Information Systems 2

4. Distributed Information Systems II: Web Services

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Business Economics and Information Systems
& Institute for Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

Web Service Protocol Stack

layer	task	examples
(Service) Transport Protocol	transport messages	HTTP, SMTP, FTP
(XML) Messaging Protocol	encode messages	XML-RPC, WS-Addressing, SOAP
(Service) Description Protocol	describe public interface	WSDL
(Service) Discovery Protocol	discover services	UDDI

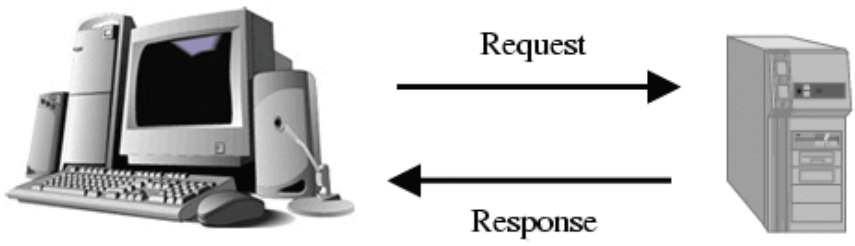
1. Message Transport: HTTP

2. Message Encoding: SOAP

3. Service Publishing: WSDL

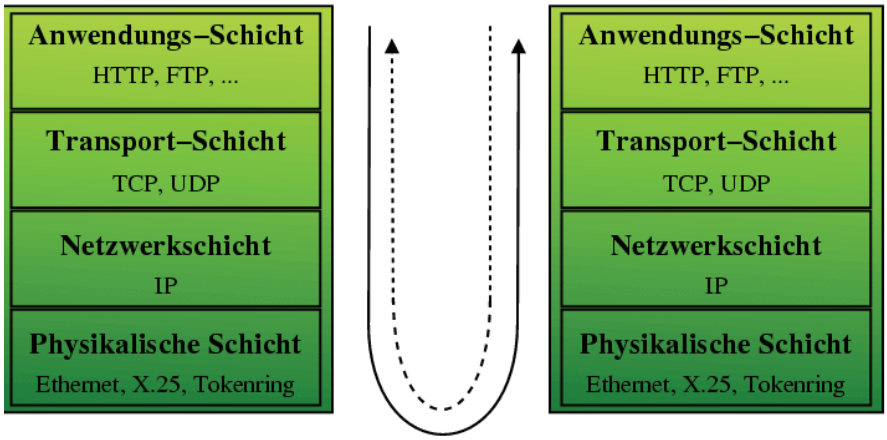
4. Implementing Web Services: Axis2 engine

Request and Response



Open Systems Interconnection Basic Reference Model (OSI Model)

Communication is structured in so-called **network layers**:



The full OSI model contains 7 layers:

application, presentation, session, transport, network, data Link, and physical layer

some of which often are lumped together in 5 or 4 layers as above.

Hypertext Transfer Protocol (HTTP)

- HTTP is a protocol for the exchange of information via a **request/response paradigma**.
- HTTP is coordinated by
  - W3C and
  - the Internet Engineering Task Force (IETF).
- Different versions of HTTP are described in a series of **Request for Comments** (**RFC**s), most actually
  - HTTP 1.1 in RFC 2616 from June, 1999.
- HTTP messages consist of
  - a **response/request line**,
  - optional **header lines**
  - an **entity body**
    (delimited by an empty line from the headers)

Hypertext Transfer Protocol (HTTP) / Syntax

⟨*request*⟩ := ⟨*request line*⟩
(⟨*general header*⟩ | ⟨*request header*⟩ | ⟨*entity header*⟩)*
⟨*CRLF*⟩
⟨*entity body*⟩

⟨*response*⟩ := ⟨*response line*⟩
(⟨*general header*⟩ | ⟨*response header*⟩ | ⟨*entity header*⟩)*
⟨*CRLF*⟩
⟨*entity body*⟩

⟨*request line*⟩ := ⟨*method*⟩ ⟨*SP*⟩ ⟨*request uri*⟩ ⟨*SP*⟩ ⟨*http version*⟩ ⟨*CRLF*⟩

⟨*response line*⟩ := ⟨*http version*⟩ ⟨*SP*⟩ ⟨*status code*⟩ ⟨*SP*⟩ ⟨*reason phrase*⟩ ⟨*CRLF*⟩

where ⟨*SP*⟩ denotes a space
and ⟨*CRLF*⟩ a newline.

