INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

Diplomarbeit

Serving Xcerpt to the Web

Clemens Schefels

Aufgabensteller:	Prof. Dr. François Bry
Betreuer:	Dipl. Inf. Tim Furche
	Dipl. Inf. Benedikt Linse
Abgabetermin:	11. August 2006

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

München, den 11. August 2006 Clemens Schefels

Abstract

Web Services are a new, increasingly common program design and communication paradigm, that is based on the SOA architecture. The Web Service approach introduces a large number of new concepts, though often recasting familiar solutions. The SOAP-based WS-* stack provides a powerful plattform for Web Service development and deployment by extending the SOAP protocol to an all-purpose tool for Web Services. This thesis outlines in the first part the core technologies of Web Services and introduces the WS-* stack.

Xcerpt is a declarative XML query and transformation language based on logic programming concepts. Its declarative approach stands in contrast to the imperative request-response nature of Web Services, which can be seen as a procedure call. In the second part of the thesis the integration of Web Service calls in Xcerpt is in focus. First different Java SOAP stacks are compared and one is chosen for the realization of the Web Service calls. Next the expected and experienced problems of the integration of an imperative concept in a declarative programming language are analyzed and solutions proposed. Finally, a proof-of-concept implementation of Web Service calls from Xcerpt is shortly described.

Zusammenfassung

Web Services, ein neues Programmierkonzept und Kommunikationsparadigma, das auf dem SOA Prinzip beruht, erfreut sich immer größerer Beliebtheit. Mit Web Services werden eine große Anzahl von neuen Konzepten eingeführt, die oft an altbekannte Lösungen erinnern. Der SOAP basierte WS-* Stack bietet eine mächtige Plattform für die Entwicklung und den Betrieb von Web Services, da er das SOAP Protokoll zu einem allzweck Web Service Werkzeug erweitert. Diese Diplomarbeit faßt im ersten Teil die Kerntechnologien der Web Services zusammen und führt den WS-* Stack ein.

Xcerpt ist eine deklarative XML Anfrage- und Transformationssprache, die auf logischen Programmierkonzepten beruht. Der deklarative Ansatz von Xcerpt steht im Gegensatz zu der imperativen "request-response" Natur der Web Services. Web Services können sogar als Prozeduraufrufe betrachtet werden. Im zweiten Teil der Arbeit steht die Integration von Web-Service-Aufrufen in Xcerpt im Vordergrund. Zuerst werden verschiedene Java SOAP Stacks miteinander verglichen, um einen schließlich für die Realisierung der Web-Service-Aufrufe auszuwählen. Als nächstes werden die zu erwartenden und beobachteten Probleme der Integration vom imperativen Konzept in eine deklarative Programmiersprache analysiert und Lösungen vorgeschlagen. Zum Abschluß wird kurz eine beispielhafte Implementierung der Web-Service-Aufrufe in Xcerpt beschrieben.

Acknowledgments

At first I like to thank all members of the “Programmier- und Modellierungssprachen” unit for the support and the pleasing working atmosphere. I like express my gratitude to

- *Prof. Dr. François Bry* for the motivating words, creative discussions and the fruitful meetings,
- *Tim Furche* and *Benedikt Linse* for the great supervision and support during the whole time,
- *Michael Eckert* for great suggestions that helped in writing this thesis.

At last I want to thank my family, especially my parents, that provided me with the possibility to study and supported me during the entire time of my studies.

CONTENTS

CHAPTER

ONE

Introduction

1.1 Introduction

Web Services are the next evolutional step of the Internet. Like Web browsers have changed the Internet for human users, Web Services bear the promise to change the Internet for computers. With Web browsers the information of the Internet is visualized only for humans, but not made accessible for computers. With the Semantic Web there are indeed some trails to help the computer to understand the data of the Internet, but the results are disillusioning. With Web Services computers will be able to exchange information over the de-centrally organized Internet over the whole world with other computers, like if they were directly connected.

But even today Web Services are needed, because the growing popularity and the commercialization of the World Wide Web change the requirements on the existing Web technologies. Not only static Web pages are requested, but more and more complex online shops, booking services, and other service providers use the Internet for professional trading. Not all existing technologies are designed for these complex tasks, new solutions are often needed. E.g., existing technologies such as HTML and HTTP provide only insufficient means for transmitting computer-processable data of different types. The Web Service approach promises to address some (or most) of these problems reusing where possible existing and established technologies, e.g., for the modular layout, language and platform independence. At the core of the Web Service approach stands the still developing, SOAP-based WS-* stack. It provides basic and increasingly also advanced messaging and remote procedure call functionality. It uses the extensibility of XML to provide a wide collection of SOAP extensions for different tasks like security, reliable messaging and transaction coordination.

XML is an important element in Web-Service related standards and in the WS-* stack. It is used, e.g., to represent exchanged messages in many remote procedure call protocols like SOAP or XML-RPC. Even WSDL, the Web Service definition language, uses XML to describes the function signature of a Web Service. The use of XML as an extensible, standardized and widely supported markup language allows the Web Service stack to become relatively independent of operating systems or even programming languages and is seen as an argument why the Web Service approach is better suited to the data-centric, distributed World Wide Web than previous remote procedure frameworks.

Hence, the choice to use an XML query and transformation language to handle Web

Service data seems obvious. Xcerpt[1] is a logic-based XML query and transformation language. It is developed at the teaching and research unit "Programming and Modelling Languages[2]" of the University of Munich[3]. The declarative paradigm of Xcerpt makes it possible to specify the query intent of the user very naturally. In the context of Web Services, such declarative query intent specifications are very much needed, but the combination with the inherently iterative, request-response style nature of Web Services proves to be challenging. Typical language design goals of logic based languages are not obvious to connect with iterative, request-response Web Services and provide a fascinating research field.

This thesis examines the core technologies of Web Services with focus on the problems of integrating Web Service calls into Xcerpt. For example, it is difficult to keep the design goal of referential transparency, which postulates that every expression has the same meaning in the same scope, in Xcerpt, if Web Service results can differ from one call to the other and the time and number of calls to a specific Web Service is not under direct control of the query programmer but rather left to the query engine (e.g., for efficiency purposes or to hide unnecessary system complexity). Even sequences of Web Services calls are a challenge, because Xcerpt provides no explicit means for the query author to control the evaluation order of data access statements. If the output of one Web Service is used as input for another the range restriction principle of Xcerpt has to be extended to make the evaluation possible and to prevent deadlocks. Some solutions for these problems are explored and recommendations for an implementation as well as a proof-of-concept implementation of Web Service calls in Xcerpt is given.

The **main contributions** of this thesis are as follows:

1. A detailed overview and comparison of both basic *Web Service technologies* and of the extended WS-* stack.

2. A consideration and comparison of the most prominent *Java libraries* that implement (part of) the WS-* stack.

3. A *discussion of the problems* involved when integrating a declarative query language such as Xcerpt with request-response style Web Services.

4. Solutions to two of the most prominent problems: (a) to *ensuring consistency over multiple successive calls* to the same Web Services due to the evaluation model used by the language and (b) to the *automatic determination of a proper, deadlock free evaluation order* of multiple Web Service calls in one rule or program.

5. An *extension of classic range restriction* is proposed that ensures sequential, deadlock-free evaluation of Web Service calls in range restricted programs. The extended property remains a static property of Xcerpt programs that is easy to verify at compile-time.

[1] http://www.xcerpt.org/
[2] http://www.pms.ifi.lmu.de/
[3] http://uni-muenchen.de/

1.2 Outline of this Thesis

This thesis consists of seven chapters. Chapter 2 introduces the main technologies of Web Services. It starts with a short introduction about the SOA (Service Oriented Architecture) principle extended by Web Services. Next the WSDL (Web Service Description Language), a language designed to define the functionalities of Web Services, and UDDI (Universal Description, Discovery and Integration), a registration service, are described. Finally, an overview of the most common Web Service messaging and remote procedure call protocols is given.

In Chapter 3, the SOAP stack is discussed with its WS-* (spoken: WS-Star) extentions. The focus lies on the Description and Discovery layer and on the Quality of Service layer of this stack.

Chapter 4 compares the different Java SOAP stacks which are needed for the implementation of Web Service functionalities into Xcerpt. At the end of this chapter, a Java SOAP stack is chosen for the implementation and some experiences with this framework are given.

The XML query and transformations language Xcerpt is introduced in Chapter 5. Furthermore the integration of Web Services calls in Xcerpt is described in Chapter 5 and all problems with its solutions are discussed. This chapter is the core of the thesis.

Chapter 6 provides a short summary about interesting research papers related to the topic of this thesis.

The last, Chapter 7, highlights some ideas how Xcerpt could be extended with other Web Service technologies.

The appendix A includes the Java source code of the test implementation of Web Services calls for the Xcerpt abstract machine AMaχoS[4].

[4] http://www.amachos.com

CHAPTER
TWO

Web Services

2.1 Motivation

Web Services are a new trend in Internet applications and software development nowadays. Every major IT company deploys a Web Services based application or open Web Services for public, like eBay[1], Amazon[2], Google[3], or Microsoft[4]. But if you compare these services you may not really get an idea of what a Web Service is: often very different technologies are used. This chapter summarizes all the main characteristics of Web Services and introduces the core technologies, which are most often used with Web Services.

2.2 Web Service

In a few words, a *Web Service* [CJ02] is a software application identified by a URI (or **U**niform **R**esource **I**dentifier), which is connected via a network to clients. But let us take a closer look:

The main concept behind Web Services is the so-called *SOA*—*S*ervice *O*riented *A*rchitecture [SOA06], where an application is no longer a large monolithic program, but it is partitioned into smaller, loosely coupled programs. These programs can be classified in *providers* (or servers), programs that contribute a service to another software component, and *consumers* (or clients), programs that request services from the providers. Provided *services* are loosely coupled together with standardized or at least well-defined interfaces. These loosely coupled programs make the architecture very extensible, due to the possibility to add or remove services with limited costs. Thus new services can be created by combining existing services. Also single services in this modular architecture are assumed to be easy to reuse and to maintain, because they have well-defined interfaces and can thus be tested separately from the rest of the program, quite in contrast to a monolithic program design.

Web Services extend the SOA principle [CJ02][HZ03]: A Web Service must have its own URI, must be accessible via a network, often the Internet, and should be platform independent. The interoperability of Web Services results from the fact that programming language

[1] http://www.ebay.com/
[2] http://www.amazon.com/
[3] http://www.google.com/
[4] http://www.microsoft.com/

independent protocols are used for the message exchanging. In this way, it is possible to implement a Web Service provider (or server) in a programming language, but the clients for the same service in another programming language. This makes combining already existing Web Services into new ones much easier—a trend currently called *mashup*, though "mashup" in the wider sense is understood as the combination of data from different sources regardless of whether a Web Service is involved or not.

In the next sections other techniques will be introduced that are related to Web Services. These techniques are often mentioned in the same breath with Web Services and belong therefore to the Web Service concept. WSDL is the definition language for Web Services. With this language it is possible to describe the functionality of a Web Service in a abstract form that is readable for computer. To find a services the UDDI concept provides a registration service where Web Service can be described and discovered with a SOAP interface. For the communication between Web Service and client or between Web Services among themselves a wide variety of messaging protocols are available. The most common protocol, which provides the best functionality, is the SOAP protocol. This functionality is reach with several extentions, described in the WS-* standards, the so-called SOAP stack. The XML-RPC protocol is very similar to SOAP, it could be seen as its predecessor, but XML-RPC is very restrictive. With REST a simple and very common protocol is introduced, rather a design principle of the Internet, which uses only the functionality of the HTTP protocol to get access to the functionality of Web Services. The last protocol is GData, a proprietary development of Google, which is very similar to REST, but uses Atom feeds to exchange data.

2.3 WSDL

WSDL [CJ02][HZ03][MH04]—**W**eb **S**ervice **D**efinition **L**anguage—is a simple way to describe important information that are necessary to communicate with a Web Service. Functionally, a WSDL file is comparable to interface definitions of other component architectures, e.g., C Header Files, Java interfaces or CORBA IDL (Interface Definition Language) file. The WSDL file is in XML format, so it is simple to generate WSDL automatically, or process and extract the information. It provides several XML-tags to describe the type definitions (relying on XML Schema), the anatomy of a message, the function signature, and the address of the Web Service Endpoint.

Furthermore other WSDL-documents can be imported and a human-readable documentation can be included to make the file easier to understand. Listing 2.1 shows the abbreviated WSDL file[5] form the Amazon[6] book shop Web Service. Since 15 March 2001, WSDL is a W3C[7] recommendation, version 2.0 has W3C Candidate Recommendation status.

[5] http://webservices.amazon.com/AWSSimpleQueueService/AWSSimpleQueueService.wsdl
[6] http://www.amazon.com/
[7] http://www.w3.org/

Listing 2.1: Amazon WSDL example

```

<wsdl:definitions xmlns:typens="http://soap.amazon.com"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"   ... >
  <wsdl:types>
    <xsd:schema xmlns=""
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://soap.amazon.com">
                <!-- ... -->
      <xsd:complexType name="ProductInfo">
        <xsd:all>
          <xsd:element name="TotalResults" type="xsd:string" minOccurs="0"/>
          <xsd:element name="TotalPages"   type="xsd:string" minOccurs="0"/>
          <xsd:element name="ListName"     type="xsd:string" minOccurs="0"/>
          <xsd:element name="Details"      type="typens:DetailsArray" minOccurs="0"/>
        </xsd:all>
      </xsd:complexType>
      <xsd:complexType name="ArtistRequest">
        <xsd:all>
          <xsd:element name="artist" type="xsd:string"/>
          <!-- ... -->
        </xsd:all>
      </xsd:complexType>
      <xsd:complexType name="AuthorRequest">
        <!-- ... -->
      </xsd:complexType>
      <!-- ... -->
    </xsd:schema>
  </wsdl:types>
  <!-- ... -->
  <message name="AuthorSearchRequest">
    <part name="AuthorSearchRequest" type="typens:AuthorRequest"/>
  </message>
  <message name="AuthorSearchResponse">
    <part name="return" type="typens:ProductInfo"/>
  </message>
  <message name="ArtistSearchRequest">
    <part name="ArtistSearchRequest" type="typens:ArtistRequest"/>
  </message>
  <message name="ArtistSearchResponse">
    <part name="return" type="typens:ProductInfo"/>
  </message>
    <!-- ... -->
  <portType name="AmazonSearchPort">
    <!-- ... -->
    <operation name="AuthorSearchRequest">
      <input message="typens:AuthorSearchRequest"/>
      <output message="typens:AuthorSearchResponse"/>
    </operation>
    <operation name="ArtistSearchRequest">
      <input message="typens:ArtistSearchRequest"/>
      <output message="typens:ArtistSearchResponse"/>
    </operation>
    <!-- ... -->
  </portType>
  <binding name="AmazonSearchBinding" type="typens:AmazonSearchPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <!-- ... -->
    <operation name="AuthorSearchRequest">
      <soap:operation soapAction="http://soap.amazon.com"/>
      <input>
        <soap:body use="encoded"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   namespace="http://soap.amazon.com"/>
      </input>
      <output>
        <soap:body use="encoded"
                   encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   namespace="http://soap.amazon.com"/>
      </output>
    </operation>
    <operation name="ArtistSearchRequest">
      <soap:operation soapAction="http://soap.amazon.com"/>
      <input>
        <!-- ... -->
      </input>
      <output>
        <!-- ... -->
      </output>
    </operation>
    <!-- ... -->
  </binding>
  <service name="AmazonSearchService">
    <port name="AmazonSearchPort" binding="typens:AmazonSearchBinding">
      <soap:address location="http://soap.amazon.com/onca/soap2"/>
    </port>
  </service>
</wsdl:definitions>
```

The above example describes the Amazon search service, that provides an interface to the search functionality of the Amazon bookstore. The WSDL file consists of

1. (lines 5-28) the types tag which defines the data types (e.g. "ProductInfo") using XML Schema type definitions (e.g. "xsd::string").

2. (lines 30-41) the message tags which describes the exchanged data. The used data types are specified in the part tags.

3. (lines 43-54) the "portType" tag which encloses the operations that can be executed on a Web Service. The input and output tags define the exchanged messages. This is a very abstract presentation.

4. (lines 55-81) the binding tag with the details of the used transport protocol (e.g. HTTP, SMTP) and data formats for every operation in the portType tag.

5. (lines 82-86) service tag which describes the endpoints. Each endpoint (port tag) is a collection of ports and bindings that are defined above.

2.4 UDDI

Universal **D**escription and **D**iscovery **I**nterface, *UDDI*, is a register and search service for Web Services [CJ02][HZ03]. It stores information about the functions, technical facts and data about the vendor of a Web Service. The UDDI registry is a Web Service itself with a SOAP interface. It is possible to register and search for a Web Service by means of the above mentioned information. The registered information is grouped as follows:

- *White pages*: Contain the basic contact information about the vendor like name, address, tax ID. It can be seen as a phone book for Web Services.

- *Yellow pages*: Consist of categorizations of Web Services. The name indicates the analogy to the yellow pages, the business register.

- *Green pages*: Provide the technical information about the Web Service.

Technical information, mostly the location of the WSDL file, about a Web Service are stored in a separated file too, called *tModel* document.

The structure of UDDI is similar to the well-known *Domain Name System*[8] (DNS) of the Internet: The *UDDI Business Register* (UBR) is organized into single *UDDI nodes*. Each node stores the same information and synchronizes this data with the other nodes. If one UDDI node is updated, all other nodes propagate this new information. But it is also possible to run a private UDDI node that contains data about Web Services that are not public. This non-public information will not be shared with the other UDDI nodes, see Figure 2.1. There are also Web-based UDDI browsers, cf. figure 2.2.

The concept of UDDI reminds of Web search engines like Google and Yahoo![9]. But there are some differences:

[8] http://tools.ietf.org/html/rfc1034
[9] http://www.yahoo.com/

- In UDDI Web Services have to be explicite registered, while Web search engines “crawl” the World Wide Web with “robots” and collect the data needed.
- The information about a Web Service in the UDDI have a given format, defined in the UDDI specification. For Web search engines no such a format exists, so that every search engine collects different data of the Web sites.
- UDDI only present the register Web Service data, while Web search engines interpret the collected data and order them according to their relevance.

UDDI is an OASIS[10] standard since version 2.0 and has nowadays reached version 3.0.

