEL	WSCI	Axis	Java
Choreography	3	2	2	2	2
Orchestration	1	2	2	0	2
Transactions	2	2	2	1	1
Service lifecycle management	1	2	2	2	2
Online choreography and orchestration	2	2	2	0	2
Process unawareness	3	2	1	2	2
Score		24	21	16	22

Table 8-3: Scoreboard functional aspects (WF = Weight Factor)

As you can see BPEL is the winner here. Axis mainly loses points on the fact that it does not support (online) orchestration. For the rest it is a reasonable close finish.

Non-functional aspect	WF	BPEL	WSCI	Axis	Java
Performance	1	1	1	2	2
Implementation availability	2	2	0	2	0
Industry support	3	2	1	2	0
Score		11	4	12	2

Table 8-4: Scoreboard non-functional aspects (WF = Weight Factor)

Axis is the winner for the non-functional aspects, with BPEL close at second place. The lack of available implementation and industry support are what causes WSCI and Java to fail here.

If you add up the scores for all aspects you get the final score, listed in table 8-5. The functional aspects weigh twice as much as the non-functional aspects as the techniques can earn twice as much points on the functional aspects. This makes sense, since for example industry support is nice, but if the technique is not suitable for the task it is not of much use.

	BPEL	WSCI	Axis	Java
Functional aspects	24	21	16	22
Non-functional aspects	11	4	12	2
Total score	35	25	28	24

Table 8-5: Total score

BPEL is the winner in this comparison, with only one point below the maximum score of 36. To prove that BPEL is actually suitable for service profiling, a prototype has been made that is described in the next chapter.

9 Prototype

This chapter describes the prototype that was built to demonstrate the suitability of BPEL for service profiling.

9.1 Introduction

The conclusion of chapter 8 was that BPEL had the best score in the comparison. A prototype has been made to see if BPEL is indeed a suitable technique for service profiling and to get some hands on experience on service profiling and BPEL.

For a quick implementation of a prototype, an already existing BPEL implementation was used. On the J2EE platform, two implementations were available: IBM BPWS4J and the Collaxa BPEL server. The last one was chosen because of the BPEL console that makes it possible to easily debug, audit and test deployed processes.

9.2 The Collaxa BPEL server

The basis for the prototype is the BPEL implementation from Collaxa [collaxa], the Collaxa BPEL server 2.0. Several names are being used for the same product: Collaxa BPEL server, Collaxa BPEL Orchestration Server, Collaxa Orchestration Server and Collaxa Web Service Orchestration Server. In this report, the name Collaxa BPEL server is used.

The 30-day trial version is available for download on the Collaxa website in two flavors: Collaxa Stand Alone (which includes the JBoss application server and Pointbase database server) and Collaxa for BEA WebLogic. Apart from these platforms, the Collaxa BPEL server is available for the SunONE application server, IBM WebSphere and Oracle 9i.

For the prototype, the stand-alone package (Collaxa BPEL Server 2.0 Release Candidate 2) is used. The package contains:

- JBoss application server
- Pointbase database server
- Collaxa BPEL engine (orchestration server)
- BPEL console (a web application deployed on JBoss that can be used to view and test deployed processes and monitor, audit and debug running and completed process instances)
- API documentation
- Examples
- Command-line tools for deploying a process

Apart from BPEL, the Collaxa BPEL server supports JBPEL, a JSP-like programming abstraction that combines BPEL with Java. JBPEL offers the asynchrony, flow coordination and compensating business flow capabilities of BPEL, WS-Transaction and WS-Coordination, plus native support for Java, EJBs and JCA, JSP integration, JMS and e-mail with attachments based messaging, sub flows, events, dynamic branching, sophisticated join patterns and more. But since JBPEL is not BPEL it is out of scope for this project.

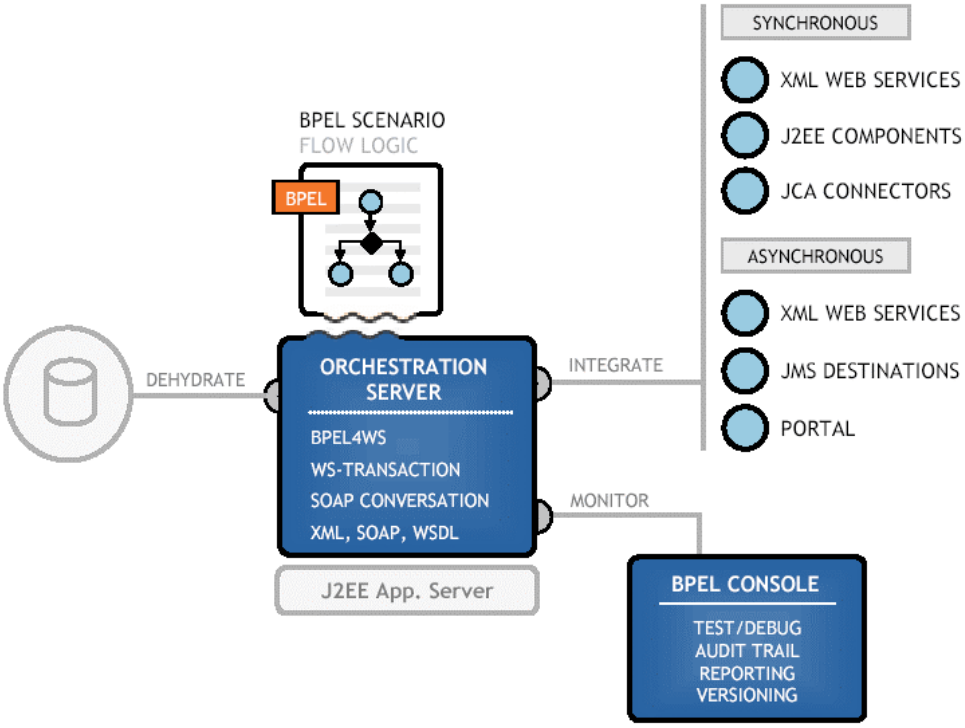

Figure 9-1: Collaxa BPEL server architecture (Collaxa website [collaxa])

Figure 9-1 shows an overview of the server architecture. Note that integrating components, connectors, etc. other than XML web services requires a JBPEL rather than a BPEL process and is out of scope for this project. Dehydrate in figure 9-1 means that inactive instances are stored in the database until they become active again.

9.3 Designing, deploying and testing a BPEL process

The design and deployment of a BPEL process can be broken into four steps:

- Add partner link information to auxiliary and base services if not already present
- Design the process
- Make deployment descriptors
- Deploy the process using the provided deployment tools

9.3.1 Partner links

BPEL requires the WSDL of each service to contain information about the partner links describing the roles the service can play in a conversation. Since this is a rather BPEL-specific feature, you usually first have to add this information to the WSDL file of a service. Refer to section 4.2.1 for more information about partner links in BPEL.

9.3.2 The process

The process description is formatted in BPEL, a kind of XML scripting language. You can design the process in a variety of ways: you may choose to directly write the XML code in a text or XML editor like GEL (www.gexperts.com), use a graphical environment like VisualScript XML (www.visualscript.com), the IBM BPWS Eclipse plug-in [bpws4j] or the Collaxa BPEL Designer Eclipse plug-in (www.collaxa.com) (see figure 9-2 for screenshots). In the design process, you will need the WSDL files of the base and auxiliary services as the partner links, operations, messages and port types are used or referenced in the BPEL file.