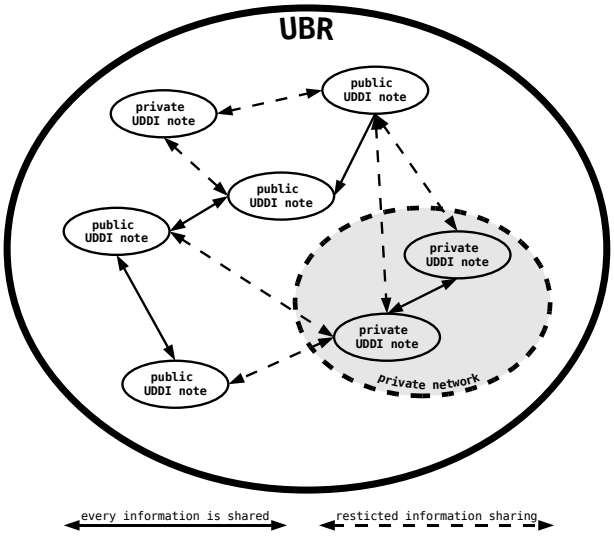

Figure 2.1: UBR with UDDI nodes

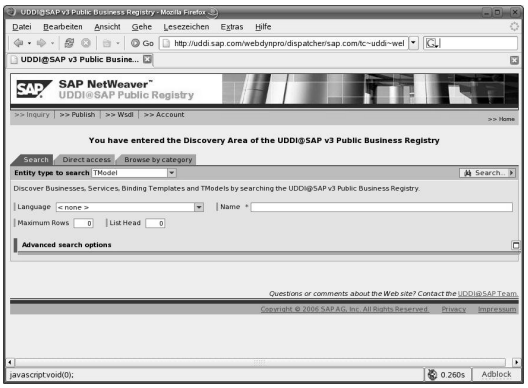

Figure 2.2: SAP UDDI service

[10]http://www.oasis-open.org/

2.5 Remote Procedure Call Protocols

With the development of computer networks, *distributed computing* became increasingly important. An application cannot only be installed on the local computer, but programs can be distributed on all computers that are connected to the same network. With *remote procedure call* (RPC) protocols it is possible to call methods of remote programs transparently, i.e., as if they are on the local computer—the software developer can use remote programs like local ones. But many programming languages and operating systems provides their own RPC protocol which are mostly incompatible among each other. Well established kinds of RPC protocols are the ISO RPC[11], CORBA[12] and RMI[13]. But Web Services have new requirements on RPC protocols like language independence and interoperability. CORBA may fulfill these requirements, but its object-oriented approach and its constraint to the IIOP protocol, are often not considered suitable to Web Service and a heterogeneous, open distributed environment.

This section will introduce the most common Web Service RPCs mechanism and conclude with a comparison.

2.5.1 SOAP

The **S**imple **O**bject **A**ccess **P**rotocol (or **S**ervice **O**riented **A**rchitecture **P**rotocol)—*SOAP*—is a very common XML messaging and RPC [SOA03] protocol. The anatomy of a SOAP message is very simple and easy to extend [CJ02][HZ03]:

A SOAP message is surrounded with an <Envelope>-tag which contains the namespaces as attributes. The following <Header>-tag is *optional* and provides a convenient space for extensions or individual tags. The WS-* standards (cf. Chapter 3) often make use of the optional header to enrich SOAP with additional tags. This facility makes SOAP one of the most interesting and flexible Web Service protocols. Finally, the actual RPC payload (e.g. function-parameter) is placed in the <Body>-tag and completes the message.

SOAP is a stateless protocol. Web Services with stateful communication have to provide their own state control mechanism. SOAP provides asynchronous (one-way) communication as well as synchronous (request-response) communication with the participants. The so-called *"document" style* describes the asynchronous communication. The name of this style is justified by the fact that only "document" data should be transmitted, no function parameters. So this style allows a relatively flexible design of the SOAP body, no restrictions are made. The *"RPC" style* defines the synchronous communication. With this style basically just function parameters are exchanged. The design of the SOAP body is stricter than in the document style, because the body has to start with the element which is named like the called method. The following subelements describes the function parameter of the method each with the parameter value.

One advantage of SOAP is its independence from the transport protocol. It is possible to use nearly every protocol in combination with SOAP. But often, common Internet protocols are used, like HTTP, because most firewalls are permeable to HTTP and must not be reconfigured to allow a communication with Web Services.

[11] http://whatis.techtarget.com/definition/0,,sid9_gci214016,00.html
[12] http://www.omg.org/corba/
[13] http://java.sun.com/products/jdk/rmi/

With SOAP it is also possible to attach files so that binary files must not be converted into XML with, e.g., base64 encoding, which would lead to a significant overhead. To attach files, SOAP uses the very common MIME[14] (Multipurpose Internet Mail Extensions) mechanism.

If an error occurs [MH04], the participants can send a SOAP message with a single <Fault>-tag in the body to the sender. With several subelements the fault can be specified, and should be used to develop a meaningful error handling.

SOAP supports simple Web Services as well as complex business Web Services in addition to WS-* standards.

Because of its wide application area many implementations in many established programming language, like Java, C, C++, or Perl, are available and make SOAP more interoperable.

Listing 2.2 shows a SOAP request to the Amazon Web Service and queries the book "Java Web Services in a Nutshell" in the book index. The received response is given in Listing 2.3.

SOAP is a W3C standard since 2000, version 1.2 is a recommendation of the W3C from the XML Protocol Working Group[15].

Listing 2.2: Amazon SOAP Request

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <SOAP-ENV:Envelope
3   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
4   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
5   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
6   xmlns:xsd="http://www.w3.org/1999/XMLSchema"
7   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
8    <SOAP-ENV:Body>
9      <ItemSearch xmlns="">
10       <SubscriptionId xsi:type="xsd:string">123</SubscriptionId>
11       <Request>
12         <Keywords xsi:type="xsd:string">
13           Java Web Services in a Nutshell
14         </Keywords>
15         <SearchIndex xsi:type="xsd:string">Books</SearchIndex>
16       </Request>
17     </ItemSearch>
18   </SOAP-ENV:Body>
19 </SOAP-ENV:Envelope>
```

[14] http://www.ietf.org/rfc/rfc2387.txt
[15] http://www.w3.org/2000/xp/Group/

Listing 2.3: Amazon SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ItemSearchResponse
     xmlns="http://webservices.amazon.com/
                    AWSECommerceService/2005-10-05">
      <OperationRequest>
        <HTTPHeaders>
          <Header Name="UserAgent" Value="SOAP::Lite/Perl/0.60">
          </Header>
        </HTTPHeaders>
        <RequestId>123</RequestId>
        <Arguments>
          <Argument Name="Service" Value="AWSECommerceService">
          </Argument>
        </Arguments>
        <RequestProcessingTime>
          0.0481729507446289
        </RequestProcessingTime>
      </OperationRequest>
      <Items>
        <Request>
          <IsValid>True</IsValid>
          <ItemSearchRequest>
            <Keywords>Java Web Services in a Nutshell</Keywords>
            <SearchIndex>Books</SearchIndex>
          </ItemSearchRequest>
        </Request>
        <TotalResults>1</TotalResults>
        <TotalPages>1</TotalPages>
        <Item>
          <ASIN>0596003994</ASIN>
          <DetailPageURL>
            http://www.amazon.com/exec/obidos/redirect?...
          </DetailPageURL>
          <ItemAttributes>
            <Author>Kim Topley</Author>
            <Manufacturer>O'Reilly Media, Inc.</Manufacturer>
            <ProductGroup>Book</ProductGroup>
            <Title>Java Web Services in a Nutshell</Title>
          </ItemAttributes>
        </Item>
      </Items>
    </ItemSearchResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The examples above describes a request (listing 2.2) to the Amazon Web Service and its response (listing 2.3), both SOAP messages:

1. The SOAP envelope contains the namespace definitions and surround the whole SOAP message (request: line 2-7, response: line 2-6).

2. The SOAP body includes the message or the remote procedure call for the Web Service (request: line 8-6, response: line 7-46). The single tags in the body are Web Service specific and can be defined in a WSDL file.

2.5.2 XML-RPC

Another common XML based RPC protocol is *XML-RPC*[16] developed by Microsoft and Dave Winer, who still overlooks the development of XML-RPC today.

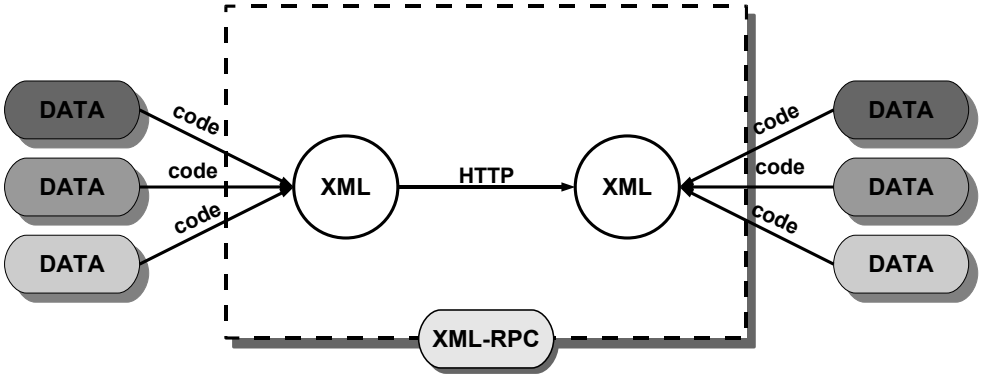

Figure 2.3: XML-RPC diagram from http://www.xmlrpc.com/

An XML-RPC message [Win99] consists of an HTTP POST request with (type and parameters of) the RPC as XML payload. It is very similar to SOAP and indicates, that XML-RPC can be seen as antecessor of SOAP. However, where SOAP uses an XML header as part of the HTTP payload, XML-RPC uses the HTTP header to specify certain metadata such as the Content-Type (text/xml) and the correct Content-Length. The XML payload is rather straightforward as well:

The <MethodCall> tag preludes the XML request message. It contains the <MethodName> item with the name of the called method encoded as a string. The next subelement is the <params> tag with the method parameter in the tags <param>. A single <param> tag element surrounds a <value> element with the data type information. Both, simple types and complex types are possible. <Struct> and <array> elements are allowed as complex data types. <Struct> consist of <member> tags with a <name> (String) and a <value> element. An <array> has a single <data> tag which surrounds any number of <value> elements. The response message of the Web Service consists of a <methodResponse> tag with a single <params> tag. Faults are a special response message type containing <value>-elements with a <struct>. This <struct> has two <member>-elements with <name>-elements "faultcode" and "faultstring".

[16] http://www.xmlrpc.com/

One goal of the XML-RPC standard is to be "as simple as possible"[17]:

- With the easily understandable structure even a non-developer can examine the RPC message and possibly edit it.
- Firewalls must not be reconfigured because of the use of HTTP (similar to SOAP section 2.5.1).
- It is easy to implement and easy to be adapted to new platforms or programming languages.

XML-RPC is mostly used for simple, not very complex Web Services. It has not such a high interoperability as SOAP, because the transport protocol can not changed, only HTTP can be used [18]. The easiness of the structure, in comparison to the complex SOAP messages with its namespaces and encoding, makes fast parsing possible. Nowadays XML-RPC is a very common Web Blog messaging protocol and enjoys wide-adoption in that area. XML-RPC is a final standard, no new releases are expected to be published in the next time.

Listing 2.4: XML-RPC Request

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version='1.0' encoding='UTF-8'?>
<methodCall>
  <methodName>flickr.echo</methodName>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>Echo</name>
            <value><string>Hello</string></value>
          </member>
          <member>
            <name>api_key</name>
            <value><string>78ce850f064baa9e7fc</string></value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

[17] http://www.xmlrpc.com/

[18] There are some projects that provide libraries to use XML-RPC with other transport protocols like JABBER (http://www.pipetree.com/jabber/XMLRPC/) or SMTP , but these projects are not official supported by Dave Winner.

Listing 2.5: XML-RPC Response

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <methodResponse>
3    <params>
4      <param>
5        <value>
6          <string>
7            <method>flickr.test.echo</method>
8            <format>xmlrpc</format>
9            <Echo>Hello</Echo>
10           <api_key>78ce850f064baa9e7fc</api_key>
11         </string>
12       </value>
13     </param>
14   </params>
15 </methodResponse>
```

The examples above show a communication between a client and the flickr Web Service. The client sends a request to the "test.echo" port which only request with a message that contains the received data. The request (listing 2.4):

1. Line 1-5 contain the HTTP header with the information about the "Content-type" and the host.
2. Line 8 lists the "MethodCall" tag that surrounds the message. It is very similar to the SOAP envelope.
3. Line 9 defines the name of the method that should executed on the Web Service.
4. Line 10-25 contain the remote procedure call. The function parameter ("Echo", "api_key") are defined as their data type (struct) with the parameter value ("Hello", "78ce850f064baa9e7fc").

The response (listing 2.5) is similar to the request. An explanation would be trivial.

2.5.3 REST

REST, **Re**presentational **S**tate **T**ransfer, is not a real RPC protocol. REST is an "architectural style for distributed hyper-media systems" [Fie00] described by Roy Fielding in his PhD thesis with the title "Architectural Styles and the Design of Network-based Software Architectures" [Fie00]. In this section, only aspects of REST that are relevant to Web Service are discussed:

The communication in a RESTful world is stateless, every message must contain all necessary information and only clients can store the states, server side cookies are prohibited. To communicate with a Web Service only the four functions of the HTTP protocol are available:

- *GET* requests the actual state of a resource and has to be side effect-free.
- *POST* enhances a resource with new information or changes existing information.
- *PUT* creates a new resource.
- *DELETE* erases a resource.

With these four operations it is possible to deploy simple Web Services. The REST model uses only approved and common techniques. It does not extend the HTTP protocol with an XML message payload so that the overhead of a REST message is minimal. Another advantage of the use of the HTTP protocol is that this protocol is very scalable as one can easily see on the continuously growing Internet, which started as a small prototypical network.

In many ways, REST can be seen as the description of how resource and service access with HTTP is intended to be used to ensure scalable information exchange in a distributed, global network that may include many layers of intermediary systems such as proxies and caches.

Like with SOAP, no reconfiguration of most of the firewall servers is necessary and even simple proxy servers can examine the REST messages including the RPC-data and control, maybe restrict the communication, in contrast to XML-based RPC, which can only be interpreted by proxies with the possibility to parse XML message parts.

The use of REST is simple and mostly adequate for simple Web Services. Accordingly REST enjoys a great popularity: Until 2003 85% of the requests to the Amazon Web Service, which provides SOAP and REST interface, are REST based [O'R03]. Listing 2.6 and listing 2.7 show such a communication with the Amazon REST interface.

Listing 2.6: Amazon REST Request

```
GET /onca/xml?Service=AWSECommerceService&
              AWSAccessKeyId=123&
              Operation=ItemSearch&
              Keywords=Java%20Web%20Services%20in%20a%20Nutshell&
              SearchIndex=Books&
              ResponseGroup=Request,Small&
              Version=2005-10-13

TE: deflate,gzip;q=0.3
Connection: TE, close
Host: webservices.amazon.com
User-Agent: LWP::Simple/5.803
```

Listing 2.7: Amazon REST Response

```
1  TTP/1.1 200 OK
2  Date: Tue, 20 Jun 2006 13:24:00 GMT
3  Server: Server
4  x-amz-id-1: 038HF55JKGS3P6GS797B
5  x-amz-id-2: fLV3wBLkmP72BYvNryzwnyr3pNT4vYoc
6  Connection: close
7  Transfer-Encoding: chunked
8  Content-Type: text/xml; charset=UTF-8
9
10 <?xml version="1.0" encoding="UTF-8"?>
11 <ItemSearchResponse xmlns="http://webservices.amazon.com/AWSECommerceService/2005-10-13">
12   <OperationRequest>
13     <HTTPHeaders>
14       <Header Name="UserAgent" Value="LWP::Simple/5.803"></Header>
15     </HTTPHeaders>
16     <RequestId>038HF55JKGS3P6GS797B</RequestId>
17     <Arguments>
18       <Argument Name="Keywords" Value="Java Web Services in a Nutshell"></Argument>
19       <Argument Name="ResponseGroup" Value="Request,Small"></Argument>
20       <Argument Name="Operation" Value="ItemSearch"></Argument>
21       <Argument Name="Service" Value="AWSECommerceService"></Argument>
22       <Argument Name="AWSAccessKeyId" Value="123"></Argument>
23       <Argument Name="SearchIndex" Value="Books"></Argument>
24       <Argument Name="Version" Value="2005-10-13"></Argument>
25     </Arguments>
26     <RequestProcessingTime>0.0682680606842041</RequestProcessingTime>
27   </OperationRequest>
28   <Items>
29     <Request>
30       <IsValid>True</IsValid>
31       <ItemSearchRequest>
32         <Keywords>Java Web Services in a Nutshell</Keywords>
33         <ResponseGroup>Request</ResponseGroup>
34         <ResponseGroup>Small</ResponseGroup>
35         <SearchIndex>Books</SearchIndex>
36       </ItemSearchRequest>
37     </Request>
38     <TotalResults>1</TotalResults>
39     <TotalPages>1</TotalPages>
40     <Item>
41       <ASIN>0596003994</ASIN>
42       <DetailPageURL>http://www.amazon.com/exec/obidos/redirect?...</DetailPageURL>
43       <ItemAttributes>
44         <Author>Kim Topley</Author>
45         <Manufacturer>O'Reilly Media, Inc.</Manufacturer>
46         <ProductGroup>Book</ProductGroup>
47         <Title>Java Web Services in a Nutshell</Title>
48       </ItemAttributes>
49     </Item>
50   </Items>
51 </ItemSearchResponse>
```

The Examples above describe a sample communication with a REST request (listing 2.6) to the Amazon Web Service and its response (listing 2.7) to the client. The Request consists only of a HTTP header with the "GET" command. The function parameter (e.g. "Keywords") are behind the question mark. The single function parameter are concatenated with ampersands. The respective function value
(e.g. "Java%20Web%20Services%20in%20a%20Nutshell") is connected with an equals sign to its function parameter. Thus a single function parameter with its value looks like:
"Keywords=Java%20Web%20Services%20in%20a%20Nutshell&".
The response contains:

1. the HTTP header (line 1-8)

2. the response message content (line 11-51). Notice also, that this content is not specified by the REST protocol, its data structure is designed by the developer of the Web Service.