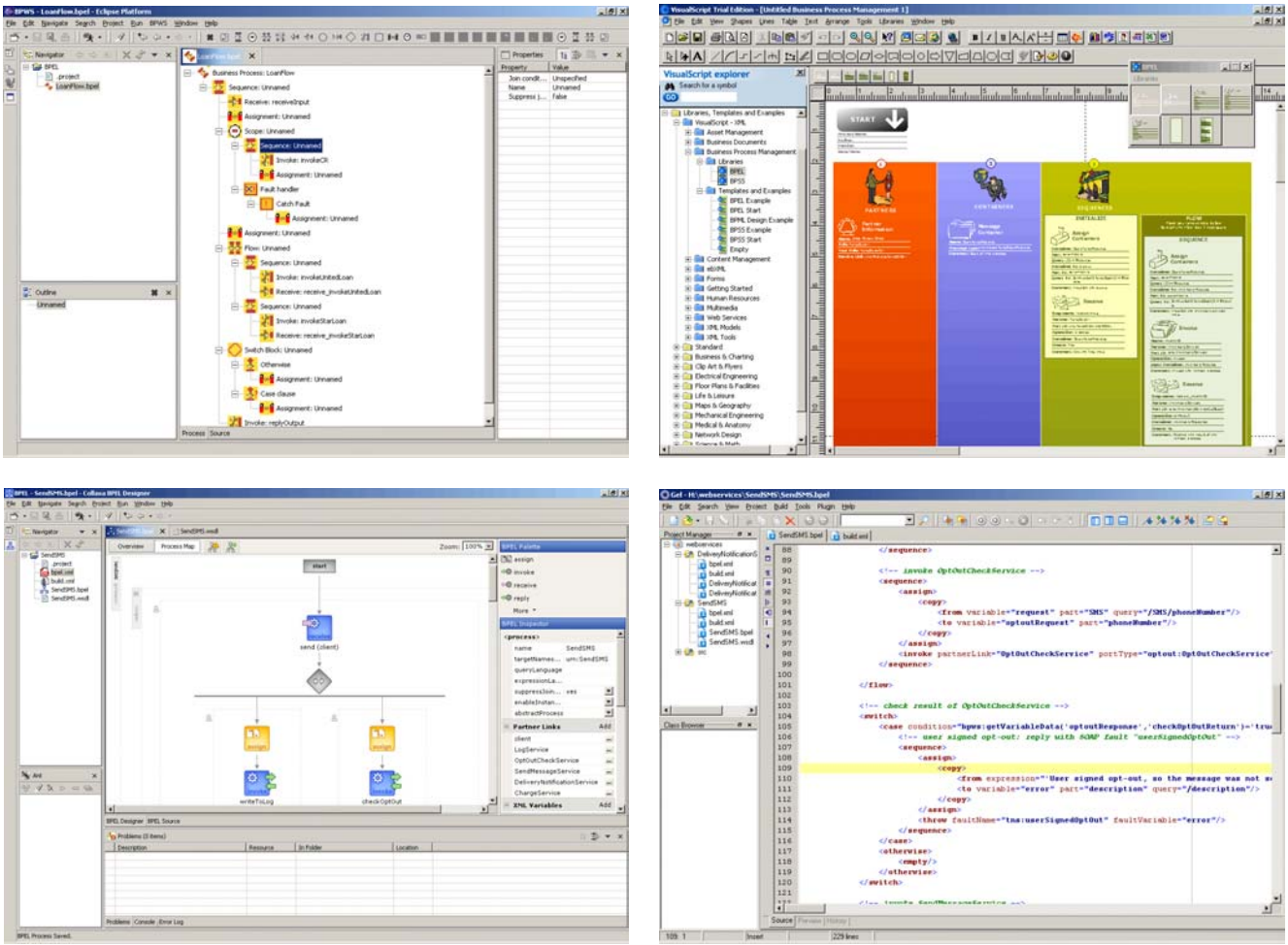

Figure 9-2: Screenshots of four ways to edit a BPEL process: IBM BPWS Eclipse plug-in, VisualScript XML, Collaxa BPEL Designer Eclipse plug-in and GEL.

Since the BPEL process itself is invoked as a web service, a WSDL file for the process has to be provided for the process. A development environment can (partially) generate this WSDL file, or it can be hand made in a text or XML editor.

9.3.3 Deployment descriptor

If a BPEL process needs to be deployed in the Collaxa BPEL server, two additional files are needed: a build file containing the tasks for the Apache Ant tool and a BPEL deployment descriptor containing the URLs or filenames of the files needed for deploying the process: the BPEL process description, its WSDL and the WSDLs of all web services used in the process.

The build file usually contains a "bpelc" task for compiling the BPEL process, and can also include other tasks for deploying additional web services, BPEL processes, servlets, etc. The build file is also used to specify the BPEL domain the process is deployed to and the name and revision tag of the process.

The Collaxa BPELZ editor automatically creates the deployment descriptor and build file. The editor also provides integration with the Collaxa BPEL server as you can deploy processes from within the editor.

9.3.4 Deploy

Running Apache Ant on the build file deploys a BPEL process. A command line script for invoking Ant the right way is packaged with the Collaxa BPEL server (collaxa/bin/cxant.bat). The "bpelc" task runs a syntax check on the BPEL process and all specified WSDLs before a BPEL execution map is compiled. This execution map is a Java representation of the BPEL process. Then everything (the BPEL source, java source and classes of the execution map and WSDLs) is packed into a jar file and deployed on the server. The directory collaxa/domains/default/deploy contains all jars of the processes deployed in the default domain. The filename of the jar file is formatted as "bpel_[name of the process]_[revision tag].jar". If a process is deployed with the same name and revision, the jar file is overwritten, causing any running instances to throw an exception.

The generated jar file is a package that can instantly be deployed on other Collaxa BPEL servers (with the same server software version). The BPEL source can be deployed on other BPEL implementations but may require a different deployment descriptor and deployment procedure.

The Collaxa BPEL server supports side-by-side versioning of processes: different revisions of the same process can coexist. Each revision is packed into its own jar file. The endpoint reference of a BPEL process looks like http://servername/collaxa/default/SendSMS/1.0 for the 1.0 revision of the SendSMS process on the default domain. If the revision tag is omitted, the latest deployed revision is always used to instantiate the new process instance. The WSDL file of the process always contains the endpoint reference to the latest deployed revision, including the revision tag.

If a new revision of a process is deployed, the old revision remains available, so existing instances of a process can continue to run until they terminate normally, while new instances use the new revision. This technique is called dynamic update and is one of the requirements in a telecom environment to be able to deliver high availability.

Using the BPEL console, the mode (open or closed) and state (on or off) of a process can be controlled. If a process is closed, no new instances may be instantiated but existing instances are permitted to complete normally. If the state of a process is set to "off", no new instances may be instantiated and access to existing instances will be denied.

9.3.5 Test

The Collaxa BPEL server comes with a BPEL console that allows developers to inspect, debug, audit, test and manage deployed processes. Below are two screenshots of the BPEL console.

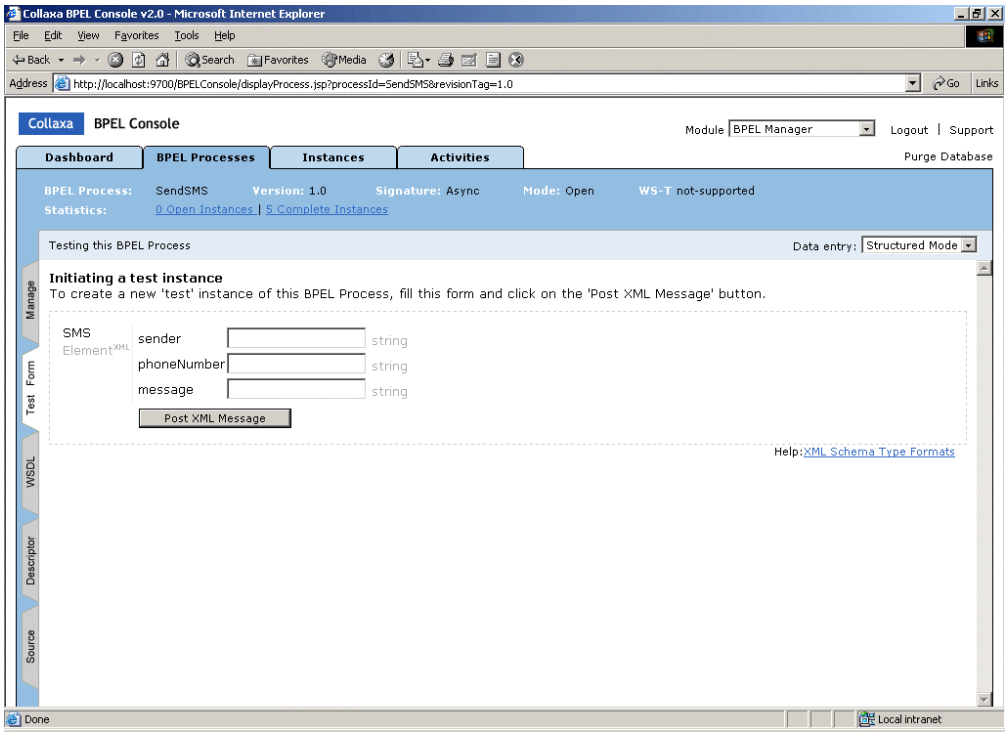

Figure 9-3a: Screenshot of the BPEL console: testing a process

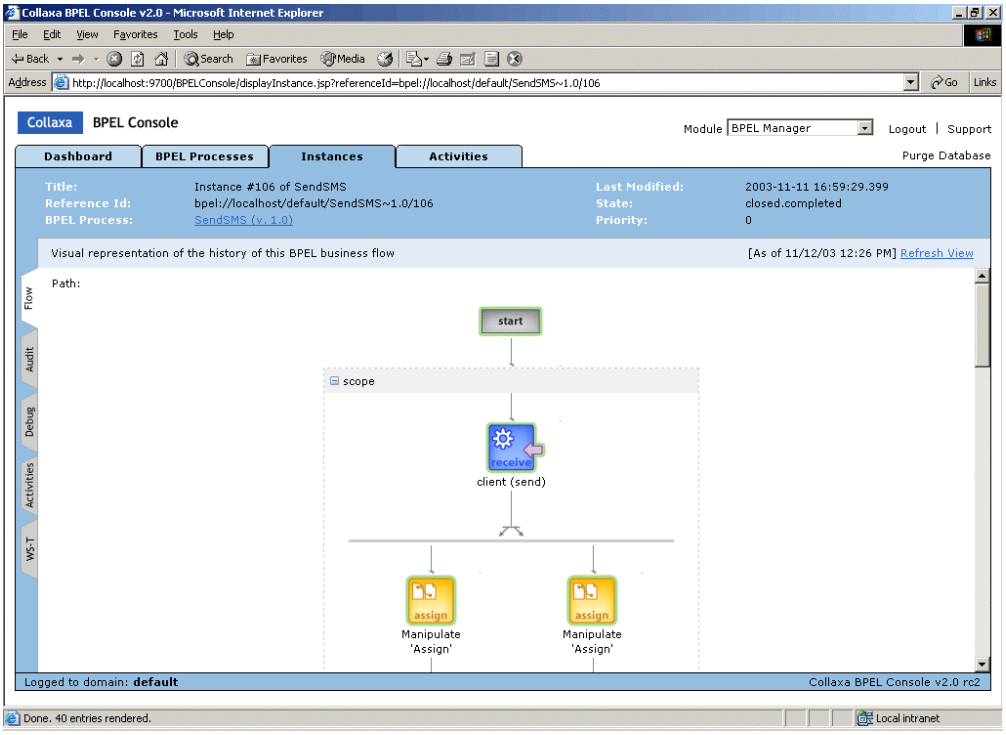

Figure 9-3b: Screenshot of the BPEL console: visual representation of a completed instance

9.4 Example SendSMS process

Consider the SendSMS service described in section 3.2. To show more of the service profiling features, the example will be modified to include an asynchronous web service, a fault handler and timeout handling. The flowchart of the modified SendSMS service is shown in figure 9-4.

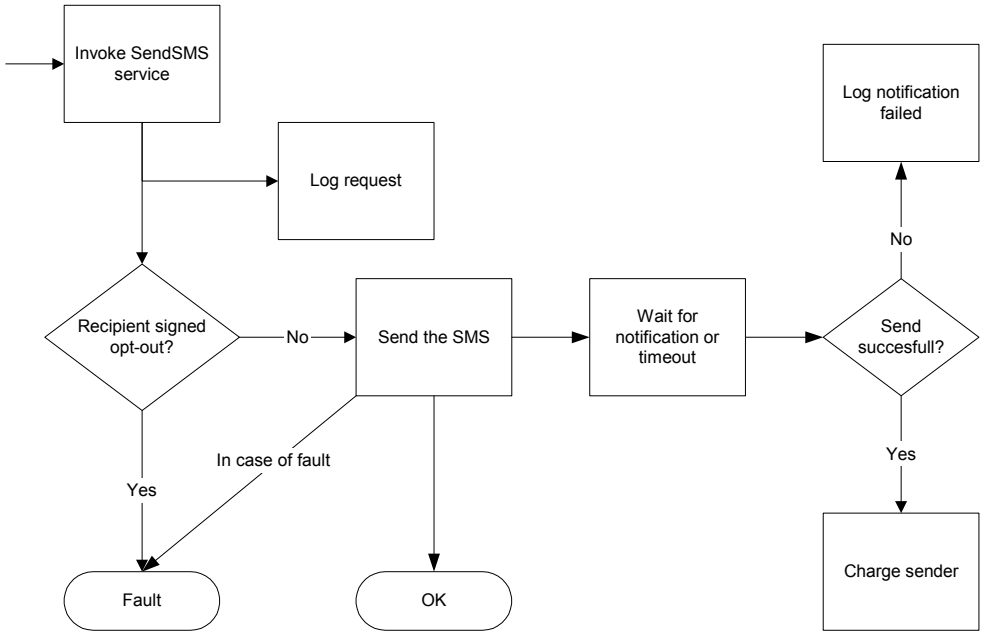

Figure 9-4: Modified version of the flowchart in figure 3-1

9.4.1 Base and auxiliary services

In order to implement the example process, these auxiliary and base services are implemented:

- LogService, a one-way web service that takes a text as parameter and writes the text and the current time to an event log.
- OptOutCheckService, a synchronous web service that takes a phone number as parameter and returns true if the owner of the specified phone number has signed an opt-out form, otherwise the service will return false.
- ChargeService, a synchronous web service that takes two parameters: a phone number and an amount. The service always returns true.
- SendMessageService, the base service. This web service needs three parameters: sender phone number, recipient phone number and a message. The service returns a message identifier that can be used to get a notification of the delivery of the message, using the DeliveryNotificationService.
- DeliveryNotificationService, an asynchronous web service that is invoked with a message identifier and performs a callback if the message is delivered.

All services are implemented as Java classes and deployed as web services using Axis on Tomcat, except for the DeliveryNotificationService, which is implemented as a BPEL process on the Collaxa BPEL server. Refer to figure 9-5 for an overview.

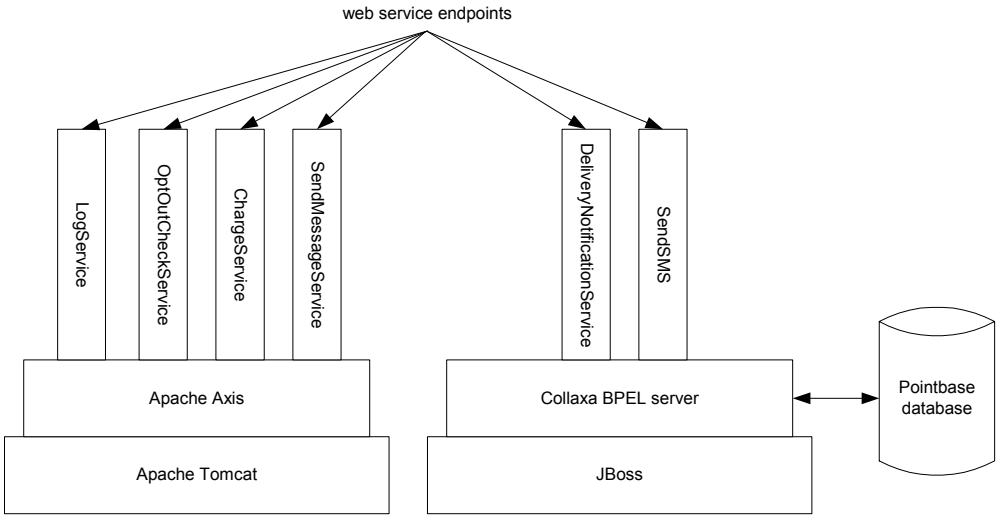

Figure 9-5: Overview of the services in the prototype

To make things easy, these web services don't actually do anything except writing a line to the event log and returning values. For testing purposes, the result of the base and auxiliary services are dependent on the last digit of the phone number of the recipient. Details can be found in section 9.5: Testing the process.