2.5.4 GData

Conceptually, the *GData* (**G**oogle **Data**) [GDa06a] protocol is a mixture of REST and XML-RPC. It is developed by Google to exchange data between the Google Calendar[19] and a client. Method calls use the REST syntax while metadata are sent with an *Atom*[20] (default) or *RSS*[21] feed via HTTP. With the HTTP function GET a new resource can be requested, in case of Google Calendar a feed. The answer will be the requested resource in form of an HTTP response with an Atom feed as payload. To create a new entry in the Google Calendar GData uses the HTTP function POST with the new entry as an Atom feed. The successful action will be acknowledged with an HTTP status-code 201 (created) and an Atom feed containing the inserted data. Updates are made with the HTTP method PUT and the data as Atom feed. The Google answer is the HTTP status-code 200 and the updated feed. To remove an entry, the DELETE function of HTTP is used. Special functions of the Google Calendar like a query can be accessed with the syntax of the REST protocol. Listing 2.8 and listing 2.9 show a sample communication between a client and the Google Calendar service.

As GData is based on Atom it inherits the essential achievement of Atom over pure HTTP: the operation of HTTP operations such as GET or POST not just on the level of entire URLs, but on smaller information units such as entries in a Blog, pictures in a collection, etc. Atom provides consistent means for this finer granular information access and management that are employed in GData as well.

The GData protocol is until today used only by the Google Calendar. But due to the popularity of all Google Web applications and the simplicity of the GData syntax, it has the potential to become a common RPC protocol. It also demonstrates the use of the Atom protocol for information management and how easy it is to extend Atom with additional data and functions.

Listing 2.8: GData Request [GDa06b]

```
GET /jo@gmail.com/private-08ce2fac8efa42f2a0d04eceb7d68cc9/full

TE: deflate,gzip;q=0.3
Connection: TE, close
Host: www.google.com/calendar/feeds/
User-Agent: LWP::Simple/5.803
```

[19] http://www.google.com/calendar
[20] http://www.atompub.org/
[21] http://www.rssboard.org/rss-specification

Listing 2.9: GData Response [GDa06b]

```
TTP/1.1 200 OK
Date: Tue, 20 Jun 2006 13:24:00 GMT
Server: Server
Connection: close
Transfer-Encoding: chunked
Content-Type: text/xml; charset=UTF-8

<feed xmlns='http://www.w3.org/2005/Atom'
 xmlns:gd='http://schemas.google.com/g/2005'>
  <id>http://www.google.com/calendar/feeds/jo@gmail.com/private-magicCookie/full</id>
  <updated>2006-03-29T07:35:59.000Z</updated>
  <title type='text'>Jo March</title>
  <subtitle type='text'>This is my main calendar.</subtitle>
  <link rel='http://schemas.google.com/g/2005#feed' type='application/atom+xml'
    href='http://www.google.com/calendar/feeds/jo@gmail.com/private-magicCookie/full'>
  </link>
  <link rel='self' type='application/atom+xml'
    href='http://www.google.com/calendar/feeds/jo@gmail.com/private-magicCookie/full'>
  </link>
  <author>
    <name>Jo March</name>
    <email>jo@gmail.com</email>
  </author>
  <generator version='1.0' uri='http://www.google.com/calendar/'>CL2</generator>
  <gd:where valueString='California'></gd:where>
  <entry>
    <id>
      http://www.google.com/calendar/feeds/jo@gmail.com/private-magicCookie/full/entryID
    </id>
    <published>2006-03-30T22:00:00.000Z</published>
    <updated>2006-03-28T05:47:31.000Z</updated>
    <category scheme='http://schemas.google.com/g/2005#kind'
      term='http://schemas.google.com/g/2005#event'></category>
    <title type='text'>Lunch with Darcy</title>
    <content type='text'>Lunch to discuss future plans.</content>
    <link rel='alternate' type='text/html'
      href='http://www.google.com/calendar/event?
            eid=aTJxcnNqbW9tcTJnaTE5cnMybmEwaW04bXMgbWFyY2guam9AZ21haWwuY29t'
      title='alternate'>
    </link>
    <link rel='self' type='application/atom+xml'
      href='http://www.google.com/calendar/feeds/jo@gmail.com/
            private-magicCookie/full/entryID'>
    </link>
    <author>
      <name>Jo March</name>
      <email>jo@gmail.com</email>
    </author>
    <gd:transparency value='http://schemas.google.com/g/2005#event.opaque'>
    </gd:transparency>
    <gd:eventStatus value='http://schemas.google.com/g/2005#event.confirmed'>
    </gd:eventStatus>
    <gd:comments>
      <gd:feedLink
        href='http://www.google.com/calendar/feeds/jo@gmail.com/
              private-magicCookie/full/entryID/comments/'>
      </gd:feedLink>
    </gd:comments>
    <gd:when startTime='2006-03-30T22:00:00.000Z' endTime='2006-03-30T23:00:00.000Z'>
    </gd:when>
    <gd:where></gd:where>
  </entry>
</feed>
```

The listings above present a GData request (listing 2.8) and the proper response (listing 2.9). The request is similar to the REST request in listing 2.6. Is consists only of a HTTP header with the “GET” function call. The call consists off the email address (to use the Google Calendar a Google mail account is needed) and the ID of the requested Atom feed (calendar data). The response is a HTTP header with the requested Atom feed as payload. It is analog to listing 2.7, a REST response.

2.5.5 Comparison

A comparison of these different Web Service RPC protocols is rather difficult, because they occupy different niches: XML-RPC is build on simple technologies, REST relies on the use of established techniques, and GData is today only used by Google's Calendar Web Service. SOAP has a wide application area, but becomes very complex.

For simple Web Services XML-RPC or REST can be used, because these two protocols are easy to understand, fast to explore, and supported by various programming languages. But only a few frameworks support REST, so that the programming time of a Web Service may be more expensive. Another advantage of XML-RPC and REST is that these protocols are final versions and will not be changed over the next time. On the one hand, once developed Web Service need not be adopted to future version to be compatible to the new versions. One the other hand, no new, maybe meaningful, extension will be integrated any more.

GData can also be used for simple Web Services, but it is not supported by many programming languages. At the time of writing this thesis, Google provides only libraries[22] for Java[23] and C#[24]. No framework or server libraries are available. But due to is similarity to REST and XML-RPC, frameworks that support these two RPCs may rapidly adapted to GData. However, the underlying Atom protocol is seeing rapid adoption in many languages and gives the added advantage of a consistent approach to fine-granular information management on the Web.

Complex Web Services, so called enterprise Web Services, have high requirements on the RPC protocol. It has to support security and reliability mechanisms, for example. SOAP offers with the WS-* standards these abilities and provides today nearly functionality of CORBA. In spite of the complexity of SOAP, especially when considering all WS-* standards, it is not very complicated to develop SOAP based Web Services, because many frameworks support WS-*. But the design of many SOAP standards is today not complete and maybe changed or adapted to the needs of the companies behind the standards, like Microsoft, IBM, and BEA.

2.6 Conclusion

Is the Web Service concept a new revolutionary technology or just a collective term for well known design pattern. The idea of splitting large programs into small modules is an establish principle of all higher level programming languages and even in the time of assembler programming, procedures are separated from the other program parts for a better re-usability. Even to connect programs over a network with a language independent remote procedure call protocol found already its realization in ISO RPC or CORBA. And CORBA actually provides the definition language IDL to describes the functionality of a remote procedure, very similar to WSDL.

The popularity of Web Services is caused, in my opinion, first by the standardization of these techniques, second by standards that are open enough to be adapted to different situations and concerns. So the developer is not committed to a concrete realization of these concepts. There exists always more solutions to realize the concepts of Web Services: Nearly every programming language can be used for an implementation of a Web Service. With

[22] http://code.google.com/apis/gdata/gdata.zip
[23] http://code.google.com/apis/gdata/client-java.html
[24] http://code.google.com/apis/gdata/client-cs.html

SOAP, XML-RPC, and REST three established messaging protocols are available, but the programmer can also develop is own protocol, like Google does with GData.

The only not very successful technique of the Web Service concept is UDDI. The few public registers provide only little information about the few registered services. Maybe the complex structure of the UDDI system or that every Web Service has to register by hand are responsible for the unpopularity. Today it is often more successful to search a Web Service with a normal Web search engine like Google or Altavista[25].

But many techniques of the Web Service concept are still under development and the weak points may be solved in the coming years.

[25] http://www.altavista.com/

CHAPTER
THREE

WS-Star

3.1 Introduction

The growing collection of the *WS-** standards (spoken: WS-Star), supervised by OASIS[1] and the W3C[2], defines the Web Service Protocol Stack. The goal of this stack is to make SOAP more independent from the transport protocol and more suitable for large, complex business software. It also should give SOAP roughly the same abilities as found in CORBA or RMI [BS04].

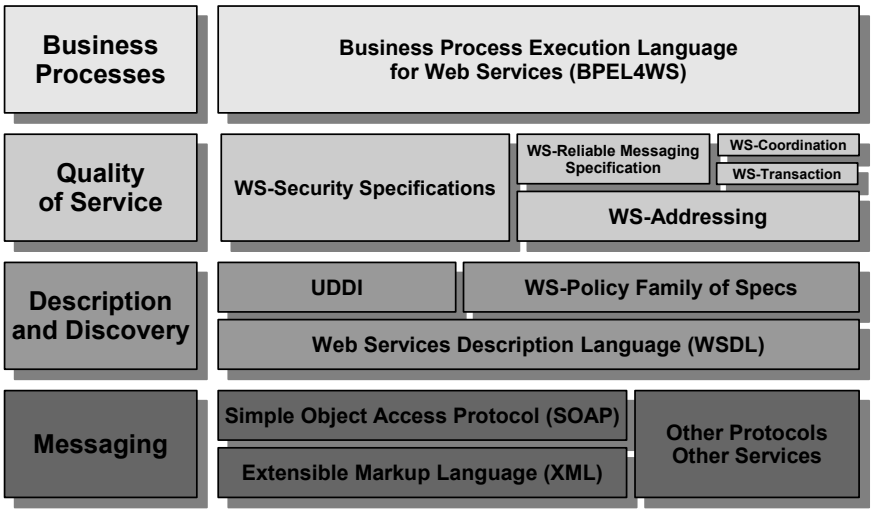

Figure 3.1: The Web Service Protocol Stack [Dod04]

WS-* is designed as a protocol in the fashion of the modular *ISO-OSI* stack. The bottom layer, named messaging layer, provides the basic communication feature for Web Services. The extensible markup language[3] (XML) is the fundament of the stack. It provides the

[1] http://www.oasis-open.org/
[2] http://www.w3.org/
[3] http://www.w3.org/TR/REC-xml/

needed language independency and easy extensibility to the WS-* stack. The SOAP protocol is the other important basis, is extended by many of the WS-* standards.

In the description and discovery layer the following standards are allocated: WSDL describes the functions of a Web Service, "WS-Policy" describes the capabilities and requirements of Web Services, and with UDDI Web Services can be discovered. The standards of the messaging layer and of the description and discovery layer are discussed in chapter 2 with the exception of "WS-Policy".

On top of the description and discovery layer, the quality of service (QoS) layer resides. It includes the "WS-Addressing" standard, which enriches SOAP messages with routing and address information. This standard is very important for the transport independency and is used or extended by many other WS-* standards. The "WS-Security" standard places a security context in the SOAP header, which contains information about the used security techniques. "WS-Coordination" describes basic features to coordinate multiple Web Services and offers basic concepts that are used in the "WS-Transaction" to provide a transaction control for SOAP. Simple transactions can use "WS-AtomicTransaction", while "WS-BusinessActivity" can manage transaction that consist of several atomic transactions. To guarantee that messages reach their destination "WS-ReliableMessaging" places an identifier tag into the SOAP header and dictates strict communication rules.

At the top of the stack the business processes layer is placed. This layer is not covered here as the BPEL4WS framework is beyond the scope of this thesis. Figure 3.1 shows the complete Web Service Protocol Stack.

In the following, the standards forming the WS-* stack are briefly summarized grouped by their position in the WS-* stack.

3.2 Description and Discovery Layer

3.2.1 WS-Policy

WS-Policy [BBC+04] introduces new abilities to WSDL to describe the capabilities, requirements, and general characteristics of Web Services. It is possible to define technical policies like the used transport protocol or policies like QoS (quality of service) characteristics. The WS-Policy framework is targeted only towards a way to *represent* these kinds of information, but does not specify the technical realization.

A policy assertion defines the required behavior of a policy subject and maybe extended by similar alternative policies. A client that wants to use the Web Service *must* satisfy and understand all policy assertion or at least one alternative.

Listing 3.1: WS-Policy example

```
1  <wsp:Policy>
2    <wsp:ExactlyOne>
3      <wsp:All>                    <!-- first assertion -->
4        <wsse:SecurityToken>
5          <wsse:TokenType>wsse:Kerberorv5TGT</wsse:TokenType>
6        </wsse:SecurityToken>
7      </wsp:All>
8      <wsp:All>                    <!-- second assertion -->
9        <wsse:SecurityToken>
10         <wsse:TokenType>wsse:wsse:X509v3</wsse:TokenType>
11       </wsse:SecurityToken>
12     </wsp:All>
13   </wsp:ExactlyOne>
14 </wsp:Policy>
```

Listing 3.1 shows a skeleton of a simple WS-Policy with two policy assertions [BBC+04, page 8]:

1. Line 1 starts the policy in the SOAP header.
2. Line 2-13 represent a collection of two alternative assertions.
3. If the alternative in line 3-7 is selected, the Web Service only accepts Kerberos tokens.
4. If the alternative in line 8-12 is selected, only X509 token are supported.

3.3 Quality of Service Layer

3.3.1 WS-Addressing

SOAP can be used in combination with many different transport protocols like HTTP[4], SMTP[5], or Jabber[6]. But this flexibility of SOAP has one disadvantage: Every protocol has its own addressing mechanism. For example it is possible to insert a SOAPAction URI into the HTTP header. This SOAPAction can be used to route the SOAP message without parsing the XML part of the message. But other protocols do not provide an extensible header like HTTP and restrict the interoperability of SOAP—it is not always possible to transform a SOAP message with a SOAPAction in the HTTP header into a message transported over another transport protocol.

The *WS-Addressing* [CFF+04] framework contributes a universal transport neutral mechanism to address Web Services and messages. It extends WSDL (the service and port elements) and the SOAP header with:

Endpoint References

> An *endpoint* is a referable entity, processor, or resource where Web Service messages can be targeted [CFF+04, page 3].

The new element <Address> contains the URI of the Web Service. The <ReferenceProperties>-tag includes individual information, provided by the issuer, to communicate with the service. Besides tags can be included to declare reference parameters, selected port type, and policy (see section 3.2.1).

Information Header The *Information Header* contains properties to identify and localize the involved endpoints. It is an extension of the optional SOAP header and should not be modified along the message path.

There are two possible message types. The "one way" message type defines no other sequenced interaction after the message is sent. This is the basic message type which is extended by the "request reply" message type. A "request reply" message consists of the initial message (the request) from the endpoint source and the answer message (the reply) from the destination.

The action element specifies the URI and the operation which should be executed on the Web Service. If no action is defined, the default action is:
[target-namespace]/[port-type-name]/[operation-name]{Request, Response}.

The framework provides also a possibility to specify fault messages. These messages are sent to the URI written in the "fault endpoint" tag or otherwise to the URI of the "reply endpoint" tag. A fault message can convey the following elements: The <Code>tag contains the source of the fault message and the sub-code element with the fault category. An example for such a fault category is "Destination Unreachable", which is sent when the destination of the message is unreachable. The description of the fault in natural language is placed into the <Reason>-tag to enable, possibly together with a the short abstract of the fault reason in the detail element, human understanding of the fault reason.

[4] http://www.w3.org/Protocols/
[5] http://tools.ietf.org/html/2821
[6] http://www.jabber.org/

The example 3.2[CFF+04, page 3] shows a SOAP message with the information header. This SOAP message has a unique ID *uuid:6B29FC40-CA47-1067-B31D-00DD010662DA* and an explicit reply address *http://business456.example/client1*. The element <wsa:To> defines the endpoint and <wsa:Action> tag the appended action.

Listing 3.2: WS-Addressing example [CFF+04]

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
            xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <S:Header>
    <wsa:MessageID>
      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://business456.example/client1</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://fabrikam123.example/Purchasing</wsa:To>
    <wsa:Action>http://fabrikam123.example/SubmitPO</wsa:Action>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
```

3.3.2 WS-Security

The *WS-Security* [WS-06] framework extends the SOAP protocol with a *security context*. The aim of this framework is to protect the SOAP message and the Web Service against *unauthorized access*.

The framework does not provide techniques to establish a security context, authentication mechanism, key derivation, exchange or advertisement of security policy, trust establishment or determination, and non-reputation, it only defines new standardized tags for delivering the necessary information about the used security techniques and parameters—it is a *container* for security information.

With ID attributes WS-Security introduces a mechanism to reference the security context. So recipients, that need not understand the whole schema of a SOAP message and that are only interested in the security context, can only examine the SOAP tags with an ID attribute. But some key schemas like XML Signature and XML Encryption do not allow attribute extensibility. Thus WS-Security provides the possibility to use the document-local ID attribute reference mechanism instead of the WS-Security recommended mechanism.

The core of the WS-Security standard is the *security header*, an extension of the optional SOAP header. A single SOAP message can contain one or more security header blocks. With the actor/role attributes the recipient of the Security Header is defined and it is possible to specify for each routing node of the SOAP message individual security information.

Even a wide variety of security tokens can be easily integrated like a <UsernameToken> with the username of the sender. The <BinarySecurityToken> tag contains binary certificates (e.g. X.509[7] certificates or Kerberos[8] tickets) with their encoding. It is also possible to reference security keys elsewhere in the message or even outside.

[7] http://www.ietf.org/rfc/rfc3280.txt
[8] http://www.ietf.org/rfc/rfc4120.txt

To identify the communication partners and to guarantee the fidelity of a message, WS-Security supports the use of signatures to sign or validate a message. Signatures can, furthermore, be confirmed. Secret information can be protected by the encryption of the body or header elements of the SOAP message. The framework provides also a mechanism to determine the freshness of a message, *timestamps*. So it is possible to place the creation time and the expiration time into the SOAP header.

Basic error handling is supported by extending the SOAP error handling. Invalid security tokens, wrong signing, etc. will be reported to the participants. Listing 3.3 shows a SOAP message including a security context with an username token, which is used for identification. Lines 15-33 specify the signature to ensure the integrity of signed elements of the SOAP message and protect them for modification. The used signature is the XML-Signature[9].

Listing 3.3: WS-Security example [WS-06]

```
1  <?xml version="1.0" encoding="utf-8"?>
2    <S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
3                xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
4      <S:Header>
5        <m:path xmlns:m="http://schemas.xmlsoap.org/rp/">
6          <m:action>http://fabrikam123.com/getQuote</m:action>
7          <m:to>http://fabrikam123.com/stocks</m:to>
8          <m:id>uuid:84b9f5d0-33fb-4a81-b02b-5b760641c1d6</m:id>
9        </m:path>
10       <wsse:Security
11         xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
12         <wsse:UsernameToken Id="MyID">
13           <wsse:Username>Zoe</wsse:Username>
14         </wsse:UsernameToken>
15         <ds:Signature>
16           <ds:SignedInfo>
17             <ds:CanonicalizationMethod
18               Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
19             <ds:SignatureMethod
20               Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
21             <ds:Reference URI="#MsgBody">
22               <ds:DigestMethod
23                 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
24               <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
25             </ds:Reference>
26           </ds:SignedInfo>
27           <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
28           <ds:KeyInfo>
29             <wsse:SecurityTokenReference>
30               <wsse:Reference URI="#MyID"/>
31             </wsse:SecurityTokenReference>
32           </ds:KeyInfo>
33         </ds:Signature>
34       </wsse:Security>
35     </S:Header>
36     <S:Body Id="MsgBody">
37       <!-- message content -->
38     </S:Body>
39   </S:Envelope>
```

3.3.3 WS-Coordination

Complex distributed applications consist of mostly more than one Web Service with often different protocols. The *WS-Coordination* [CCF+05b] framework provides protocols to coordinate the activities between different Web Services. An *activity* consists of several Web Service calls which belong to one task which *can* be very complex. It is possible that more Web Services are involved in one activity. An example for an activity could be the booking of a journey. This task can be divided into several Web Service calls like the reservation of a seat in a plane and the reservation of the hotel room. All Web Service operation together yield the activity.

[9] http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/

The WS-Coordination defines a compound Web Service that is partitioned into three component services (see also figure 3.2 [CCF+05b, page 5]):

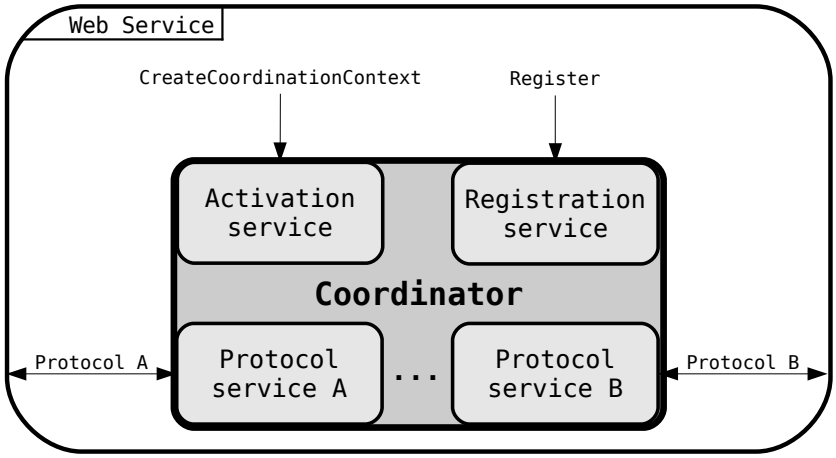

Figure 3.2: WS-Coordination

Activation Service This service creates new activities and the proper *coordination context*. The coordination context is an extension of the SOAP header and provides the information about the access to the registration service, the coordination type (e.g. WS-AtomicTransaction or WS-BusinessActivity), and relevant extensions. Listing 3.4 shows an example Coordination Context.

Listing 3.4: Coordination Context [CCJL04]

```
<wscoor:CoordinationContext>
  <wsu:Expires>2012-04-22T00:00:00.0000000-07:00</wsu:Expires>
  <wsu:Identifier>
    uuid:0f05758b-1f0d-4248-a911-90f7bd18ae52
  </wsu:Identifier>
  <wscoor:CoordinationType>
    http://schemas.xmlsoap.org/ws/2003/09/wsat
  </wscoor:CoordinationType>
  <wscoor:RegistrationService>
    <wsa:Address>
      http://Business456.com/
            MyCoordinationService/RegistrationCoordinator
    </wsa:Address>
    <wsa:ReferenceProperties>
      <mstx:ex
        xmlns:mstx=http://schemas.microsoft.com/wsat/extensibility
        mstx:transactionId="uuid:cfb01dc0-5073-405a"/>
    </wsa:ReferenceProperties>
  </wscoor:RegistrationService>
</wscoor:CoordinationContext>
```

The above example describes a coordination context, which includes the following information:

1. The identifier of the activity in lines 3-5.
2. The type of the coordination, here atomic transaction, in lines 6-8
3. In lines 9-19 the reference to the endpoint of the registration service (http://Business456.com/MyCoordinationService/RegistrationCoordinator).

Registration Service Once the application has received the coordination context, it can register for the activity at the *registration service*. An application can invite other applications to take part in this activity. The invitation contains the coordination context. Now the invited application has two options: It can use its own coordination service to participate in the coordination by registering the new activity. Or it can use the registration service of the host application.

Protocol Service The *protocol service* exchanges the messages between the registered applications. This is very important, because the participate Web Services could use different protocol. So the protocol service has to convert the exchanged protocol for each service. To prevent misunderstandings and errors the WS-Coordination framework supports the following set of faults

- Invalid state: A delivered message had an invalid state.
- Invalid protocol: A delivered message used an invalid protocol.
- Invalid parameters: A delivered message had invalid parameters.
- No activity: Sent by the coordinator, if the participant return no answer message.
- Context refused: Sent by the coordinator, if the coordination context was not accepted.
- Already registered: Sent by the Coordinator, if a participant tries to register again for the same protocol.

The security model of the WS-Coordination framework is based on the WS-Security standard. Only authorized principals can create coordination context or register with an activity/coordination context.

3.3.4 WS-Transaction

WS-AtomicTransaction

The *WS-AtomicTransaction* specification [CCF+05c] defines an extensible framework for the coordination of simple transactions. It extends the WS-Coordination standard with a simple transaction management such as commitment control and a basic rollback function.

The commitment control is based on the "all-or-nothing" principle. This principle means that the coordinator service only commits the transaction if all registered participants successfully acknowledged the transaction, otherwise the transaction will be aborted. For complex transactions the use of the BusinessActivity-Framework is recommended.

AtomicTransaction Protocol The AtomicTransaction-Protocol consists of two parts:

Completion With the completion protocol the application, who wants to initiate a transaction, communicates with the coordinator service. The application signals the coordinator, that a transaction should be committed or rolled back. In the other direction, the coordinator can tell the participant whether a transaction has been committed or aborted.

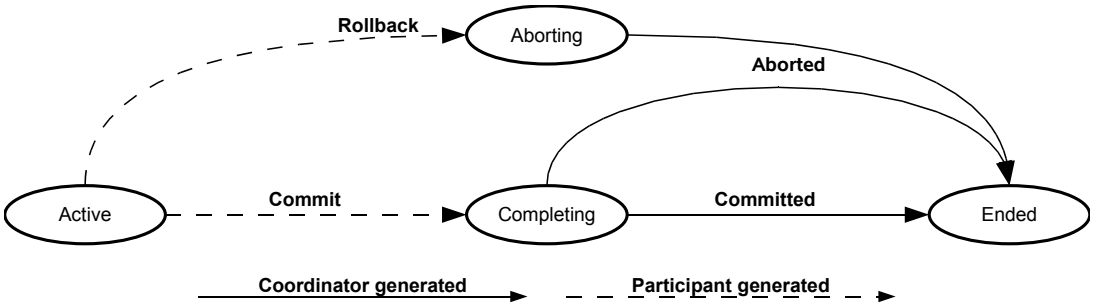

Figure 3.3: Completion Protocol [CCF+05c, page 7]

Two-Phase Commit Protocol (2PC) The Two-Phase Protocol coordinates the multiple participants of an atomic transaction. The two phases of this protocol are:

Volatile Two-Phase Commit Protocol The root coordinator starts the "prepare" phase after receiving a commit notification from the completion protocol. Now all involved participants must response whether their are able to fulfill successfully the transaction with a "prepared" notification, otherwise they send an "aborted" notification and the transaction will be aborted. If all participants are able to fulfill the transaction, the next phase is started.

Durable Two-Phase Commit Protocol After the Volatile phase the root coordinator is preparing the durable phase. Now all participants are ready to perform the transaction. Thus the coordinator has only to send a "commit" signal to initiate the transaction. After the participants have fulfilled the task, they acknowledge the successful transaction to the coordinator and the transaction is concluded.

WS-BusinessActivity

In contrary to the WS-AtomicTransaction standard the *WS-BusinessActivity* [CCF+05a] framework is designed to manage the transactions of a large number of distributed applications with complex structures and relationships. It is an extention of the above described WS-AtomicTransaction protocol.

A business activity consists of a significant number of atomic transactions and may consume many resources over a long time period, even several days. The response time of a single transaction may take considerable time and human interactions may be needed before responding. Due to the complexity and the possibly severe consequences of even a single transaction error, handling of a business activity is very difficult and undoing a single activity is mostly not possible or meaningful. Even an uniform security strategy is difficult to achieve,

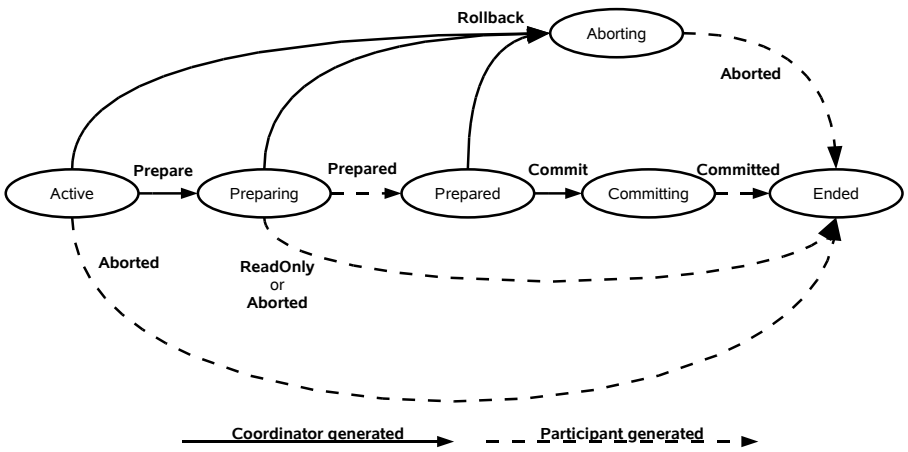

Figure 3.4: Two-Phase Commit Protocol [CCF+05c, page 7]

because it is possible that participants with different security policies are involved in the same activity.

All these points require to enrich the WS-Coordination framework with the following points:

- All state transitions are recorded as reliable, including application state and coordination metadata.
- All notifications are acknowledged in the protocol to ensure a consistent view of state between the coordinator and participant.
- Each notification is defined as an individual message. Transport level request/response retry and time out are not sufficient mechanisms to achieve end-to-end agreement coordination for long running activities.

The BusinessActivity Coordination protocol allows to part business application into business activity scopes, a set of operations on a collection of Web Services with an agreed outcome. So business application can select which child tasks are included or how to handle an exception thrown by a child task. In contrary to the atomic transactions it is also possible that a participant leaves the activity. If a task fails, it can permit its output directly without waiting for solicitation. Thus the coordinator can faster adapt the goals and transactions.

BusinessActivity Protocol

BusinessAgreementWithParticipantCompletion Protocol A Participant registers with the coordinator that manages the protocol. The participant must know when it has completed its work for a business activity and sends a completion notification to the coordinator. If an error occurs, the participant has to send a “fault” message to the coordinator. A participant can always leave this protocol with an exit notification. The coordinator ends the protocol with a “closed” message or with a “compensated” message, if the participant has to roll back the transaction.

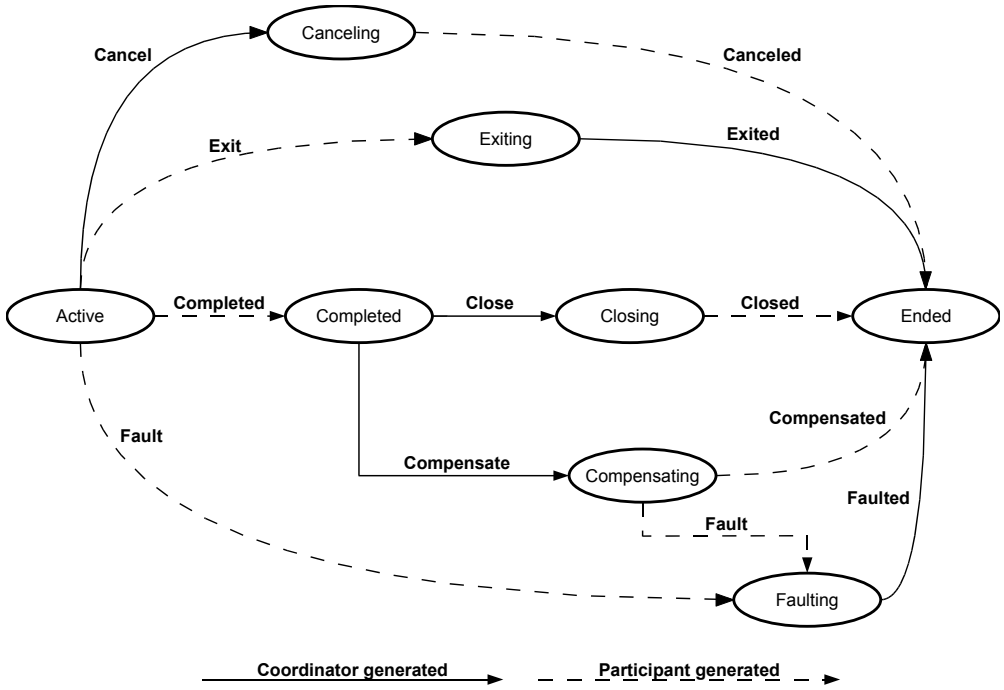

Figure 3.5: BusinessAgreementWithParticipantCompletion Protocol [CCF+05a, page 8]

BusinessAgreementWithCoordinator Protocol The BusinessAgreementWithCoordinator protocol is very simular to the BusinessAgreementWithParticipantCompletion protocol, but here the coordinator notifies the participants, when the task is finished, not the other way round.

3.3.5 WS-ReliableMessaging

For large software products, especially enterprise software, it is often important to guarantee that messages reach their target or, to avoid miss-understandings between the communication partners, are only delivered once. The very popular HTTP protocol does not guarantee this kind of reliability, so it is obvious to extend the SOAP protocol with reliable messaging. The WS-* stack provides two solutions to aim this goal: On the one hand *WS-Reliability* [WS-04], supported by Sun Microsystems[10] and Oracle[11], on the other hand *WS-ReliableMessaging* [BBC+05], proposed by IBM[12], Microsoft[13], and BEA Systems[14]. These two standards are not compatible to each other, but provide similar features. In this thesis only the WS-ReliableMessaging standard is analyzed, because the scope of this thesis is the evalutaion of Web Service concepts and not the concret realizations. So it would be redundant to describe the same concepts twice. The WS-ReliableMessaging framework allows a reliable communication in the presence of software component, system, or network failures. It is

[10] http://www.sun.com/
[11] http://www.oracle.com/
[12] http://www.ibm.com/
[13] http://www.microsoft.com/
[14] http://www.bea.com/

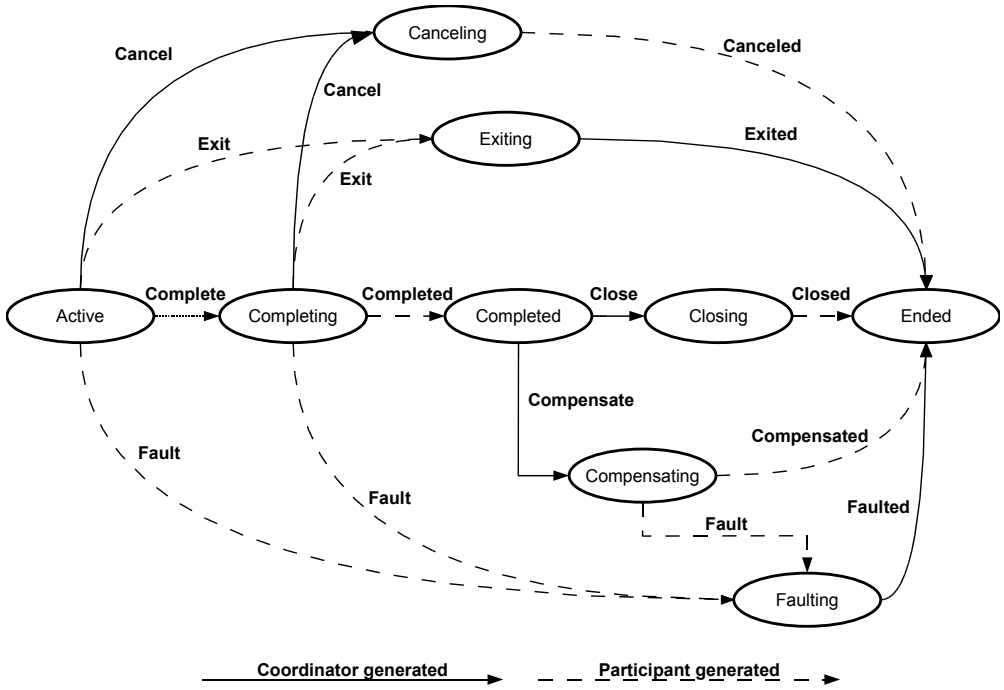

Figure 3.6: BusinessAgreementWithCoordinator Protocol [CCF+05a, page 9]

modular and extents the WS-Security, WS-Policy, and other WS-* standards. The basic assurances a Web Service endpoint can provide are:

- *AtMostOnce*: Messages will be delivered at most *once* (no duplication) or an error will be raised.
- *AlLeastOnce*: *Every* sent message will be delivered. It is possible that same messages will be delivered several times.
- *ExactlyOnce*: *Every* sent messages will be delivered *and* the messages are delivered at most *once* (no duplication).
- *InOrder*: Messages will be delivered in the *order* that they were sent. The messages *may be* omitted or can be duplicated.

The following preconditions have to be fulfilled:

- The reliable message source must have a URI.
- The reliable message source must have knowledge of destination's policies (see section 3.2.1).
- If security is required both, the reliable message source and the reliable message destination, must have a security context (see Section 3.3.2).

The Protocol WS-ReliableMessaging is implemented based on a sliding window sequence numbers algorithm similar to the one used in TCP: The core element of the reliable message header, an extension of the SOAP header, is the <Sequence> block. It must contain a unique identifier for each message and a message number, which increments by one from the initial value of one. Listing 3.5 represents a SOAP message with a sequence block like described above:

1. Lines 3-7 contain the squence block of a SOAP header.
2. Line 4 describes the unique identifier of the SOAP message.
3. Line 5 the message number ("8").
4. Line 6 indicates that this is the last message of the sequence and that the receiver has to send an acknowledge message as next.

Listing 3.5: Reliable Message with Sequence Header