9.4.2 The BPEL process

The BPEL process was created using an XML editor, as not all graphical editors are very easy to work with yet and to maintain complete control over the BPEL source code.

Figure 9-6 shows an outline of the actual BPEL process and provides a link between the flowchart in figure 9-4 and the BPEL source code. All words between < and > map directly to BPEL activities that can be found in the source code, while the names with the little arrows refer to the BPEL variables and SOAP messages that are exchanged between the BPEL process and the external web services.

In the upper left corner are the global fault handlers. They catch any fault that is either explicitly thrown by the BPEL process or occur in the process. Around the <invoke SendMessageService/> is a sub-scope with its own fault handler to catch any fault in any namespace when invoking the SendMessageService and rethrows the "sendFailed" fault in the namespace of the process.

Refer to appendix 1 for the complete BPEL source code and appendix 2 for the accompanying WSDL.

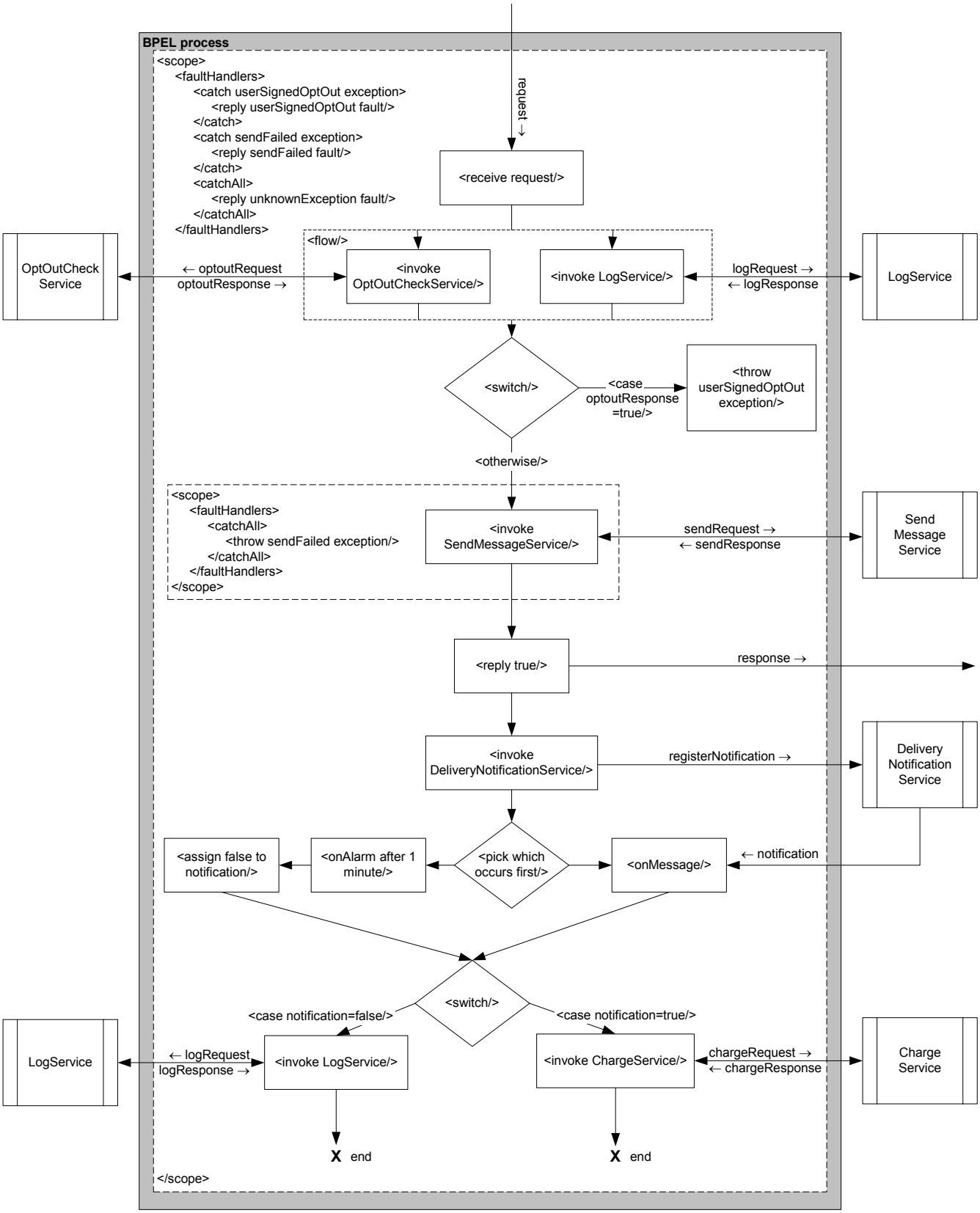

Figure 9-6: Pseudo-flowchart of the SendSMS process that can be mapped to the BPEL source

9.5 Testing the process

Several test cases are built in to be able to test specific aspects of the service profiling process. The behavior of the base and auxiliary services is dependent on the last digit of the phone number of the recipient.

9.5.1 Case 1: user signed opt-out

If the last digit of the phone number is a 0, the OptOutCheckService will return true, indicating that the recipient has signed an opt-out form and the sender is not allowed to send a message to this recipient. The SendSMS process throws a fault that is caught by the global fault handlers of the process and a SOAP fault message with the userSignedOptOut fault is returned.

9.5.2 Case 2: service throws exception

If the last digit of the phone number is a 1, the SendMessageService throws an exception that is caught by the local fault handler in the SendSMS process. The fault is rethrown as a sendFailed fault in the SendSMS namespace. This fault is caught by the global fault handlers of the process and a SOAP fault message with the sendFailed fault is returned.

9.5.3 Case 3: delivery failed

If the last digit of the phone number is a 2, the DeliveryNotificationService performs a callback on the "notification" operation 30 seconds after the "register" operation. The value of the notification is "false", indicating that the delivery failed. The SendSMS process does therefore not invoke the ChargeService but invokes the LogService to write to the log file that the delivery has failed. The SendSMS process will return "true" indicating that sending the message succeeded.

9.5.4 Case 4: delivery notification timeout

If the last digit of the phone number is a 2, the DeliveryNotificationService performs a callback on the "notification" operation 3 minutes after the "register" operation. The value of the notification is "true", indicating that the delivery succeeded. But the timeout (<onAlarm/>) in the SendSMS process is set at one minute so the timeout will occur first. The process does not invoke the ChargeService but invokes the LogService to write to the log file that the delivery has failed. The SendSMS process will return "true" indicating that sending the message succeeded.

9.5.5 Case 5: normal operation

If the last digit of the phone number is anything but a 0, 1 or 2, the DeliveryNotificationService performs a callback on the "notification" operation 30 seconds after the "register" operation. The value of the notification is "true", indicating that the delivery succeeded. The process invokes the ChargeService to charge the sender for the message. The SendSMS process will return "true" indicating that sending the message succeeded.

9.5.6 Performing the tests

The tests were performed on a desktop PC running the prototype on Apache Tomcat, Apache Axis and the Collaxa BPEL server. The XMLSpy tool (www.xmlspy.com) was used to construct the SOAP requests for invoking the SendSMS service and to inspect the returned SOAP messages. XMLSpy automatically generated a SOAP request based on the WSDL of a web service. Since the XMLSpy tool is a general web service tool and not BPEL aware, this also was a test for the unawareness of the whole underlying process.

The prototype passed all tests successfully.

9.6 Discussion

The prototype gives a good notion of the possibilities of BPEL. It also shows that BPEL is very well suited for service profiling. The prototype contains almost all functional aspects that were identified in chapter 1.

The prototype supports (online) choreography because multiple services are combined into one with the result being dependent on the parameters of the invocation, orchestration because of several conditional actions (see the two <switch/> blocks in figure 9-6), the invoking tool (XMLSpy) does not need to know anything of BPEL and the service lifecycle is well defined an manageable through the BPEL console. Transactions are supported via the fault handlers in the process. Longer running transactions with rollback procedures are not tested, but the Collaxa BPEL Server comes with a few examples that support transactions using the built-in support for WS-Transaction. Additional features that are included in the prototype are both synchronous and asynchronous web services and exception handling.