```
1   <SOAP:envelope>
2     <SOAP:header>
3       <sequence>
4         <Identifier>http://www.example.com/ws</Identifier>
5         <MessageNumber>8</MessageNumber>
6         <LastMessage></LastMessage>
7       </sequence>
8     </SOAP:header>
9     <SOAP:body>
10      ...
11    </SOAP:body>
12  </SOAP:envelope>
```

At the beginning of a reliable communication the sender has to contact the receiver with a special message whose sequence block contains the <CreateSequence> element. If the receiver is ready to accept the following sequence of messages, it has to return a message with a <CreateSequenceReponse> element in the sequence block. The last element of a sequence, which excludes the <LastMessage> element, has to be acknowledged by the receiver by sending a message with a <SequenceAcknowledgement> header block. This <SequenceAcknowledgement> header block contains the sequence numbers of all *received* messages. So the sender can control that the communication partner has *received* all parts of a sequence and, if required, send an element of the sequence a second time. To force an acknowledgment, the sender can simply include an <AckRequested> header into a message. Finally, if all messages are successfully sent and received, a <SequenceTerminationt> header in a message finished the sequence.

Figure 3.7 shows a sample reliable conversation between two endpoints. Endpoint A wants to send a sequence consisting of three messages to endpoint B (CreateSequence), which B acknowledge with a "CreateSequenceResponse". Then endpoint A sends the sequence, but the second message is lost. B acknowledges only message one and three, so that A has to send message two again. After endpoint B has acknowledged, that all messages have successfully been received, A terminates the conversation.

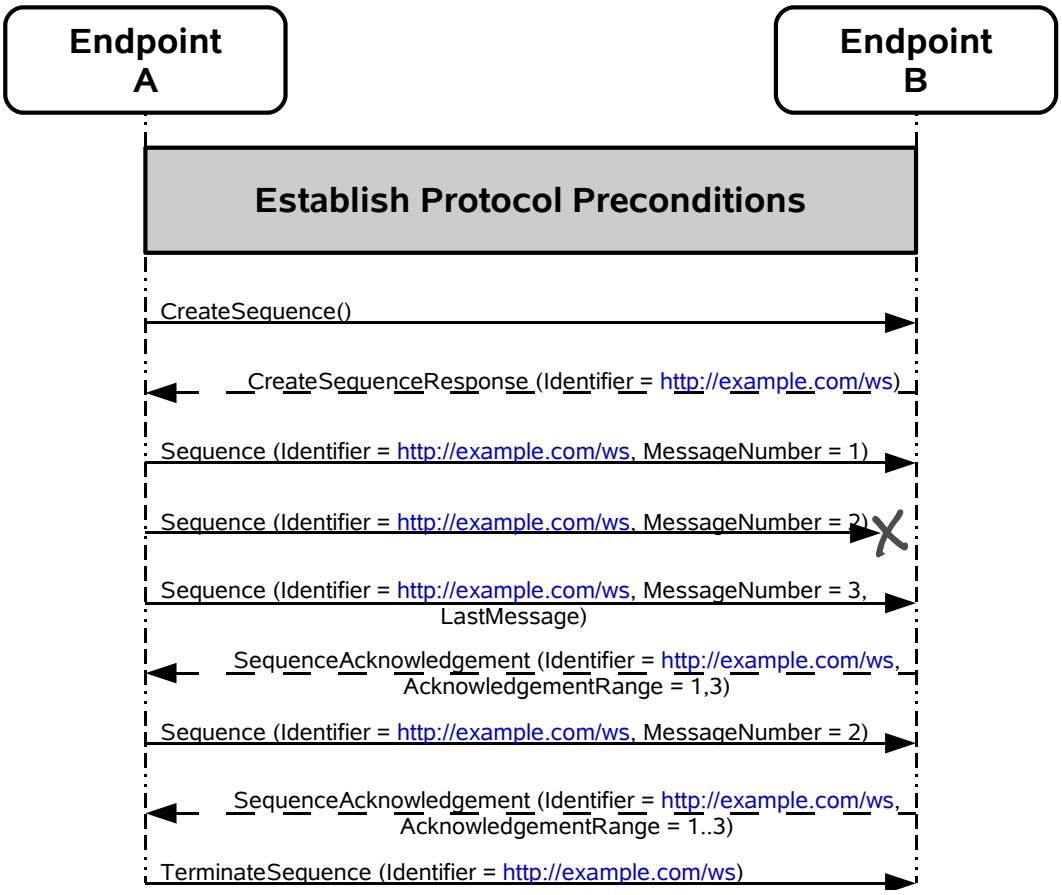

Figure 3.7: Reliable communication between two endpoints [BBC+05, page 8]

CHAPTER

FOUR

Implementation

4.1 Framework Comparison

4.1.1 Preliminary Note

The following comparison of the different Java SOAP stacks focuses on the one hand on the list of features (see Section 4.1.12) and on the other hand on the performance. The performance of an implementation relying on the SOAP stack mostly depends on the speed of the XML parser and serializer. To parse or generate a SOAP message three approaches are very common [GBSK03]:

1. The *DOM*[1] (**D**ocument **O**bject **M**odel) based approach parses the whole XML document, here the SOAP message, and builds a complete tree representation in memory. Thus random access to the document is possible as well as navigation through the tree and easy updates. But parsing the complete document, especially large documents, costs significant memory and hinders performance if only parts of the documents are needed. The W3C DOM particularly suffers from rather space-inefficient Java implementations. Alternative document object models such as XOM[2] provide better performance characteristics, but are not widely supported.

2. In contrast *SAX*[3] (**S**imple **A**PI for **X**ML) is an event-driven *push model* [MB] for parsing XML documents. It fires (or "pushes") events while it reads the XML document sequentially. Event handlers catch the pushed events and enable access to the content. Thus applications have to control which parts of the document to process immediately, which to store in memory. If only a prefix of elements of the documents are need, not the whole XML document has to be parsed, which causes a performance benefit. But applications with a SAX parser need a complex state model, because there is no tree materialisation for easy navigation and orientation available, while walking through the document. Finally, it is not possible to jump back or directly to a single XML element, the document has to be walked sequentially.

[1] http://www.w3.org/DOM/
[2] http://www.xom.nu/
[3] http://www.saxproject.org/

3. *StAX* (**St**reaming **A**PI for **X**ML) [StA] is very simular to SAX: It is an event-driven *pull model*, where the program has to request (or to "pull") the next parsing events, if it is needed. StAX has two models:

   The *cursor model* returns events like SAX, while the *iterator model* returns the events as objects, which costs slightly more performance and causes an obvious memory overhead, but is often more comfortable. The performance is comparable to SAX, but the state model is integrate into the parser and not, like in the case SAX, into the application.

A SOAP message has to be parsed only once. It is possible that the messages are very large, there is no limit to the length. SAX and StAX fulfilled exactly these requirements, and provide a good performance, while the advantages of DOM are not that relevant for SOAP message parsing.

4.1.2 Java WSDP

JWSDP[4]—**J**ava **W**eb **S**ervices **D**eveloper **P**ack—is not a real Web Service framework, it is a collection of java libraries and tools that are necessary to develop and deploy Web Services. There are libraries for core XML processing, XML binding, and Web Services. The next paragraphs describe the libraries important for SOAP parsing:

SAAJ

The *SAAJ*[5] (**S**OAP with **A**ttachments **A**PI for **J**ava) library provides a simple way to build, read, and modify SOAP messages, even with attachments[MH04]. SAAJ can be used in combination with JAX-WS or stand-alone with its own integrated messaging functions. Clients can connect directly with a SOAPConnected object to the service and need no servlet-/J2EE-container for point-to-point connection [SUN], but it provides only synchronous, point-to-point messaging and has no WSDL support.

To parse or generate the SOAP message SAAJ uses DOM level 2, more precisely JDOM[6] (**J**ava **DOM**).

JAXM

The **J**ava **A**PI for **X**ML **M**essaging[7] (*JAXM*) allows synchronous request/response communication as well as asynchronous one-way communication [CJ02]. The messaging mechanism of JAXM is a bit different to the concept of SAAJ: The messages between client and server can be managed with a messaging provider. This messaging provider allows asynchronous request/response communication and can feature additional services like reliable delivery service, routing, or WS-* standards, but it needs a servlet-/J2EE-container. Without a message provider only synchronous exchanges as with SAAJ is possible. Like SAAJ JAXM is JDOM-based.

[4] http://java.sun.com/webservices/jwsdp/
[5] http://java.sun.com/webservices/saaj/
[6] http://www.jdom.org/
[7] http://java.sun.com/webservices/jaxm/

JAX-WS

JAX-WS[8], old name JAX-RPC, provides a more practical oriented way to deploy Web Services. It comes with tools that support the developer by automatically generating client/server code from a WSDL file. At the cost of a some predefined choices, the developer avoids repetitive coding. JAX-WS comes with a StAX based parser, developed in the Java community process, group JSR (**J**ava **S**pecification **R**equests) 173[9].

4.1.3 WSIF

The Web Service framework from the Apache Software Foundation[10] is called *WSIF*[11] which stands for **W**eb **S**ervices **I**nvocation **F**ramework. The core of the WSIF is the Web Service description, the WSDL file. Every with WSDL describable Web Service can be invoked with WSIF. The used SOAP stack is called AXIS:

AXIS

The **A**pache e**X**tensible **I**nteraction **S**ystem, short called *AXIS*[12], is a re-implementation off the Apache SOAP project[13] and has its origin in the IBM SOAP stack SOAP4J[14]. It can handle synchronous request/response as well as one-way messaging. From a WSDL file it is possible to generate a Web Service client or server code or the other way round.

A benefit of AXIS is its modular architecture [Bay03]: Its structure can be divided into several modules for transport, administration, etc. Each of these modules consists of a chain of handlers that are responsible for a single task like logging for example. All these modules and handlers are registered in the AXIS XML configuration file. By adding new modules or handlers into this file the AXIS engine can be extended by additional features. If a message is received, it runs through the chain—if a message is send it run through the chain in the other direction.

SOAP4J and Apache SOAP are DOM based SOAP engines, but AXIS changed to the SAX based approach and thus reached better performance with the Xerces[15] SAX parser in comparison to the both other engines.

AXIS2

AXIS2[16] is a complete new implementation of the AXIS SOAP stack. It provides simular features like AXIS, but should have a higher performance. It uses a modular architecture with modules and handler chains as well, but has a new object model with name AXIOM[17] (**AX**is **O**bject **M**odel). This is a StAX based XML Infoset model, which causes the expectation of better performance in comparison to AXIS. Today AXIS2 is not integrated into the WSIF framework.

[8] https://jax-ws.dev.java.net/
[9] http://www.jcp.org/en/jsr/detail?id=173
[10] http://www.apache.org/
[11] http://ws.apache.org/wsif/
[12] http://ws.apache.org/axis/
[13] http://ws.apache.org/soap/
[14] http://www.alphaworks.ibm.com/tech/soap4j/
[15] http://xerces.apache.org/
[16] http://ws.apache.org/axis2/
[17] http://ws.apache.org/commons/axiom/

4.1.4 ETTK for Web Services

The IBM *ETTK* – **E**merging **T**echnologies **T**ool**k**it – *for Web Services*[18] is the successor of the WSTK[19] (**W**eb **S**ervice **T**ool **K**it). It comes with the IBM Java SDK[20], and its own application server (WebSphere Application Server[21]), some WS-* support, and UDDI4J[22], an open-source Java implementation of UDDI [Tid01]. IBM's WebSphere SOAP engine is closed-source, thus little details about the used parser are available.

4.1.5 XFire

The open-source Web Service Framework *XFire*[23] is developed by the Codehaus project[24]. XFire supports many different transport protocols—HTTP, JMS, XMPP, In-JVM, etc. It places emphasis on the easy usability and the good performance[25] of the framework.

There are two ways to generate a client and server stub for a Web Service. Either it is possible to write the java code using the API or, without java knowledge, by writing XML files.

The good performance may have its origin in the use of a StAX[26] processor in the SOAP engine.

4.1.6 XINS

XINS[27] is a specification-oriented framework. This means that at the beginning of the implementation of a Web Service, a functional specification has to written in a XML file. From this XML file XINS generates a HTML or OpenDocument-format documentation and client as well as server code. Finally, it generates an empty-shell Web application in the form of a WAR file. With an additional tool the generation of the WSDL file is possible too. XINS supports only HTTP with REST, POX-RPC, XML-RPC, and SOAP as Web Service RPC. It has better than average Log4J-based[28] logging support. XINS implements a SAX approach to parse a SOAP message.

4.1.7 JibxSoap

JibxSoap[29] is a SOAP engine that is based on JiBX[30]. This framework binds XML data to Java objects. The strength of JibxSoap is that it supports document/literal Web Services rather than RPC/encoded document style. It should have a good performance and flexibility. But today it has only reached alpha stadium with same limitation.

[18] http://www.alphaworks.ibm.com/tech/ettkws
[19] http://www.alphaworks.ibm.com/tech/webservicestoolkit
[20] http://www-128.ibm.com/developerworks/java/jdk/
[21] http://www-306.ibm.com/software/webservers/appserv/was/
[22] http://www-106.ibm.com/developerworks/webservices/library/ws-uddi4j.html
[23] http://xfire.codehaus.org/
[24] http://www.codehaus.org/
[25] http://xfire.codehaus.org/Performance
[26] http://stax.codehaus.org/
[27] http://xins.sourceforge.net/
[28] http://logging.apache.org/log4j/
[29] http://jibx.sourceforge.net/jibxsoap/
[30] http://jibx.sourceforge.net/

4.1.8 JSoap

JSOAP[31] is a small and fast SOAP engine, which supports only SOAP 1.1, but with attachments. It can handle WSDL and is not bound to a specific transport protocol. It provides the possibility to write a client without being confronted with the SOAP details (called high-level-client) or by building the SOAP message with all element manual (low-level-client) [MÖ3]. JSOAP supports all JAXP-parser—DOM as well as SAX based.

4.1.9 Celtix

The open-source Enterprise Service Bus *Celtix*[32] is a Co-production of IONA Technologies[33] and ObjectWeb[34]. It provides a complete JAX-WS 2.0 implementation, supports same WS-* standards and script languages like Java-Script. Celtix has its own native SOAP engine with SOAP 1.1 support [Tec06]. Only HTTP or JMS are supported as transport protocol, but the framework can be extended with custom protocols.

4.1.10 Glue

The *Glue*[35] framework is provided from webMethods. It comes with an integrated Web server, a servlet engine, and a JSP engine. Thus, it is possible to use it standalone without a servlet-/J2EE-container. It automatically maps Java to WSDL. The entire Glue framework is not an open-source project, but webMethods provides a free, though slightly restricted "standard" version without support on their Web site[35].

4.1.11 JBossWS

The new version of the *JBossWS*[36] framework has its own SOAP 1.2 engine, no longer AXIS. It supports lots of WS-* standard and RPC/literal, Document/literal, and Message document style for web services. The new JBoss XML parser is based on SAX.

4.1.12 Feature List

The following lists compare features of the most common SOAP stacks. These comparisons are inspired from similar ones published on the Web side http://xfire.codehaus.org/Stack+Comparison by the Codehaus project, the vendor of the XFire SOAP stack.

General Features [Cod06]

This table summarizes the general features of the Java SOAP stacks. These features are very important for the choice of a Java SOAP stack to implement Web Service calls in Xcerpt. An open source framework provides independency from proprietary approaches, and gives the chance to adopt the framework to the ones needs. RPC-encoding is an encoding style for

[31] http://soap.fmui.de/
[32] http://celtix.objectweb.org/
[33] http://www.iona.com/
[34] http://www.objectweb.org/
[35] http://www.webmethods.com/
[36] http://www.jboss.org/wiki/Wiki.jsp?page=JBossWS

SOAP messages. This style is not supported by the WS-I[37], an open industry organization that promotes the interoperability of Web Services, but is still in use. For example, the Google Web Service uses the RPC-encoding style. The next points on the list contain the supported Web Service RPC protocols. REST support would be preferable in Xcerpt, because many Web Services provide only a REST interface and should not be ignored. SOAP 1.2 is the latest version of the SOAP protocol, discussed in section 2.5.1. The table shows additionally two performance and development centric measures: which implementation uses StAX XML parsers, which provide the ability to generate code from an WSDL file.

Feature	AXIS 1.x	AXIS2	Celtix	Glue	JBossWS	XFire (1.1)
Open Source	X	X	X	-	X	X
RPC-Encoding	X	-	-	X	X	-
REST Support	-	X	-	-	-	-
SOAP 1.1	X	X	X	X	X	X
SOAP 1.2	X	X	-	X	X	X
SOAP with Attachments	X	X	-	X	X	-
Streaming XML (StAX based)	-	X	-	-	-	X
WSDL->Code (Client)	X	X	X	X	X	X
WSDL->Code (Server)	X	X	X	X	X	X

[37] http://www.ws-i.org/

JSR standards [Cod06]

JSR[38] standards enable a good interoperability between the different frameworks. The JSR standards are requirements on the Java platform. New specification to Java are often developed in a JSR group:

- The JSR 181[39] group specifies an annotated format to enable definition of Java Web Services in a J2EE container.
- The JSR 208 group[40] (JBI) supports the SOA integration into the Java platform.
- The JSR 224 group[41] (JAX-WS) supports the integration of XML based RPC protocols into the Java platform. It is the successor of JAX-RPC.

SAAJ provides a standard way to send SOAP with attachments over the Internet from the Java platform.

Feature	AXIS 1.x	AXIS2	Celtix	Glue	JBossWS	XFire (1.1)
JAX-RPC	X	-	-	X?	X	-
JAX-WS	-	-	X	-	-	-
JBI	?	-	X	-	-	X
JSR 181	X	-	X	-	X	X
JSR 181 on Java 1.4	-	-	-	-	-	X
SAAJ	X	X	X	X	?	-

WS-* [Cod06]

This table lists the supported WS-* standards of the Java SOAP stacks. Not all of these standards are introduced in chapter 3. The WS-Eventing[42] standard defines basic "event notification massage" abilities, similar to the competing WS-Notification[43] standard.

Feature	AXIS 1.x	AXIS2	Celtix	Glue	JBossWS	XFire (1.1)
WS-Addressing	X	X	X	X	X	X
WS-Eventing	-	-	-	-	-	X
WS-Notification	X	-	?	-	-	X
WS-ReliableMessaging	X	X	X	-	-	-
WS-Policy	-	X	-	-	-	-
WS-Security	X	X	-	X	X	X
WSDL 1.1 Support	X	X	X	X	X	X
WSDL 2.0 Support	-	-	-	-	-	-

[38] http://www.jcp.org/
[39] http://www.jcp.org/en/jsr/detail?id=181
[40] http://www.jcp.org/en/jsr/detail?id=208
[41] http://jcp.org/en/jsr/detail?id=224
[42] http://www.w3.org/Submission/WS-Eventing/
[43] http://www-128.ibm.com/developerworks/library/specification/ws-notification/

Transports [Cod06]

Here the protocols are listed that can be used as transport protocol for the messaging protocols of Web Services.

- HTTP[44] is the common Internet protocol. It is used to exchange data between Web servers and Web Browsers.
- JMS[45], the Java API for exchanging messages.
- Jabber[46] is a protocol for instant messaging.
- SMTP[47] is a common email transport protocol[48].
- TCP[49] is the basic, reliable Internet transport protocol.

Feature	AXIS 1.x	AXIS2	Celtix	Glue	JBossWS	XFire (1.1)
HTTP	X	X	X	X	X	X
JMS	X	X	X	X	X	X
Jabber	X	-	-	-	-	X
SMTP	X	X	-	-	-	-
TCP	X	X	-	-	-	-

4.1.13 Decision

To implement Web Service calls in Xcerpt, one of the explored Java SOAP stacks has to be chosen. To decide which one fulfills the requirements, the diagram 4.1 was designed. The red bars symbolizes filters labeled with requirements that have to be fulfilled by the frameworks. If a framework does not fulfill a requirement, it is dropped by the respective filter. Thus the list of the frameworks shrinks and at the bottom of the diagram those framework should be left, that fulfill all requested requirements.

The first filter should eliminate all frameworks, that did not reach a stable development phase at the time of writing. Unstable SOAP stacks are not fully developed, provide mostly not the full functionality of the final version, and deep changes into the design cannot be excluded.

In the next filter all frameworks that are closed source and not freely available are dropped. The open source attribute is very important, because this kind of software uses mostly open standards and not proprietary approaches. So the users does not depend on the solutions of one company, but can change maybe to another framework. Furthermore, the licensing of Xcerpt is open-source as well. Finally, even if the SOAP stack would not fit exactly to the requirements of Xcerpt, it would be possible to adapt the chosen solution to our needs without involvement of the stack vendor.

[44] http://www.ietf.org/rfc/rfc2616.txt
[45] http://java.sun.com/products/jms/
[46] http://www.jabber.org/
[47] http://www.ietf.org/rfc/rfc821.txt
[48] http://www.ietf.org/rfc/rfc793.txt
[49] http://www.ietf.org/rfc/rfc793.txt

The support of WS-* is needed for further projects. With these standards the whole range of the SOAP technology is available and can integrated in Xcerpt. So more kinds of Web Services can queried and new areas of application may open up to Xcerpt.

The SOAP stack should also provides a good performance. Above, the most common approaches of XML parsers are discussed with the conclusion that SAX and StAX based SOAP stacks provide the best performance. XFire and AXIS2 are the only remaining framework that use a parser which implementation is StAX based.

At last, many Web Services RPC protocols should be supported by the framework. The REST protocol is very desirable, because many Web Services nowadays provides only a REST interface. AXIS2 is the only framework that supports the REST approach.

4.1.14 Experiences with AXIS2

In appendix A the Java code of a test implementation of a Web Service call is listed. This class should deliver insight to the abilities and weak points of the AXIS2 framework.

The AXIS2 project advertises on their Web page with good documentations and with the ease-of-use of the framework. In our experience, this claim, however, is not quite reality at the time of writing. The provided documentation was very simple and not many examples were available. The structure of the Java API was not described and even the Java-doc document was only fragmentary. The names of the Java classes were not very intuitive and sometimes even seem illogical.

AXIS2 was just released at the time of writing this thesis and so no third party documentation was available. The implementation of the test class needed considerable amount of time and proved a rather tedious task, because with the lack off good documentation many trials were necessary. Probably not the best solution was found.

Very recently IBM published a comprehensive, an at first glance very useful documentation[50] about the AXIS2 framework, but sadly to late, that this work could profit from it.

[50] http://www-128.ibm.com/developerworks/webservices/library/ws-webaxis1/

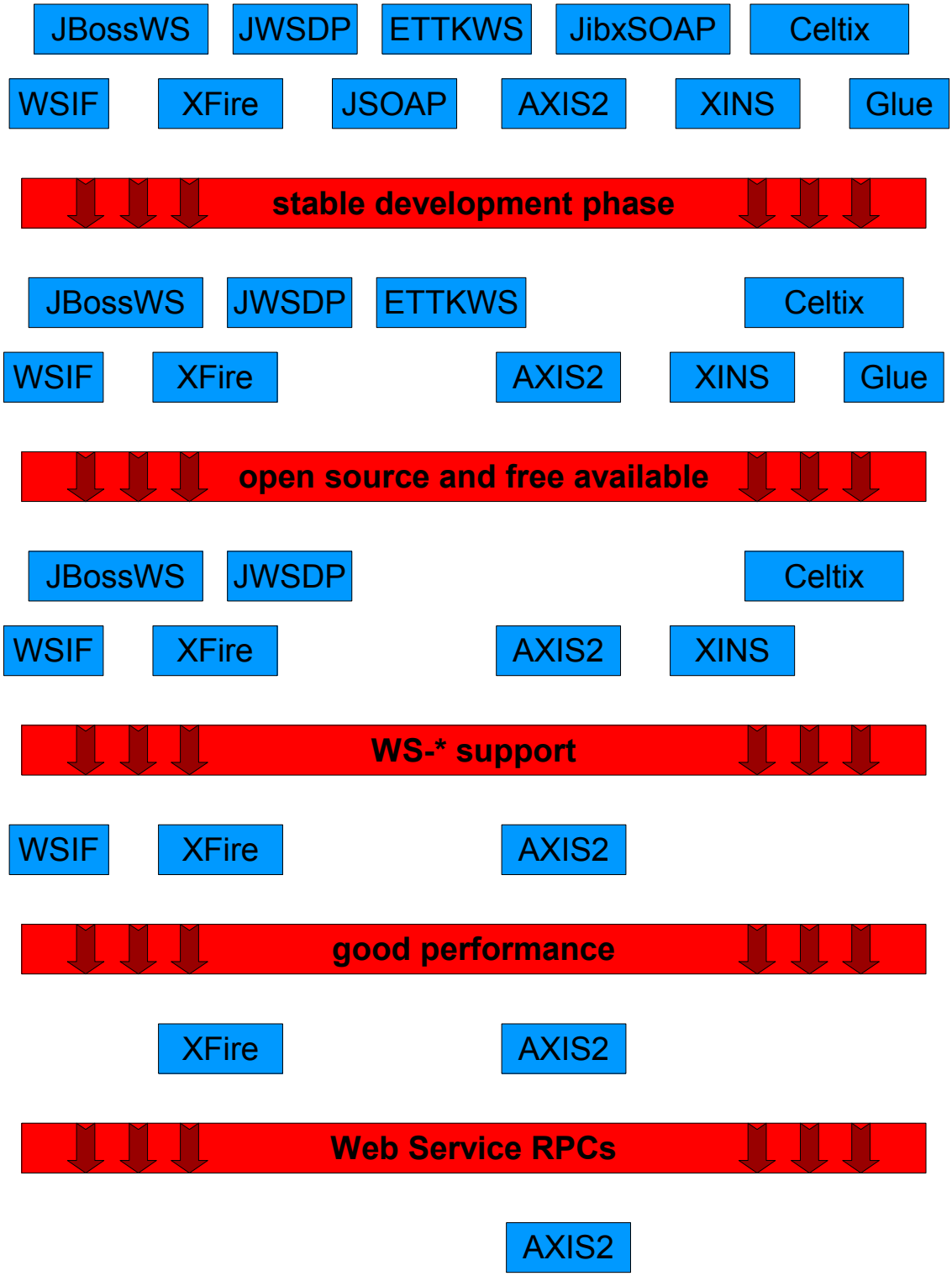

Figure 4.1: Decision Filter Diagram

CHAPTER
FIVE

Xcerpt and Web Services

5.1 Short Introduction to Xcerpt

Xcerpt [Sch04] is a query and transformation language for the markup language XML. Xcerpt has its roots in logic programming and is a representative of the declarative programming paradigm. Thus Xcerpt is *referential transparent*, so that every occurrence of an expression in the same context has the same meaning. Thanks to its *answer closedness*, one can use any Xcerpt answer as an Xcerpt query. Another benefit of Xcerpt is that it separates query and construct terms, which makes the programs better to read and to understand, as well as offering certain advantages for program analysis and execution. In contrast to other XML transformation and query languages like XQuery or XPath, which are both path-based, Xcerpt uses a pattern- and rule-based approach. In path-based languages the query term has to specify a navigation through the document-tree, while in pattern-based languages the query term presents a form or example for the selected data. Xcerpt uses the asymmetric simulation unification to evaluate queries [BS02b].

Terms are the basic parts of an Xcerpt program. There are three kinds of terms:
A **data term** is an abstract representation of semi-structured data, e.g. an XML document, in Xcerpt. The XML structured data in listing 5.1 which represent two books can be represented in Xcerpt like shown in listing 5.2. Square brackets [ ] indicate an ordered element, while curly braces { } represent an unordered element.

Listing 5.1: XML bib [BS02a]

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <authors>
      <author><last>Stevens</last><fist>W.</first></author>
    </authors>
    <publisher>Addison-Wesley</publisher>
    <prise>65.95</prise>
  </book>
  <book>
    <title>Advanced Programming in the Unix environment</title>
    <authors>
      <author><last>Stevens</last><fist>W.</first></author>
    </authors>
    <publisher>Addison-Wesley</publisher>
    <prise>65.95</prise>
  </book>
</bib>
```

Listing 5.2: Xcerpt data term [BS02a]

```
bib {
  book {
    title { "TCP/IP Illustrated" }
    authors [
      author { last { "Stevens"  }, first { "W."  } }
            ],
    publisher { "Addison-Wesley" },
    prise { "65.95"  }
  }
   book {
    title { "Advanced Programming in the Unix environment" },
    authors [
      author { last { "Stevens"  }, first { "W."  } }
            ],
    publisher { "Addison-Wesley" },
    prise { "65.95"  }
  }
}
```

Query terms look like incomplete data terms with variables as place holders. They are patterns to be matched with the queried data term. The variables hold the matched content of the data term. Data that matches with "*Exact sub-term patterns*", represented by double braces ([[ ]] and {{ }}), must not contain more sub-terms than the query term, while "*partial sub-term patterns*", represented by single braces, may have more sub-terms than the query term. Listing 5.3 shows a query term that matches to a knot "title" with the content "TCP/IP Illustrated" which has to be the child of a "book" and a "bib" knot.

Listing 5.3: Xcerpt query term

```
bib {{
  book {{
    title { "TCP/IP Illustrated" }
  }}
}}
```

Construct terms reorganize the data of the query result, bound to in the variables, and form a new data term. Ordered and unordered elements are represented like in data terms with square and curly braces.

Construct-query rules are the building blocks of an Xcerpt program. These rules consist of a construct term, called "*head*", and a non-empty query term, called "*body*". It can be seen as an *if... then* rule—*if* the body succeeded *then* the result is constructed like defined in the head. Xcerpt allows to use the head of a term in the body of another term, so-called rule chaining. This enables the possibility of recursive programs. Thus complex queries can be parted into a set of rules, which are easier to understand and to read. Construct-query rules, which specify an output source like a file are called *goals*.

Listing 5.4: Xcerpt construct term [Sch04]

```
in {
  resource [ "http://www.xcerpt.org/bib.xml", "xml" ],
  bib {{
    book {{
      var Title -> title {{ }}
    }}
  }}
}
```

The above example (listing 5.4) describes a construct term which selects the titles of the books from the file "bib.xml" and assigns the results to the variable "title".

Listing 5.5: Xcerpt goal term [Sch04]

```
GOAL
  out {
     resource [ "file:prices.html", "html" ],
     html [
       head [ title [ "Price Comparison" ] ],
       body [ var Content ]
     ]
  }
FROM
  var Content -> table {{ }}
END
```

The listing 5.5 above shows a goal that writes its results, structured in a table, into a HTML file with name "prices.html".

5.2 Web Services united with Xcerpt

5.2.1 Syntax

To access an external XML source [Sch04], like a local file or a remote web site, Xcerpt provides the following syntax:

```
in(location=<uri>)
  <query>
```

Now the external source can be queried and the result assigned to a variable, for example. The syntax of a Web Service call is guided by the syntax used to access an external resource, but has to take into consideration that Web Service calls may include named parameters.

```
webservice( (soap|rest)=<uri>,
            (<parameter-name>::<parameter-type>=<parameter-value>)* )
  <query>
```

The keyword "*webservice*" introduces a Web Service call followed by attributes of that Web Service call defining the parameters to be used. There are two predefined attributes "*soap*" and "*rest*" that are mutually exclusive. They indicate which protocol and (in their value) which Web Service URI to use. The remaining parameters are defined by the Web Service and may additionally be given a type using double colons to indicate the parameter type. The *parameter type* is, e.g., a type according to XML-Schema[1], *parameter name* is the name of the parameter defined for example in the WSDL file, and *parameter value* the value that will be submitted to the Web Service server.

The following example is a Web Service call of the Google search service, where the query part selects all found URIs. The parameter list is abbreviated. The messages, that are exchanged between Xcerpt and Google are displayed in Listing 5.6 and in Listing 5.7:

```
webservice(soap="http://api.google.com/search/beta2",
           q::string="Xcerpt", start::int="0", ...)
  return {
    resultElements {
      item { var Url -> url {{ }} }
    }
  }
}
```

[1] http://www.w3.org/XML/Schema

Listing 5.6: Xcerpt request

```
<ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <key xsi:type="xsd:string">12f3456a7d89</key>
  <q xsi:type="xsd:string">Xcerpt</q>
  <start xsi:type="xsd:int">0</start>
  <maxResults xsi:type="xsd:int">2</maxResults>
  <filter xsi:type="xsd:boolean">true</filter>
  <restrict xsi:type="xsd:string"></restrict>
  <safeSearch xsi:type="xsd:boolean">false</safeSearch>
  <lr xsi:type="xsd:string"></lr>
  <ie xsi:type="xsd:string"></ie>
  <latin1 xsi:type="xsd:string">latin1</latin1>
</ns1:doGoogleSearch>
```

Listing 5.7: Google response

```
<return xsi:type="ns1:GoogleSearchResult">
  <directoryCategories xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
                       xsi:type="ns2:Array"
                       ns2:arrayType="ns1:DirectoryCategory[0]">
  </directoryCategories>
  <documentFiltering xsi:type="xsd:boolean">true</documentFiltering>
  <endIndex xsi:type="xsd:int">2</endIndex>
  <estimateIsExact xsi:type="xsd:boolean">false</estimateIsExact>
  <estimatedTotalResultsCount xsi:type="xsd:int">28900</estimatedTotalResultsCount>
  <resultElements xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
                  xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[2]">
    <item xsi:type="ns1:ResultElement">
      <URL xsi:type="xsd:string">http://www.xcerpt.org/</URL>
      <cachedSize xsi:type="xsd:string">27k</cachedSize>
      <directoryCategory xsi:type="ns1:DirectoryCategory">
        <fullViewableName xsi:type="xsd:string" />
        <specialEncoding xsi:type="xsd:string" />
      </directoryCategory>
      <directoryTitle xsi:type="xsd:string" />
      <hostName xsi:type="xsd:string" />
      <relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
      <snippet xsi:type="xsd:string">
        A logic-based query and transformation language for XML and semistructured data.
      </snippet>
      <summary xsi:type="xsd:string" />
      <title xsi:type="xsd:string">&lt;b>Xcerpt&lt;/b></title>
    </item>
    <item xsi:type="ns1:ResultElement">
      <URL xsi:type="xsd:string">http://sourceforge.net/projects/xcerpt/</URL>
      <cachedSize xsi:type="xsd:string">30k</cachedSize>
      <directoryCategory xsi:type="ns1:DirectoryCategory">
        <fullViewableName xsi:type="xsd:string" />
        <specialEncoding xsi:type="xsd:string" />
      </directoryCategory>
      <directoryTitle xsi:type="xsd:string" />
      <hostName xsi:type="xsd:string" />
      <relatedInformationPresent xsi:type="xsd:boolean">true</relatedInformationPresent>
      <snippet xsi:type="xsd:string">
        The world&amp;#39;s largest development and download repository of Open Source code.
      </snippet>
      <summary xsi:type="xsd:string" />
      <title xsi:type="xsd:string">SourceForge.net: &lt;b>Xcerpt&lt;/b></title>
    </item>
  </resultElements>
  <searchComments xsi:type="xsd:string" />
  <searchQuery xsi:type="xsd:string">Xcerpt</searchQuery>
  <searchTime xsi:type="xsd:double">0.28077</searchTime>
  <searchTips xsi:type="xsd:string" />
  <startIndex xsi:type="xsd:int">1</startIndex>
</return>
```

Syntactically Web Service parameters are rather close to Xcerpt *construct* terms. But, for this work, we limit the allowed constructs in Web Service parameters rather strictly: only ground terms and variables are allowed in Web Service parameters. Section 5.2.3 discusses why optional variables are problematic. Constructs such as *all* or *some* as used in normal

construct terms are currently also prohibited, but can most likely be allowed without much further adaption.

At last, the current version of the Web Service call implementation provides not the possibility to declare namespaces for the parameter types. The namespaces are hard coded, but in further versions the option to specify own namespaces will be integrated.