It is not possible to say very much specific about the performance in a real world situation. The tests were performed on a normal desktop machine (Pentium 3, 800Mhz, 512MB RAM) and showed reasonable fast response times once the process was loaded into memory. Network latencies were not tested as everything was run on one machine only. For extra performance and reliability, the Collaxa BPEL Server is scalable into a cluster of multiple servers with a centralized (Oracle) database.

Designing a process was found to be rather easy when you are familiar with BPEL. Unfortunately, present editors still require that the designer has some knowledge about XML, XMLSchema, WSDL and XML namespaces. Processes designed with standard XML and BPEL tools could easily be deployed on the BPEL server using the deployment tools that came with it.

An offered service in a telecom environment needs some extra features. Usually a profile is selected that contains the settings and Service Level Agreement (SLA) that belong to the service requester. This profile selection process can best be combined with the authentication and authorization since that is something that needs to be performed for each process. This way not every process that is designed needs to explicitly include a profile selection service.

Because BPEL instances can not share variables, it is not easy to embed SLA runtime enforcement into a BPEL process. Two alternatives are to make a SLA runtime enforcement web service or embed it in the base and auxiliary services where needed.

Another concern when using BPEL in a telecom environment is if the BPEL server supports the OA&M (operations, administrations and management) and provisioning facilities of an operator. If this is not the case, managing the BPEL server may be very costly. In other words, it might be a concern to what extent the BPEL server can be integrated in the network of an operator.

The Collaxa BPEL Server has some limitations regarding the BPEL specification. The server does not support BPEL properties, which makes it impossible to read or write the SOAP headers. Since the Collaxa BPEL Server takes care of the WS-Addressing headers in asynchronous web services, it is not possible to create a combined synchronous and asynchronous process (a process with a request/reply/notify interaction scheme).

10 Limitations and improvements

Based on the experience gained during the research, this chapter presents the limitations when using BPEL for service profiling. The last paragraph contains some suggestions for improvements to overcome the limitations.

10.1 Introduction

The ideal service profiling technique would score the maximum number of points in chapter 8. But the criteria mentioned in chapter 8 are not the only aspects the ideal technique should include. As BPEL scores almost the maximum number of points, it serves as a good starting point. By discussing the limitations of BPEL that came up during the research and the implementation of the prototype it becomes clear what features the ideal technique should have and how BPEL can evolve towards that ideal technique.

10.2 BPEL limitations

The only aspect from chapter 8 where BPEL does not score the maximum number of points is performance. This is because all base and auxiliary services must be web services. So even if a service is a local service that can easily be directly invoked there is the web service overhead of SOAP encoding and decoding including all extra features like WS-Transaction or WS-Addressing. The overall performance of the service profiling process would be much better if local services do not need to have a web service interface, but an interface with less (processing) overhead. Collaxa introduces a number of extensions to BPEL which make it possible to integrate other types of services like J2EE components and JMS destinations (see figure 9-1), but these extensions are not standardized, and (what might be even more important) not platform independent.

The fact that BPEL expects a partnerlink declaration in the WSDL of every service that is used in the process, limits the easy development of BPEL processes, as almost no web services at all include partner links (yet). This could be something that is automatically added in the future if BPEL is very widely used, but since it does not provide extra information but is merely an abstraction from the WSDL port types, maybe it would have been better not to implement it in this way. Now a BPEL designer first has to make an adapted version of the WSDL for each service before the service can be used in a BPEL process.

The number of operations on variables, properties and expressions is very limited. For the BPEL 1.1 version, XPath 1.0 is used as query and expression language. XPath 1.0 provides only a very limited set of data types and operations (for example, there is no date/time data type and division of numbers is not possible). It is possible to use a different language, but the BPEL engine has to support that language. XPath 1.0 is at present the only language a BPEL engine must support, so using another language (or another version of XPath) will probably break compatibility with other BPEL engines.

As signalled in section 9.6, BPEL itself is not suitable to embed SLA runtime enforcement. Since introducing variables that are shared between instances of a process would mean a rather fundamental change in BPEL, it might be better to design a SLA enforcement (web) service or embed the enforcement in the base and auxiliary services where needed.

The current BPEL editors are very limited in their possibilities. Of the editors I have tried (refer to section 9.3.2), the Collaxa BPEL designer is the most advanced and easy to use editor, but it is still in very early beta stage. This editor is the only one that lets you import a WSDL and select possible values from drop down boxes instead of having to type them yourself. The process description is built by dragging and dropping activities into place and fill in or select the right parameters. Although this editor is on the right way, you sometimes still need to type in message names and namespace identifiers. Also, you just have to type in the WSDL of the process. So a BPEL designer still needs knowledge about XML, WSDL, XMLSchema and XML namespaces, etc.

10.3 Improvements

BPEL is on the right track for service profiling, but there are some points that need to be improved in future versions:

- Extensions should be included to make it possible to call services other than web services
- The partner links in the WSDL must be optional
- The query and expression language must be upgraded to a language with more features, like XPath 2.0 (which includes XQuery)
- The BPEL editors need to mature so a BPEL designer only has to focus on the business aspects of the process and spend only minimal effort on the technical details

11 Conclusions and recommendations

This chapter presents the conclusions of this research. It also identifies some recommendations for Ericsson for implementing a service profiling environment.

11.1 Conclusions

From the web services state of the art overview it becomes clear that although the basic standards for web services are well defined, the additional specifications are far from standardized and some of them show considerable overlap with other specifications. The WS-I does a good job on interoperability by defining profiles that state what versions and what specifications to use. Also, IPR claims can form a barrier for the worldwide adoption of some of the specifications.

The best service profiling technique of the one reviewed in this report is BPEL. The other techniques lose most points on lack of support for online orchestration (Axis), implementation availability and industry support (WSCI and the proprietary Java solution).

The BPEL service profiling prototype showed that BPEL is indeed suitable for service profiling, but it also showed the limitations when using BPEL for this purpose. The main issue is performance because of the web services overhead when invoking base and auxiliary services. The other limitations are likely to be taken away by later versions of the BPEL language.

Furthermore, the prototype showed that designing a new process or making a new version of an already existing service is very easily done using standard XML and BPEL tools and the deployment tools that came with the BPEL server.

Right now, the BPEL language and products are probably not mature enough to offer telecom grade applications. But with such a large industry support it will only be a matter of time until the BPEL language, servers and editors mature.

11.2 Recommendations

Ericsson should watch BPEL closely because it is likely to become the dominant service integration technique for web services. If Ericsson wants to stay informed of early developments in the BPEL area, it is advisable to become an OASIS Contributor and join the BPEL Technical Committee. It is up to Ericsson if it also wants to contribute to the BPEL language, as it is not its core business.

For the integration of BPEL in Ericsson products, additional research should be done to check the performance and reliability of the BPEL implementation that will be used. Also, integration with OA&M and provisioning facilities should be tested.

A market study needs to show if operators are willing to use BPEL for creating service offerings and if they are ready for this new approach in service creation by (visually) scripting a process. The study also needs to come up with specific requirements for the service profiling environment. If they are very different from the functional and non-functional criteria used in this research the choice for a certain technique might be completely different.

The OASIS BPEL Technical Committee should consider the suggested improvements by allowing other services than web services, making the partner link declarations in WSDL optional and upgrade the query and expression language to XPath 2.0.

References

[3gpp] *Third Generation Partnership Project*, http://www.3gpp.org/

[anderson] A. Anderson, *Minutes from XACML Focus Group on RBAC, 24 april 2003*, XACML OASIS mailing list, April 24, 2003, http://lists.oasis-open.org/archives/xacml/200304/msg00067.html

[axis] *Apache Axis*, http://ws.apache.org/axis/

[axislicense] The Apache Software Foundation, *The Apache Software License, Version 1.1*, http://ws.apache.org/dist/LICENSE.txt

[basicprofile] Keith Ballinger et al., *Basic Profile Version 1.0a*, August 8, 2003, http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0a.htm

[berlind] David Berlind, *BPEL4WS vs. WSCI*, March 6, 2003, http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2912083-2,00.html

[bpel1.0] Francisco Curbera et al., *Business Process Execution Language for Web Services version 1.0*, August 2002, http://msdn.microsoft.com/library/en-us/dnbiz2k2/html/bpel1-0.asp

[bpel1.1] Tony Andrews et al., *Business Process Execution Language for Web Services version 1.1*, May 5, 2003, http://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf

[bpellicense] Microsoft, *Royalty Free Business Process Execution Language for Web Services Specification License Agreement*, http://msdn.microsoft.com/webservices/docs/bpel_license.aspx

[bpws4j] *The IBM Business Process Execution Language for Web Services Java™ Run Time (BPWS4J)*, http://alphaworks.ibm.com/tech/bpws4j

[btp] OASIS Committee Specification, Mike Abott et al., *Business Transaction Protocol version 1.0*, June 3, 2002, http://www.oasis-open.org/committees/business-transactions/documents/specification/2002-06-03.BTP_cttee_spec_1.0.pdf

[chappell] Dave Chappell, *Will the Real Reliable Messaging Please Stand Up?*, April 8, 2003, http://xml.coverpages.org/Chappell-WSRelSOAP.pdf

[collaxa] *Collaxa BPEL Server 2.0*, http://www.collaxa.com/product.welcome.html

[coverpages] Cover Pages, *Web Services Specifications for Business Transactions and Process Automation*, August 12, 2002, http://xml.coverpages.org/ni2002-08-12-a.html

[della-libera] Giovanni Della-Libera et al., *Security in a Web Services World: a Proposed Architecture and Roadmap; A Joint White Paper from IBM Corporation and Micorosft Corporation*, April 7, 2002, http://msdn.microsoft.com/library/en-us/dnwssecur/html/securitywhitepaper.asp

[dime] IETF Internet-Draft, Henrik Frystyk Nielsen et al., *Direct Internet Message Encapsulation (DIME)*, June 17, 2002, http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-02.txt

[ebxml] *ebXML*, http://www.ebxml.org/

[ecma] Ecma International, *Standard ECMA-348: Web Services Description Language (WSDL) for CSTA Phase III*, June 2003, http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-348.pdf

[fontana] J. Fontana, *Liberty turning first spec over to OASIS as turmoil brewing over identity management*, April 11, 2003, http://www.nwfusion.com/news/2003/0411liboasis.html

[freedman] Michael Freedman, *Overview of WSRP and JSR 168 standards – An interview with Michael Freedman,* July 12, 2002, http://portalstudio.oracle.com/pls/ops/docs/FOLDER/COMMUNITY/PDK/ARTICLES/overview.wsrp.jsr168.standards.interview.html

[furniss] Peter Furniss and Alistair Green, *WS-C+T and BTP: Comments and comparisons*, November 20, 2002, http://www.choreology.com/standards/btp_wsc+t.html

[govatos] Greg Govatos, *UDDI is Yellow Pages of Web services*, May 27, 2002, http://www.nwfusion.com/news/tech/2002/0527tech.html

[httpr] Andrew Banks et al., *HTTPR Specification*, April 1, 2002, http://www-106.ibm.com/developerworks/library/ws-httprspec/

[irani] R. Irani, *Part II – ebXML and Web Services, The Way to do Business*, July 25, 2001, http://www.webservicesarchitect.com/content/articles/irani03.asp

[kearns] D. Kearns, *Java scripting languages: Which is right for you?*, April 5, 2002, http://www.javaworld.com/javaworld/jw-04-2002/jw-0405-scripts.html

[liberty] *The Liberty Alliance Project*, http://www.projectliberty.org/

[mms] 3GPP TS 23.140: *Multimedia Messaging Service (MMS); Functional Description; Stage 2*, version 5.6.0 Release 5, March 2003, http://www.3gpp.org/ftp/Specs/archive/23_series/23.140/23140-560.zip

[oma] *Open Mobile Alliance*, http://www.openmobilealliance.org/

[omaoverview] Open Mobile Alliance, *Fostering Worldwide Growth in the Mobile Service Market*, March 2003, http://www.openmobilealliance.org/docs/OMA-PublicOverviewFinal.pdf

[omarelease] Open Mobile Alliance, *Overview of OMA Release Program*, April 2003, http://www.openmobilealliance.org/docs/OMA_Release_Program_Overview_2003_April.pdf

[parlay] *The Parlay Group*, http://www.parlay.org/

[roberts] P. Roberts, *Liberty Alliance releases spec to coexist with Passport*, February 7, 2003, http://www.nwfusion.com/news/2003/0207liberallia.html

[saml] *OASIS Security Services Technical Committee*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[soap] W3C Note, Don Box et al., *Simple Object Access Protocol (SOAP) 1.1*, May 8, 2000, http://www.w3.org/TR/SOAP/

[soapattach] W3C Note, John Barton et al., *SOAP Messages with Attachments*, December 11, 2000, http://www.w3.org/TR/SOAP-attachments

[soapattach2] Adam Bosworth et al., *SOAP Messages with Attachments*, April 1, 2003, http://dev2dev.bea.com/technologies/webservices/SOAP_Messages_Attachments.jsp

[taft] Daryll Taft, *Tackling Web Service Transactions*, September 23, 2003, http://www.eweek.com/article2/0,4149,1276856,00.asp

[uddi] *OASIS UDDI member section*, http://www.uddi.org/

[wsack] Ruslan Bilorusets et al., *Web Service Acknowledgement Protocol (WS-Acknowledgement) 0.91*, February 26, 2003, http://dev2dev.bea.com/technologies/webservices/WS-Acknowledgement-0_9.jsp

[wsaddressing] Adam Bosworth et al., *Web Services Addressing (WS-Addressing)*, March 13, 2003, http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-addressing.asp

[wsattach] IETF Internet-Draft, Henrik Frystyk Nielsen et al., *WS-Attachments*, June 17, 2002, http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-soap-01.txt

[wscallback] Yaron Goland et al., *WS-CallBack Protocol (WS-CallBack) 0.91*, February 26, 2003, http://dev2dev.bea.com/technologies/webservices/WS-CallBack-0_9.jsp

[wsci] W3C Note, Assaf Arkin et al., *Web Service Choreography Interface (WSCI) 1.0*, August 8, 2002, http://www.w3.org/TR/wsci/

[wsci-ipr] David Orchard et al., *Submission request to the World Wide Web Consortium*, June 28, 2002, http://www.w3.org/Submission/2002/04/

[wscoordination] Felipe Cabrera et al., *Web Services Coordination (WS-Coordination)*, August 2002, http://dev2dev.bea.com/technologies/webservices/ws-coordination.jsp

[wsdl] W3C Note, Erik Christensen et al., *Web Services Description Language (WSDL) 1.1*, March 15, 2001, http://www.w3.org/TR/wsdl

[wsfl] Prof. Dr. F. Leymann, *Web Services Flow Language (WSFL 1.0)*, May 2001, http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

[wsia] OASIS Web Services Interactive Applications Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsia

[wsmanage] XML Cover Pages, *IBM, Computer Associates, and Talking Blocks Release WS-Manageability Specification*, September 15, 2003, http://xml.coverpages.org/ni2003-09-15-b.html

[wsmsgdata] Yaron Goland et al., *Web Services Message Data (WS-MessageData) 0.91*, February 26, 2003, http://dev2dev.bea.com/technologies/webservices/WS-MessageData-0_9.jsp

[wspolicy] MSDN, *WS-Policy Specification Index Page*, http://msdn.microsoft.com/library/en-us/dnglobspec/html/wspolicyspecindex.asp

[wsreferral] Henrik Frystyk Nielsen et al., *Web Services Referral Protocol (WS-Referral)*, October 23, 2001, http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-referral.asp

[wsreliability] Colleen Evans et al., *Web Services Reliability (WS-Reliability) Ver1.0*, January 8, 2003, http://developers.sun.com/techtopics/webservices/ws-reliability.v1.0.pdf

[wsreliablemsg] Ruslan Bilorusets et al., *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)*, March 13, 2003, http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-reliablemessaging.asp

[wsrouting] Henrik Frystyk Nielsen and Satish Thatte, *Web Services Routing Protocol (WS-Routing)*, October 23, 2001, http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-routing.asp