5.2.2 Referential Transparency and Web Services

Referential transparency is one important design principle of Xcerpt and of all declarative programming languages. It says that every occurrence of an expression in the same scope must have the same meaning. But a Web Service call cannot fulfill this principle. If you request a query, for example to the Google Web Service, results can differ from one call to the other, due to changes to the Google web site index or dynamic ranking adaption. Another example would be Web Services that are connected with a database like travel reservation services. If you request such a service twice, it is very probable that the results are different, because in the meantime someone could book a journey and affect the content of the database. Summed up: The time of a Web Service call can affect its result, and an implementation of Web Service calls in Xcerpt would violate the principle of referential transparency.

If an implementation would execute every Web Service call, the programmer has to be aware of the fact that a call of the same Web Service *can* lead to different results and the programmer *cannot* count on referential transparency. Furthermore a Web Service call costs a lot of time, caused by the round-trip delay time, the time a message needs from the sender to the receiver and vice versa, and the responding time of the Web Service, especially of large business services. This means a considerable performance deficit for Xcerpt and long waiting time for the user.

An implementation of Web Service calls in Xcerpt without violating the referential transparency principle leads to the following assumption: A call of a Web Service has to be *idempotent*. Idempotency means, that every execution of a function, here the Web Service, results in the *same* response. A benefit of this assumption is, that it is now possible to fetch the result of a Web Service call only once and store the received result. This stored result could yield the data for all further occurrences of the same Web Service call in the same scope. On the one hand this approach would increase the performance by saving the waiting time for every response. On the other hand not the current data of a Web Services are use for computation, and may adulterate the result of the Xcerpt program. Even a caching strategy is needed to manage the stored results. To hold the result too long into memory would waste memory space. But to keep them too short would cause a new request to Web Services and make the solution obsolete.

An alternative to this assumption would be to let the programmer decide whether he wants the "fetch once" mechanism like discussed above or the execution of every Web Service call. A special keyword at the Web Service call syntax could signalize that the call should initiate a new request to the Web Service, which maybe violates the referential transparency principle. A Web Service call without this keyword could use "fetch once" mechanism and save the referential transparency. This solution would leave all doors open and provide a powerful solution to Xcerpt. But the danger is that programs with lots of Web Service calls become very in-transparent for humans, because every Web Service call has to be inspected precisely to recognize the chosen call solution. Even the programmer has to work very accurately, because the different syntax of Web Service calls provides a error source and may not be very

obvious for every Xcerpt user.

At last, and likely the easiest solution would be to simply ignore this problem, aware of the fact that referential transparency is violated by some Web Service calls. In aspect to the strength of Xcerpt this solution is very practicable. Xcerpt's strength is the querying of Web Services, that deliver XML data for transforming. These Web Services have mostly no side effects on the requested data and thus modify the data not very often. The chance to get the same response several times is very high and a violation of the referential transparency principle is low.

For an implementation a good performance and an easier syntax are decisive. The idempotency and the ignorance approach above provide a good performance to Xcerpt, but determine Xcerpt to one approach. The solution with two different syntaxes for Web Service calls is the most mighty one and Xcerpt is undetermined to one approach. If a good distinguishable syntax for the different approaches, "fetch once" and execute ever Web Service call, is chosen, the weakest point could be ruled out.

5.2.3 Sequences of Web Service Calls

Web Services calls are a little bit different in comparison to the inclusion of an external file. A Web Service often not only has an output, it has an input, the function parameter, too. These input parameters can be assigned to a variable. This variable can contain only a simple string value, that is already known, or the output of a previous Web Services call, which will be known until execution time. In an imperative, flow controlled, programming language this would not be a problem, but in declarative language it is. In Xcerpt, it is not possible to specify the exact evaluation order[2]. So it might happen that, in a sequence of Web Service calls, a variable "v" is used as container for the output of Web Service "A" and as an input parameter for a Web Service "B". In a declarative language, it is not possible to anticipate the evaluation order of the two Web Services calls, maybe "A" will be called first and than "B" or "B" first and than "A". If "A" is called first, everything would be fine, because if "B" is executed the variable "v" contains the output of "A". But if Web Service "B" is executed first, the input parameter variable "v" has no meaningful value and the Web Service call "B" would yield an unintentional result.

The example below illustrates this scenario: The first Web Service call searches the author of the book "Java Web Services in a Nutshell" and binds the result to the variable "Author". Next the Google search Web Service uses the variable "Author" to find all Web pages with the name of the author of the book "Java Web Services in a Nutshell". If Google Web Service is called before the Amazon Web Service the variable "Author" does not contain the author name and the Xcerpt program does not lead to the expected result.

[2]Though there has been some discussions on supporting explicit sequential forms of conjunction and disjunction in Xcerpt.

```
in {
  webservice(soap="http://webservices.amazon.com/...",
                  keywords::string="Java Web Services in a Nutshell",...)
  ItemSearchResponse {
    var Author -> Author {{ }}
  }
  webservice(soap="http://api.google.com/search/beta2", q::string=Author, ...)
  return {
    resultElements {
      item { var Url -> url {{ }} }
    }
  }
}
```

Range Restriction

The range restriction for Xcerpt programs is introduced in [Sch04, Chapter six] and intuitively means that all variables of a rule must occur at least once outside of a negation in the body of the rule. As in conventional logic programming, range restriction makes the formulation of the formal semantics of a program easier and helps to prevent programming mistakes.

To decide whether a rule is range-restricted the variables are marked with polarities: Positive polarity is marked with a plus "+" and symbolizes a consuming occurrence of a variable, which will be assign to a value at execution time. Negative polarity is marked with a minus "-" and means a defining occurrence of a variable, that is already assign to a value at execution time. Furthermore, variables can be marked with an question mark "?" which means that the variable is optional and can occur as well in defining as in consuming context.

Notations

- A program P consists of a set of rules (or goals) R, here written $P = \{R_1, \ldots, R_n\}$ ($n \in \mathbb{N}$).
- A rule consists of a construct term t^c and a set of queries, here written $R = t^c \leftarrow\ Q_1 \vee \ldots \vee\ Q_n (n \in \mathbb{N})$.
- A query Q_n consists of a set of query terms t^q with $Q_n = t^q_{n1} \wedge \ldots \wedge t^q_{nm}$ ($n, m \in \mathbb{N}$).
- A query term t^q with a *Web Service call* is denoted as $t^{WS} = (p_1, \ldots, p_r; t^{q'})$ ($n \in \mathbb{N}$) where $p_1, \ldots, p_r$ are Xcerpt *parameter construct terms* specifying the parameters of the Web service call and $t^{q'}$ is a query term to be evaluated against the result of the Web Service call. Notice, that conventional Xcerpt resource specifications are special cases of Web Service calls, where $r = 1$ and p_1 is the URI of the resource.

The following definition of range restriction originates from [Sch04]:

Definition: Range Restriction after [Sch04] Let R be a rule or goal and let $R' = t^c \leftarrow\ Q_1 \vee \ldots \vee Q_n$ ($n \geq 0$) be the disjunctive rule normal form (see [Sch04, Definiton 6.3]) of R. R is said to be *range restricted*, iff

1. for each disjunct Q_i $(1 \leq i \leq n)$ holds that each variable occurring with positive polarity in either t^c or Q_i also occurs at least once with negative polarity in Q_i.

2. each variable attributed as *optional* and with *negative polarity* in at least one of the Q_i $(1 \leq i \leq n)$, and without another non-optional, negative occurrence in Q_i, is also attributed as optional in all positive occurrences in Q_i and t^c.

A program P is called *range restricted*, if all rules $R \in P$ are *range restricted*.

In the context of Web service access, however, a weakness of this definition becomes obvious: If variables can occur in resource access statements such as resources URIs or Web Service parameters, these variable occurrences are of positive polarity. However, it does not suffice to simply demand that for each such variable also an occurrence with negative polarity exists as the following example shows: Given a rule $R : t_c \leftarrow t_1^{WS} \wedge t_2^{WS}$ with $t_1^{WS} = (\texttt{var}\, X; \texttt{desc var}\, Y)$ and $t_2^{WS} = (\texttt{var}\, Y; \texttt{desc var}\, X)$. For this rule, it is not possible to establish bindings from the occurrences of X and Y with negative polarity as the Web Service calls are (cyclic) dependent on each other.

To achieve a sequential evaluation of Web Service calls and thus a safe range restriction for Xcerpt programs with Web Service calls, the definition of range restriction above has to be extended:

Definition: Range Restriction for Xcerpt Programs with Web Service calls Let R be a rule or goal and let $R' = t^c \leftarrow\ Q_1 \vee \ldots \vee Q_n$ $(n \geq 0)$ be the disjunctive rule normal form (see [Sch04, Definiton 6.3]) of R. R is said to be *range restricted*, iff

1. for each disjunct $Q_i = t_{i1}^q \wedge \ldots \wedge t_{im}^q$ a strict (partial) order[3] $<$ on $\{t_{i1}^q, \ldots, t_{im}^q\}$ $(i, j, m \in \mathbb{N})$ exists such that for each variable x occurring non-optional in the Web Service parameters of t_{ij}^q there exists a $t_{ik}^q < t_{ij}^q$ in which x occurs with negative polarity. Intuitively, $<$ establishes a sequence in which the Web Service calls can be evaluated without violating range restriction.

2. for all variable occurrences outside of Web Service parameters, the above formulation of range restriction applies.

Notice, that this formulation of range restriction does not impose any restrictions on optional variables in Web Service parameters. This is due to the syntactic restriction discussed in Section 5.2.1 that currently forbids the use of optional in Web Service parameters.

The motivation for this prohibition lies in examples such as the rule $R : t_c \leftarrow t_1^{WS} \wedge t_2^{WS}$ with $t_1^{WS} = (u_1, \texttt{optional var}\, X; \texttt{desc var} Y)$ and $t_2^{WS} = (u_1, \texttt{optional var}\, Y; \texttt{desc var}\, X)$. The design goal for optional in Xcerpt is that it (1) does not affect query matching and (2) produces bindings wherever possible. These two design goals, however, conflict in this case: Neither of the possible semantics matches nicely with these design goals:

1. *Treat optional like non-optional:* In this case the above rule would be rejected as not range restricted merely due to the addition of an optional query term (violation of design goal (1)).

[3]Recall, that a strict partial order is an irreflexive, transitive, antisymmetric binary relation.

2. *Establish an order, possibly ignoring one or more optionals:* In this case we could choose either order between t_1^{WS} and t_2^{WS}. But we have no way to know which order will produce more bindings without evaluating both orders (violation of design goal (2)).

3. *Evaluate all orders that are range restricted if ignoring optionals:* This comes closest to the usual maximization semantics for optional, but has the distinct disadvantage that the user can no longer imagine the Web Service calls to be executed in any particular order. Furthermore, it clearly increases the evaluation costs. Notice, that this semantics can be achieved by rewriting the rule using only disjunction. This seems to be the semantics that is most desirable, if optional in Web Service parameters is to be allowed.

Monads

Another approach to allow sequential evaluation in declarative programming languages is to use *monads* like the programming language Haskell. A monad is a triple (M, unit, $\star$). M is the type constructor from a normal type to a monad. Unit represents the identity function, while the infix function $\star$ executes a function on a monad. With monads it is possible to evaluate sequence in written order. If the approach should be implemented in Xcerpt a syntactical construct is needed to mark a sequence block. Now the programmer has to bring the Web Service calls in the right order to avoid unbounded variables as function parameter like in an imperative programming languages like C or Java.

Deadlocks

The original formulation of range restriction could lead to deadlocks where one Web Service call depends on results of another that depends on results of the former.

The revised formulation of range restriction introduced above solves this problem by introducing a (partial) order on Web Service calls thus preventing the *cyclic wait* deadlock condition.

Procedural Abstraction over Web Services

Procedural abstraction is a very common technique to structure a program. A program can be split into logical parts, which makes it easier to maintain and more readable. The major advantage of such a structuring is that single program parts can be reused and need not be written again.

The sequentialization of Web Service calls is very similar to procedural abstraction. A sequence of Web Services calls can also be seen as a logical part of an Xcerpt rule. If these parts would be marked with named tags, it would be possible to reuse these sequences in other rules only be using the tag name. This procedural abstraction over sequences would simplify the programming effort.

The problem of procedural abstraction over Web Service calls is that the rule may now also have "parameters" that need to be bound before the rule can be executed. In other words, it looses its relational character and becomes more like a function.

An extension of Xcerpt rules separating "in" and "out" blocks of a function header is under consideration, but out of the scope of this work.

5.2.4 Error Handling

An error can be seen as an un-normal, unmeant, and sometimes unexpected behavior of a program at runtime. A typical example is the division by zero.

Exception An *exception* is a special data type for handling errors. If an error occurs, an exception is raised (or thrown), what means, that the normal program flow is interrupted and a special handler treat the exception. Some programming languages allow the programmer to influence the reaction on an exception. The creation of own exception types is mostly possible too, and gives a good solution to adapt the error handling to one's requirements.

In Java [LYK+00] for example, program blocks, that could throw an exception, can be surrounded with a *try-catch-clause*. If an exception is thrown, the program code in the catch block will be executed and the exception maybe handled without terminating the program. An optional *finally* block can be used to execute code, if an error happens in the try block or not. If an exception is not handled with a try-catch-clause, the program block will be interrupted and the exception will be hand through the program until an adaptive exception handler is found. If no adequate handler is available, the program will be terminated and an error message will be shown.

An advantage of this kind of error prevention is, that exceptions are *type safe*. This means that the programmer is forced to handle any error that may occur at runtime. Exception allow also a better diagnosis of errors, because the exception type can be significant.

A disadvantage of this approach is that exceptions are not *side-effect* free and can therefore be hard to use in declarative programming languages, particularly in logic-based programming languages. One approach [kS90] is to transform the logic program by eliminating the exceptions with *negation as failure*.

Error Parameter Another approach to handle errors in programs is to use *error parameter* [LS98] to specify the occurred error. If an error occurs, the program terminates and returns a significant value to signalize the unexpected program flow. The programmer has to explicitly handle this return value with normal program constructs, like an if clause, to react adequate. This kind of error treatment is used, e.g., in C. Here often a program block return "-1" to signalize an unexpected termination, mostly an error. This approach is *side-effect free* and could be good integrated into a logic based programming language. But if the programmer ignores the error parameter, the errors will not be handled and the program can react unpredictable or crash. The error analysis is often rather involved and error prone, due to the lack of nuances of the error value.

Xcerpt: Web Services and Errors At a Web Service call a wide variety of errors are possible: The server can be unavailable, the answer take too long time, the network be interrupted, or the function parameter are incorrect. Xcerpt does not provide an error handling mechanism like most logic based programming languages. An implementation of an *exception* handling mechanism would be difficult and cause deep change in Xcerpt.

So the more practical approach of *error parameter* is chosen in this work to handle errors at Web Service calls. Every error will be delivered through as an XML message to the programmer. The programmer has to write extra rules for error handling, otherwise the Xcerpt program will not return the expected results.

This approach is for Web Services very practicable, due to the most errors are already available as XML documents. For example, SOAP provides fault messages to inform the communication partners about errors and even HTTP error codes are only HTML pages, which are XML documents too.

5.2.5 Iterative Web Services

With the caching of Web Service results Web Service calls can be enhanced with an iterative approach. It is now possible to cut the results in bits and pieces, and request only parts of the result on demand. Combined with a clever caching strategy it could be possible to reduce the waiting time between requests of the same Web Service. Xcerpt could, while allocating the cached result parts to the program, already request the next parts of the results from the Web Service server. The waiting time between the requests would so not visible for the user.

An example is again the Google search. Google sort the results of a query to relevance. The web page represent these results of the query, which can be millions of relevant pages, by default as a list of ten hits. The pages with the highest relevancy are at the top, the lowest at the bottom. With the next button on the bottom of the web page the next ten result can be requested.

The Web Service of Google provides the same feature with a function parameter "start", which contains the result page as value, and "maxResults", which defines the maximal results per request. The Xcerpt caching strategy has only to increase the "start" parameter to get the next results, while the program uses the cached results. The only difficulty is to catch the right moment to request the next results: Too early would unnecessarily increase the network traffic, because the program maybe need no more results. If the next results are to late requested the program has to wait for the data and the performance benefit is lost.

5.3 Conclusion

To integrate Web Services into Xcerpt is a little bit like integrating procedural concepts into a logic based programming language, because a Web Service call can be seen as a procedure call. A single procedure call is not problematic in Xcerpt, but if more Web Services calls are executed, both concepts are very difficult to unite. The problem of sequential Web Service calls is described above and is a good example for this kind of problem.

Another disadvantage is the absence of important procedural concepts in Xcerpt. For example to handle an exception that occurs during a Web Service call would lead to very complex changes at the language design. Furthermore, little research has been done on exception handling in logic based languages, so that practical solutions are not available.

On the other hand the integration of the Web Service technology into an XML transformation and query language is a logical step, because the core technology of Web Services is based on XML and could offer Xcerpt a lot of new application areas. Interesting are mainly Web Services that provide XML data like the Google search service. Web Services that provide special services like the picture administration service Flickr[4] are not so interesting, because Xcerpt main focus is on querying and transforming XML data and not the management of communication.

[4]http://www.flickr.com/

CHAPTER
SIX

Related Work

Related Work

XQuery at Your Web Service [OS04] The goal of this research paper is to obtain direct access to the XML data of a SOAP message. This is useful, because the SOAP message has not to be converted in, e.g., object-oriented classes, which provides mostly no natural representation of a message. Furthermore if the structure of a SOAP message is changed, their representation classes have to be adopted too. With a binding between WSDL and XQuery[1] the authors developed commando-line tools to generate automatic importable XQuery module for direct Web Service access. The other direction is also possible: Get a Web Service from a XQuery module. An advantage of the direct access of SOAP messages is that errors are detected at compile time, because of the static type system of XQuery.

The aim of the paper is very similar to the goal of this thesis: Both want to integrate Web Service calls into an XML query and transformation language. But the differences are that XQuery is a functional programming language, while Xcerpt is logic based. In XQuery sequences of Web Service call are not a problem, because XQuery provides the possibility to evaluate sequences in a given order. Even the referential transparency problem is not relevant in XQuery, as the formal semantics assumes a single evaluation of each expression and this is reflected in most implementations.

Yoo-Hoo! Building a Presence Service with XQuery and WSDL [FOS04] This is a demonstration of the XButler project. XButler project provides the development of fast and more reliable Web Services by integrating XQuery into the infrastructure of Web Services. It uses the ideas of [OS04].