[wsrp] Alan Kropp et al., *Web Services for Remote Portlets Specification; working draft 0.94*, April 10, 2003, http://www.oasis-open.org/committees/download.php/1510/wsrp-specification-1.0-draft-0.94.pdf

[wssecurity] MSDN, *WS-Security Specification Index Page*, http://msdn.microsoft.com/library/en-us/dnglobspec/html/wssecurspecindex.asp

[wstransaction] Felibe Cabrera et al., *Web Services Transaction (WS-Transaction)*, August 2002, http://dev2dev.bea.com/technologies/webservices/ws-transaction.jsp

[wstransaction2] XML Cover Pages, *Updated Specifications for the Web Services Transaction* Framework, September 16, 2003, http://xml.coverpages.org/ni2003-09-16-a.html

[wsui] Ed Anuff et al., *Web Service User Interface (WSUI) 1.0*, July 26, 2002, http://www.wsui.org/doc/wsui

[wsxl] IBM Note, Ali Arsanjani et al., *(WSXL) Web Services Experience Language Version 2*, April 10, 2002, http://www-106.ibm.com/developerworks/library/ws-wsxl/

[xacml] *OASIS eXtensible Access Control Markup Language TC*, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

[xlang] Satish Thatte, *XLANG Web Services for Business Process Design*, May 2001, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/

[xmlrpc] *XML-RPC Home Page*, http://www.xmlrpc.org/

List of abbreviations

3GPP	Third Generation Partnership Project
API	Application Programming Interface
ASP	Application Service Provider
B2B	Business to Business
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BTP	Business Transaction Protocol
CPA	Collaboration Protocol Agreement
CPP	Collaboration Protocol Profile
CSTA	Computer Supported Telecommunications Applications
DIME	Direct Internet Message Encapsulation
ebXML	electronic business XML
EJB	Enterprise Java Bean
ETSI	European Telecommunication Standard Initiative
HTML	Hypertext Markup Language
HTTP	HyperText Transfer Protocol
HTTPR	Reliable HTTP
ICT	Information and Communication Technology
ID	Identifier
ID-FF	Identity Federation Framework
ID-SIS	Identity Services Interface Specifications
ID-WSF	Identity Web Services Framework
IETF	Internet Engineering Task Force
IN	Intelligent Networks
IP	Internet Protocol
IPR	Intellectual Property Rights
ISO	International Organization for Standardization
IT	Information Technology
J2EE	Java 2 Enterprise Edition
JAIN	Java Advanced Intelligent Network
JBPEL	Java BPEL
JCA	J2EE Connector Architecture
JCP	Java Community Process
JMS	Java Message Service
JSP	Java Server Pages
LDAP	Lightweight Directory Access Protocol
MIME	Multi-purpose Internet Mail Extensions
MMS	Multimedia Messaging Service
OASIS	Organization for the Advancement of Structured Information Standards
OA&M	Operations, Administration and Management
OMA	Open Mobile Alliance
OSA	Open Service Access
P3P	Platform for Privacy Preferences Project
PAP	Policy Administration Point
PEP	Policy Enforcement Point
PDP	Policy Decision Point
PKI	Public Key Infrastructure
SAML	Security Assertion Markup Language
SLA	Service Level Agreement
SMS	Short Message Service
SOAP	Simple Object Access Protocol

SwA	SOAP with Attachments
TC	Technical Committee
TCP	Transfer Control Protocol
TCP/IP	Transfer Control Protocol/Internet Protocol
UDDI	Universal Description, Discovery and Integration
UN/CEFACT	United Nations Centre for Trade Facilitation and Electronic Business
URI	Uniform Resource Identifier
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language
XML-RPC	XML Remote Procedure Call
XrML	eXtensible rights Markup Language
W3C	World Wide Web Consortium
WML	Wireless Markup Language
WS-CAF	Web services Composite Application Framework
WS-CF	Web Service Coordination Framework
WS-CTX	Web Service Context
WS-TXM	Web Service Transaction Management
WS-I	Web Services Interoperability Organization
WSC	Web Service Consumer
WSCI	Web Services Choreography Interface
WSDD	Web Service Deployment Descriptor
WSDL	Web Services Description Language
WSDM	Web Services Distributed Management
WSFL	Web Services Flow Language
WSIA	Web Services for Interactive Applications
WSIL	Web Service Inspection Language
WSP	Web Service Provider
WSRP	Web Services for Remote Portlets
WSUI	Web Services User Interface
WSXL	Web Services eXperience Language

Appendix 1: SendSMS BPEL source

Below is the BPEL source code of the SendSMS service discussed in chapter 9.

```
<process name="SendSMS"
targetNamespace="urn:SendSMS"
suppressJoinFailure="yes"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:tns="urn:SendSMS"
xmlns:log="urn:LogService"
xmlns:optout="urn:OptOutCheckService"
xmlns:send="urn:SendMessageService"
xmlns:not="urn:DeliveryNotificationService"
xmlns:charge="urn:ChargeService"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <!-- links to the web service requester (client) and other web services -->
    <partnerLinks>
        <partnerLink name="client" partnerLinkType="tns:SendSMSLink" myRole="service"/>
        <partnerLink name="LogService" partnerLinkType="log:LogServiceLink" partnerRole="service"/>
        <partnerLink name="OptOutCheckService" partnerLinkType="optout:OptOutCheckServiceLink" partnerRole="service"/>
        <partnerLink name="SendMessageService" partnerLinkType="send:SendMessageServiceLink" partnerRole="service"/>
        <partnerLink name="DeliveryNotificationService" partnerLinkType="not:DeliveryNotificationServiceLink"
                                                                partnerRole="serviceProvider" myRole="serviceRequester"/>
        <partnerLink name="ChargeService" partnerLinkType="charge:ChargeServiceLink" partnerRole="service"/>
    </partnerLinks>

    <!-- declaration of variables representing exchanged messages -->
    <variables>
        <variable name="request" messageType="tns:SendSMSRequest"/>
        <variable name="response" messageType="tns:SendSMSResponse"/>
        <variable name="error" messageType="tns:errorMessage"/>

        <variable name="logRequest" messageType="log:writeToLogRequest"/>
        <variable name="logResponse" messageType="log:writeToLogResponse"/>

        <variable name="optoutRequest" messageType="optout:checkOptOutRequest"/>
        <variable name="optoutResponse" messageType="optout:checkOptOutResponse"/>

        <variable name="sendRequest" messageType="send:sendRequest"/>
        <variable name="sendResponse" messageType="send:sendResponse"/>

        <variable name="registerNotification" messageType="not:registerNotification"/>
        <variable name="notification" messageType="not:notification"/>

        <variable name="chargeRequest" messageType="charge:chargeRequest"/>
        <variable name="chargeResponse" messageType="charge:chargeResponse"/>
    </variables>

    <!-- the process description -->
    <scope variableAccessSerializable="no">

        <!-- fault handlers -->
        <faultHandlers>
            <catch faultName="tns:userSignedOptOut">
                <sequence>
                    <assign>
                        <copy>
                            <from expression="'User signed opt-out, so the message was not sent.'"/>
                            <to variable="error" part="description" query="/description"/>
                        </copy>
                    </assign>
                    <reply partnerLink="client" portType="tns:SendSMS" operation="send" variable="error"
                                                                                faultName="tns:userSignedOptOut"/>
                </sequence>
            </catch>
            <catch faultName="tns:sendFailed">
                <sequence>
                    <assign>
                        <copy>
```

```
                    <from expression="'Send failed. Please try again later.'"/>
                    <to variable="error" part="description" query="/description"/>
                </copy>
            </assign>
            <reply partnerLink="client" portType="tns:SendSMS" operation="send" variable="error" faultName="tns:sendFailed"/>
        </sequence>
    </catch>
    <catchAll>
        <sequence>
            <assign>
                <copy>
                    <from expression="'Unknown exception occured'"/>
                    <to variable="error" part="description" query="/description"/>
                </copy>
            </assign>
            <reply partnerLink="client" portType="tns:SendSMS" operation="send" variable="error"
                                                                                    faultName="tns:unknownException"/>
        </sequence>
    </catchAll>
</faultHandlers>

<!-- the process -->
<sequence>

    <!-- receive initial request -->
    <receive partnerLink="client" portType="tns:SendSMS" operation="send" variable="request" createInstance="yes"/>

    <!-- invoke LogService and OptOutCheckService in parallel -->
    <flow>

        <!-- invoke LogService -->
        <sequence>
            <assign>
                <copy>
                    <from expression="concat('LogService invoked for ',
                                                bpws:getVariableData('request','SMS','/SMS/phoneNumber'))"/>
                    <to variable="logRequest" part="event" query="/event"/>
                </copy>
            </assign>
            <invoke partnerLink="LogService" portType="log:LogService" operation="writeToLog" inputVariable="logRequest"
                                                                                    outputVariable="logResponse"/>
        </sequence>

        <!-- invoke OptOutCheckService -->
        <sequence>
            <assign>
                <copy>
                    <from variable="request" part="SMS" query="/SMS/phoneNumber"/>
                    <to variable="optoutRequest" part="phoneNumber"/>
                </copy>
            </assign>
            <invoke partnerLink="OptOutCheckService" portType="optout:OptOutCheckService" operation="checkOptOut"
                                            inputVariable="optoutRequest" outputVariable="optoutResponse"/>
        </sequence>

    </flow>

    <!-- check result of OptOutCheckService -->
    <switch>
        <case condition="bpws:getVariableData('optoutResponse','checkOptOutReturn')='true'">
            <!-- user signed opt-out: reply with SOAP fault "userSignedOptOut" -->
            <throw faultName="tns:userSignedOptOut"/>
        </case>
        <otherwise>
            <empty/>
        </otherwise>
    </switch>

    <!-- invoke SendMessageService -->
    <assign>
        <copy>
            <from variable="request" part="SMS" query="/SMS/phoneNumber"/>
            <to variable="sendRequest" part="phoneNumber" query="/phoneNumber"/>
        </copy>
    </assign>
    <assign>
```

```
        <copy>
            <from variable="request" part="SMS" query="/SMS/sender"/>
            <to variable="sendRequest" part="sender" query="/sender"/>
        </copy>
    </assign>
    <assign>
        <copy>
            <from variable="request" part="SMS" query="/SMS/message"/>
            <to variable="sendRequest" part="message" query="/message"/>
        </copy>
    </assign>
    <scope variableAccessSerializable="no">
        <faultHandlers>
            <catchAll>
                <throw faultName="tns:sendFailed"/>
            </catchAll>
        </faultHandlers>
        <invoke partnerLink="SendMessageService" portType="send:SendMessageService" operation="send"
                                                inputVariable="sendRequest" outputVariable="sendResponse"/>
    </scope>

    <!-- reply true to client -->
    <assign>
        <copy>
            <from expression="'true'"/>
            <to variable="response" part="result" query="/result"/>
        </copy>
    </assign>
    <reply partnerLink="client" portType="tns:SendSMS" operation="send" variable="response"/>

    <!-- invoke DeliveryNotificationService -->
    <assign>
        <copy>
            <from variable="sendResponse" part="sendReturn" query="/sendReturn"/>
            <to variable="registerNotification" part="messageIdentifier" query="/messageIdentifier"/>
        </copy>
    </assign>
    <invoke partnerLink="DeliveryNotificationService" portType="not:DeliveryNotificationService" operation="register"
                                                                                inputVariable="registerNotification"/>

    <!-- wait for notification or timeout -->
    <pick>
        <!-- wait for message... -->
        <onMessage partnerLink="DeliveryNotificationService" portType="not:DeliveryNotificationServiceCallback"
                                                                        operation="notification" variable="notification">
            <empty/>
        </onMessage>
        <!-- ...or assign false to result after 1 minute -->
        <onAlarm for="PT1M">
            <assign>
                <copy>
                    <from expression="'false'"/>
                    <to variable="notification" part="result" query="/result"/>
                </copy>
            </assign>
        </onAlarm>
    </pick>

    <!-- invoke ChargeService if notification reports success -->
    <switch>
        <case condition="bpws:getVariableData('notification','result')='true'">
            <!-- invoke ChargeService -->
            <sequence>
                <assign>
                    <copy>
                        <from variable="request" part="SMS" query="/SMS/sender"/>
                        <to variable="chargeRequest" part="who"/>
                    </copy>
                </assign>
                <assign>
                    <copy>
                        <from expression="'0.16'"/>
                        <to variable="chargeRequest" part="amount" query="/amount"/>
                    </copy>
                </assign>
                <invoke partnerLink="ChargeService" portType="charge:ChargeService" operation="charge"
```

```
                                                                    inputVariable="chargeRequest" outputVariable="chargeResponse"/>
                        </sequence>
                    </case>
                    <otherwise>
                        <!-- log the failure of the notification -->
                        <sequence>
                            <assign>
                                <copy>
                                    <from expression="'Notification: send failed, no charging'"/>
                                    <to variable="logRequest" part="event" query="/event"/>
                                </copy>
                            </assign>
                            <invoke partnerLink="LogService" portType="log:LogService" operation="writeToLog"
                                                                    inputVariable="logRequest" outputVariable="logResponse"/>
                        </sequence>
                    </otherwise>
                </switch>

            </sequence>
        </scope>
</process>
```

Appendix 2: SendSMS WSDL

Below is the WSDL document of the SendSMS BPEL process. This is not the complete WSDL document, but only the abstract description. Bindings and the endpoint are added when the SendSMS process is deployed on the Collaxa BPEL server.

```
<definitions name="SendSMS"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="urn:SendSMS"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:SendSMS">

    <types>
        <schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="urn:SendSMS"
                                                                                xmlns="http://www.w3.org/2001/XMLSchema">
            <element name="SMS">
                <complexType>
                    <sequence>
                        <element name="sender" type="string"/>
                        <element name="phoneNumber" type="string"/>
                        <element name="message" type="string"/>
                    </sequence>
                </complexType>
            </element>
        </schema>
    </types>

    <message name="SendSMSRequest">
        <part name="SMS" element="tns:SMS"/>
    </message>
    <message name="SendSMSResponse">
        <part name="result" type="xsd:boolean"/>
    </message>
    <message name="errorMessage">
        <part name="description" type="xsd:string"/>
    </message>

    <portType name="SendSMS">
        <operation name="send">
            <input message="tns:SendSMSRequest"/>
            <output message="tns:SendSMSResponse"/>
            <fault name="sendFailed" message="tns:errorMessage"/>
            <fault name="userSignedOptOut" message="tns:errorMessage"/>
            <fault name="unknownException" message="tns:errorMessage"/>
        </operation>
    </portType>

    <plnk:partnerLinkType name="SendSMSLink">
        <plnk:role name="service">
            <plnk:portType name="tns:SendSMS"/>
        </plnk:role>
    </plnk:partnerLinkType>

</definitions>
```

Appendix 3: How to install the software

This appendix is provided as a guide to installing the software needed for the prototype and deploying the prototype and all other web services.

The software has been tested on Microsoft Windows 2000 and XP with a recent Sun Java SDK (1.4.2). Since the total setup consists of two servers running at the same time, it is advisable to have at least 512 MB RAM installed.

The report comes with a CD that contains the software. There are a directory Software that contains the server software and a few BPEL editors, and a Webservices directory that contains the web services and the BPEL processes.

First install the base and auxiliary services platform by installing Apache Tomcat 4.1 and Apache Axis 1.1. The prototype expects Tomcat to be installed on port 80. If you install Tomcat on another port you need to change some of the WSDL files and deployment descriptors to include the right port number. Please verify that Axis has been installed properly by validating you Axis installation with the Axis Happiness Page. Then make sure Tomcat is running and execute the deploy-webservices.bat file in the Webservices directory on the CD to deploy the web services on Axis. You may need to change some paths in the batch file to match your local configuration.

Since the Collaxa BPEL Server comes as an all-in-one package it is very easy to install. Once you have installed the Collaxa BPEL Server you can deploy the BPEL processes by executing deploy-DeliveryNotificationService.bat and deploy-SendSMS.bat. Make sure you first deploy the DeliveryNotificationService because the SendSMS process needs this service.

Now you are set up to execute the process using the BPEL Console on http://localhost:9700/BPELConsole or use an external tool. A link to the WSDL of the SendSMS process can be found in the BPEL Console.