Web Service Composition Using a Deductive XML Rule Language [BAV05] This paper describes a system, called SWIM, to compose different Web Services with the use of a mediator service. SWIM, a knowledge based system, provides a deductive language X-DEVICE that makes it possible to delegate a single request to different Web Services and combining the answer to a single response.

This idea may serve as inspiration for further works, that could realize an similar approach with Xcerpt. But for this thesis the composition of Web Service is beyond the scope.

[1] http://www.w3.org/TR/xquery/

XL: An XML Programming Language for Web Service Specification and Composition [FGK03] Web Services and XML are very closely coupled. A programming language for Web Services should therefore support XML as best as possible. The authors of this paper present a programming language design, which is based on XML. It is compliant to the W3C standards XQuery, XML Protocol, and XML Schema. The advantages of such a language is that there is no mixture of different programming paradigms, no lossy transformation from, e.g., XML to Java-objects, and no different type systems. XL provides high-level programming with declarative constructs (exception handling, logging, data lineage, time-triggered actions).

A Programming Language for Web Service Development [CDR05] The author of this article proposes a programming language for Web Services with focus on WSDL support which is based on XML. It integrates XQuery for expression handling and imperative language constructs, like loops and sequence. The author of this paper means also that it is very usefull, because other languages have no concurrency control or message correlation. The problem of integrating imperative constructs to support sequences of Web Service calls in Xcerpt is a topic of this thesis, cf. Chapter 5.

CHAPTER
SEVEN

Future Work

Future Work

This thesis aims at preparing the way for the integration of Web Service technology into Xcerpt. The next paragraphs should give some ideas on how to extend Xcerpt with additional techniques supporting Web Services:

Integration of REST, XML-RPC, and GData This thesis only implemented a Web Service call with the SOAP protocol in Xcerpt. But many Web Services use other protocols like REST, XML-RPC, or GData. So it would be useful to integrate the other protocols in Xcerpt too. The AXIS2 framework provides the support for REST, but the other protocols have to use other libraries.

Web Service call from a WSDL file Many Web Services provide a WSDL file with the information about the Web Service functions. It would be a useful feature to use this file to facilitate Web Service calls in Xcerpt. It should be possible to generate automatically a function call. The programmer would only have to specify the location of the WSDL file and the used function parameters.

A Web Service with Xcerpt Not only Web Services can be invoked from Xcerpt. The query and transformation language is qualified for providing programs as Web Service *server* itself. For this implementation the AXIS2 SOAP stack can be used as well. Only an application server is needed to manage the communication between the participants. AXIS2 provides also the possibility to generate automatically a `war` Web archive deployment file, which contains already all necessary information for a Java application server such as Tomcat[1]. So an Xcerpt Web Service could be comfortably created.

Build an Xcerpt Web Service from a WSDL file The creation of a Web Service client from a WSDL file was discussed above, so it is not very difficult to generate the skeleton of a Web Service server from a WSDL file. It is often useful to develop first the WSDL file as the interface and functional specification of the Web Service. A program could read this file and

[1] http://tomcat.apache.org/

generate automatically a skeleton with the needed rules that is used as basis for the actual implementation of the Web Service.

Distributed Web Services with Xcerpt Different Web Services can be easily combined to a new service, which has to be managed by a mediator service. Such a mediator service has to split the incoming calls and send them to each of the combined Web Services. The responses of each service must be merged to a single response. An Xcerpt program could be used as mediator service, because transforming XML messages is one strength of Xcerpt.

UDDI exploration with Xcerpt An Xcerpt program could use the UDDI register service data to find an adequate Web Service or Web Services for a predefined task. With the WSDL files available in the register it could generate a mediator service like described above.

Unfortunately UDDI provides only little information about the functional details of a Web Service, but maybe in further version it should be extended to search engines for Web Services.

APPENDIX

A

Code

```
/*
 *
 * WebServiceSOAP.java
 *
 */

package com.amachos.webservices;

import com.amachos.datamodel.cursor.*;
import java.io.InputStream;
import java.io.StringReader;
import java.util.Iterator;
import javax.xml.stream.XMLEventReader;
import javax.xml.stream.XMLInputFactory;
import javax.xml.stream.XMLStreamException;
import javax.xml.stream.XMLStreamReader;
import org.apache.axiom.om.OMAbstractFactory;
import org.apache.axiom.om.OMElement;
import org.apache.axiom.om.OMNamespace;
import org.apache.axiom.om.impl.builder.StAXOMBuilder;
import org.apache.axiom.soap.SOAPFactory;
import org.apache.axis2.AxisFault;
import org.apache.axis2.Constants;
import org.apache.axis2.addressing.EndpointReference;
import org.apache.axis2.client.Options;
import org.apache.axis2.client.ServiceClient;
import org.apache.axis2.context.MessageContext;
import org.apache.axis2.context.MessageContextConstants;


/**
 * @author Clemens Schefels
 * @version 0.2
 * realizes a Web Service call with SOAP from Xcerpt
 */
public class WebServiceSOAP implements WebServiceEngine {
```

```
    OMElement operation, omelement;
    MessageContext msgContext;
    SOAPFactory omFactory;
    OMNamespace defNs;
    ServiceClient serviceClient;


    /**
     * Constructor of WebServiceSOAP
     */
    public WebServiceSOAP() {
        omFactory = OMAbstractFactory.getSOAP11Factory();
        defNs = OMAbstractFactory.getSOAP11Factory().createOMNamespace("
            , "");
    }

    /**
     * calls the Web Service
     * @param uri - address of the endpoint (Web Service = ws)
     * @param parameters - an array the consists of: (Action, port name,
         ws name, namespace of ws, function_call). The function_call looks
          like xsd_type:function_name=function_parameter ("
         example:boolean:safeSearch=false")
     * @return returns the result of the SOAP-call, the SOAP-body.
     */
    public Iterator call(String uri, String... parameters) {
        operation = omFactory.createOMElement(parameters[1].toString(),
            parameters[2].toString(), parameters[3].toString());
//      parameter[1] := port name, parameter[2] := Web Services name,
   parameter[3] := Web Service namespace

//      i = 4, because parameters[x], x>= contains the information about
   the function-calls
        for(int i = 4; parameters.length > i; i++) {
            extractParameter(parameters[i]);
        }

        Options options = new Options();
        EndpointReference endpoint = new EndpointReference(uri);
        options.setTo(endpoint);
        options.setAction(parameters[0].toString()); //parameter[0] :=
            action of the Web Service
        options.setTransportInProtocol(Constants.TRANSPORT_HTTP);
        options.setProperty(MessageContextConstants.CHUNKED, Constants.
            VALUE_FALSE);

        operation.declareNamespace("http://www.w3.org/2001/XMLSchema-
            instance","xsi");
        operation.declareNamespace("http://www.w3.org/2001/XMLSchema","
            xsd");
        OMElement result = null;
        try {
```

```
        result = sendAndreceive(options, endpoint);
    } catch (AxisFault ex) {
        ex.printStackTrace();
    }

    return result.getChildren();

}

public Iterator<Node[]> call(String uri, Node... parameters) {
    return null;
}

public Iterator<Attribute[]> call(String uri, Attribute... attributes
   ) {
    return null;
}

/**
 * adds the functions_call of a port to an omelement (The SOAP
    message)
 * @param fname is the function-name
 * @param xsd_type is the type of the function-parameter (string, int
    , boolean)
 * @param param contains the function-parameter
 */
private void makeParameter(String fname, String xsd_type, String
   param) {
    omelement = omFactory.createOMElement(fname, defNs);
    omelement.addAttribute("xsi:type", "xsd:" + xsd_type, null);
    omelement.addChild(omFactory.createOMText(param));
    operation.addChild(omelement);
}

/**
 * get the part of the parameter arry that contains the function-name
    , the type of the function-parameter, and the function-parameter,
 * part it into three independent Strings and calls @link(
    makeParameter)
 * @param para contains the function-name, the type of the function-
    parameter, and the function-parameter ("
    example:boolean:safeSearch=false")
 */
private void extractParameter(String par) {
    String para = par; //par looks like: xsd_type:function_name=
       function_parameter ("example:boolean:safeSearch=false")
    if(para.contains(":") & para.contains("=")) {
        String type = para.split(":")[0];
        String fname = (String) para.subSequence((para.indexOf(":"))
           +1, para.indexOf("="));
        String param = (String) para.subSequence((para.indexOf("="))
           +1, para.length());
        makeParameter(fname, type, param);
```

```
        }
    }

    /**
     * makes the SOAP call and returns the result
     * @param options the SOAP message
     * @param endpoint the address of the Web Service
     * @return returns the result of the SOAP-call (the SOAP-body)
     */
    private OMElement sendAndreceive(Options options, EndpointReference
       endpoint) throws AxisFault {
        serviceClient = new ServiceClient();
        serviceClient.setOptions(options);
        serviceClient.setTargetEPR(endpoint);
        System.out.println(options.toString());
        return serviceClient.sendReceive(operation);
    }
}
```

BIBLIOGRAPHY

[BAV05] Nick Bassiliades, Dimosthenis Anagnostopoulos, and Ioannis Vlahavas. Web Service Composition Using a Deductive XML Rule Language. *Distributed and Parallel Databases*, 17(2):135–178, 2005. 59

[Bay03] Thomas Bayer. Apache Axis: Architektur und Erweiterbarkeit, 2003. `http://www.oio.de/public/xml/axis-extensibility.htm` (access: 29.05.2006). 39

[BBC+04] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Ashok Malhotra, Anthony Nadalin, Nataraj Nagaratnam, Mark Nottingham, Hemma Prafullchandra, Claus von Riegen, Chris Sharp, and John Shewchuk. *Web Services Policy Framework (WS-Policy)*, 2004. `http://www-128.ibm.com/developerworks/library/specification/ws-polfram/`. 25

[BBC+05] Ruslan Bilorusets, Don Box, Luis Felipe Cabrera, Doug Davis, Donald Ferguson, , Tom Freund, Mary Ann Hondo, John Ibbotson, Lei Jin, Chris Kaler, Amelia Lewis, Rodney Limprecht, Steve Lucco, Don Mullen, Anthony Nadalin, Mark Nottingham, David Orchard, Jamie Roots, Shivajee Samdarshi, John Shewchuk, and Tony Storey. *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)*, 2005. `http://www-128.ibm.com/developerworks/library/specification/ws-rm/`. 33, 36

[BS02a] François Bry and Sebastian Schaffert. A gentle introduction into xcerpt, a rule-based query and transformation language for xml. In *Proceedings of International Workshop on Rule Markup Languages for Business Rules on the Semantic Web, Sardinia, Italy (14th June 2002)*, 2002. 48

[BS02b] François Bry and Sebastian Schaffert. Towards a declarative query and transformation language for xml and semistructured data: Simulation unification. In *Proceedings of International Conference on Logic Programming, Copenhagen, Denmark (29th July–1st August 2002)*, volume 2401 of *LNCS*. Springer-Verlag, 2002. 47

[BS04] Thomes Bayer and Andreas Spall. Die WS-* Spezfikationen, 2004. `http://www.oio.de/public/xml/web-service-specifications.htm` (access: 11.04.2006). 23

[CCF+05a] Luis Felipe Cabrera, George Copeland, Max Feingold, Robert W Freund, Tom Freund, Sean Joyce, Johannes Klein, David Langworthy, Mark Little, Frank Leymann, Eric Newcomer, David Orchard, Ian Robinson, Tony Storey, and Satish Thatte. *Web Services Business Activity Framework (WS-BusinessActivity)*, 2005. `http://www-128.ibm.com/developerworks/library/specification/ws-tx/`. 31, 33, 34

[CCF+05b] Luis Felipe Cabrera, George Copeland, Robert W Freund, Tom Freund, Jim Johnson, Sean Joyce, Chris Kaler, Johannes Klein, David Langworthy, Mark Little, Anthony Nadalin, Eric Newcomer, David Orchard, Ian Robinson, John Shewchuk, and Tony Storey. *Web Services Coordination (WS-Coordination)*, 2005. `http://www-128.ibm.com/developerworks/library/specification/ws-tx/`. 28, 29

[CCF+05c] Luis Felipe Cabrera, George Copeland, Robert W Freund, Tom Freund, Jim Johnson, Sean Joyce, Chris Kaler, Johannes Klein, David Langworthy, Mark Little, Anthony Nadalin, Eric Newcomer, David Orchard, Ian Robinson, Tony Storey, and Satish Thatte. *Web Services Atomic Transaction (WS-AtomicTransaction)*, 2005. `http://www-128.ibm.com/developerworks/library/specification/ws-tx/`. 30, 31, 32

[CCJL04] Luis Felipe Cabrera, George Copeland, Jim Johnson, and David Langworthy. Coordinating web services activities with ws-coordination, ws-atomictransaction, and ws-businessactivity. *MSDN Library*, 2004. `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/wsacoord.asp` (access: 11.05.2006). 29

[CDR05] Dominic Cooney, Marlon Dumas, and Paul Roe. A programming language for web service development. In *ACSC '05: Proceedings of the Twenty-eighth Australasian conference on Computer Science*, pages 143–150, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc. 60

[CFF+04] Erik Christensen, Donald Ferguson, Jeffrey Frey, Marc Hadley, Chris Kaler, David Langworthy, Frank Leymann, Brad Lovering, Steve Lucco, Steve Millet, Nirmal Mukhi, Mark Nottingham, David Orchard, John Shewchuk, Eugène Sindambiwe, Tony Storey, Sanjiva Weerawarana, and Steve Winkler. *Web Services Addressing (WS-Addressing)*, 2004. `http://www.w3.org/Submission/ws-addressing/`. 26, 27

[CJ02] David A. Chappell and Tyler Jewell. *Java Web Services*. O'Reilly, 2002. 5, 6, 8, 10, 38

[Cod06] Codehaus. Stack Comparison, 2006. `http://xfire.codehaus.org/Stack+Comparison` (access: 06.06.2006). 41, 43, 44

[Dod04] Mahesh H. Dodani. From objects to services: A journey in search of component reuse nirvana. *Journal of Object Technology*, 3(8):49–54, 2004. `http://www.jot.fm/issues/issue_2004_09/column5`. 23

[FGK03] Daniela Florescu, Andreas Grünhagen, and Donald Kossmann. XL: an XML programming language for Web service specification and composition. *Computer Networks*, 42(5):641–660, 2003. 60

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures.* dissertation, University of California, ICS, Irvine, 2000. 16

[FOS04] Mary Fernández, Nicola Onose, and Jérôme Siméon. Yoo-Hoo!: building a presence service with XQuery and WSDL. In *Proc. ACM SIGMOD*, pages 911–912, New York, NY, USA, 2004. ACM Press. 59

[GBSK03] Ping Guo, Julie Basu, Mark Scardina, and K. Karun. Parsing XML Efficiently, 2003. `http://www.oracle.com/technology/oramag/oracle/03-sep/o53devxml.html`. 37

[GDa06a] *Google Data APIs Protocol*, 2006. `http://code.google.com/apis/gdata/protocol.html` (access: 18.04.2006). 18

[GDa06b] *Google Data APIs Protocol*, 2006. `http://code.google.com/apis/gdata/calendar.html` (access: 18.04.2006). 18, 19

[HZ03] Manfred Hein and Henner Zeller. *Java Web Services.* Addision-Wesley, 2003. 5, 6, 8, 10

[kS90] Robert A. kowalski and Fariba Sadri. Logic programs with exceptions. pages 598–613, 1990. 57

[LS98] Jun Lang and David B. Stewart. A study of the applicability of existing exception-handling techniques to component-based real-time software technology. *ACM Trans. Program. Lang. Syst.*, 20(2):274–301, 1998. 57

[LYK+00] Seungll Lee, Byung-Sun Yang, Suhyun Kim, Seongbae Park, Soo-Mook Moon, Kemal Ebcioğlu, and Erik Altman. Efficient java exception handling in just-in-time compilation. In *JAVA '00: Proceedings of the ACM 2000 conference on Java Grande*, pages 1–8, New York, NY, USA, 2000. ACM Press. 57

[MÖ3] Florian Müller. Spheon JSOAP, 2003. `http://soap.fmui.de/spheon-jsoap.pdf`. 41

[MB] David Megginson and David Brownell. Events vs. Trees. `http://www.saxproject.org/event.html` (access: 02.06.2006). 37

[MH04] Richard Monson-Heafel. *J2EE Web Services.* Addision-Wesley, 2004. 6, 11, 38

[O'R03] Tim O'Reilly. REST vs. SOAP at Amazon, 2003. `http://www.oreillynet.com/pub/wlg/3005` (access: 23.04.2006). 16

[OS04] Nicola Onose and Jérôme Siméon. XQuery at your Web Service. In *Proc. Int. World Wide Web Conf.*, pages 603–611, New York, NY, USA, 2004. ACM Press. 59

[Sch04] Sebastian Schaffert. *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. Dissertation/Ph.D. thesis, Institute of Computer Science, LMU, Munich, 2004. PhD Thesis, Institute for Informatics, University of Munich, 2004. 47, 49, 50, 54, 55

[SOA03] *SOAP Version 1.2 Part 1: Messaging Framework*, 2003. `http://www.w3.org/TR/soap12-part1/`. 10

[SOA06] Serviceorientierte Architektur, 2006. `http://de.wikipedia.org/w/index.php?title=Serviceorientierte_Architektur&oldid=19343473` (access: 24.07.2006). 5

[StA] Original Java Specification Request (JSR). `http://jcp.org/en/jsr/detail?id=173` (access: 02.06.2006). 38

[SUN] SUN. SOAP with Attachments API for Java(TM) 1.3. `https://saaj.dev.java.net/source/browse/*checkout*/saaj/saaj-ri/docs/index.html` (access: 05.06.2006). 38

[Tec06] IONA Technologies. Celtix-Projekt von IONA und ObjectWeb bietet Open-Source-ESB der Enterprise-Klasse, 2006. `http://www.iona.com/de/releases/080506_celtix.html` (access: 05.06.2006). 41

[Tid01] Doug Tidwell. UDDI4J: Matchmaking for Web services, 2001. `http://www-106.ibm.com/developerworks/webservices/library/ws-uddi4j.html` (access: 06.06.2006). 40

[Win99] Dave Winer. *XML-RPC Specification*, 1999. `http://www.xmlrpc.com/spec` (access: 16.02.2006). 13

[WS-04] *Web Services Reliable Messaging TC WS-Reliability 1.1*, 2004. `http://www.oasis-open.org/committees/wsrm/`. 33

[WS-06] *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, 2006. `http://www-128.ibm.com/developerworks/library/specification/ws-secure/`. 27, 28