Most common Request Headers

Host
server request is sent to.

From
client response originated from.

User-Agent
browser used on the client.

Accept, Accept-Charset, Accept-Encoding, Accept-Language
charset, encoding and language prefered by the client.

Referer
URI of resource containing the link to the request URI.

Authorization
login and password information.

If-modified-since
conditional request.

Most common Entity and General Headers

Entity headers:

Content-Encoding, Content-Length, Content-Type, Content-Language
encoding, length, type and language of content entity returned.

Last-modified
timestamp entity last has been modified.

Expires
timestamp until entity is valid.

General headers:

Date
date and time of request / response.

Request Methods

GET
- Requests the entity identified by the request URI.
- Signals that the resource should not be altered by the operations.

POST
- Submits data to the specified resource and requests a result entity in return.
- The data is sent in the entity body of the request.

PUT
- Uploads an entity for storage under the request URI.

DELETE
- Deletes the entity identified by the request URI.

as well as the more specialized methods HEAD, TRACE, OPTIONS and CONNECT.

HTTP Status Codes

The success of the request is signaled by a status code:

code	meaning
⋮	⋮
200	OK
201	Created
⋮	⋮
301	Moved Permanently
⋮	⋮
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
⋮	⋮
500	Internal Server Error
⋮	⋮

Example HTTP Headers

```
1 GET /index.html HTTP/1.1
2 Host: localhost:8090
3 User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.8.1.14) Gecko/20080410 SUSE/2.0.0.14-0.1 Firefox/2.0.0.14
4 Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
5 Accept-Language: en-us,en;q=0.5
6 Accept-Encoding: gzip,deflate
7 Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
8 Keep-Alive: 300
9 Connection: keep-alive
```

Figure 3: Request by Firefox

```
1 GET /index.html HTTP/1.1
2 User-Agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux) KHTML/3.5.7 (like Gecko) SUSE
3 Accept: text/html, image/jpeg, image/png, text/*, image/*, */*
4 Accept-Encoding: x-gzip, x-deflate, gzip, deflate
5 Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5
6 Accept-Language: en, de
7 Host: localhost:8090
8 Connection: Keep-Alive
```

Figure 4: Request by Konqueror

Example HTTP Headers

```
1 HTTP/1.1 200 OK
2 Server: Apache-Coyote/1.1
3 ETag: W/"10852-1213607922000"
4 Last-Modified: Mon, 16 Jun 2008 09:18:42 GMT
5 Content-Type: text/html
6 Content-Length: 10852
7 Date: Mon, 16 Jun 2008 20:59:45 GMT
8 Connection: keep-alive
9
10 <html lang="de">
11 <head>
12 ...
13 </head>
14 ...
15 </html>
```

Figure 5: Response by Tomcat

- SOAP defines a format for exchanging structured and typed information between peers in a decentralized, distributed environment, consisting of:
  - **Messaging Framework**: Processing Model, Extensibility Model, Protocol Binding Framework, Message Construct.
  - **Adjuncts**: SOAP Data Model, SOAP Encoding, SOAP RPC Representation, a Convention for Describing Features and Bindings, Message Exchange Patterns and Features, SOAP HTTP Binding.
- SOAP is an XML application. Its namespace is
  http://www.w3.org/2003/05/soap-envelope
- SOAP can use the XML Schema type system.
- SOAP is managed by the W3C,
  its actual version is SOAP 1.2 (April 27, 2007).
- SOAP originally was the acronym for *Simple Object Access Protocol*, but this name is no longer used.

Core SOAP Components

1. **SOAP Message Format**:
   — provides overall structure (**envelope**) of request/response messages.
2. **SOAP Data Model**:
   — conceptual type system for arguments and return values of procedures.
3. **SOAP Encoding**:
   — XML representation of the SOAP Data Model.
4. **SOAP Remote Procedure Calls (RPCs)**:
   — how to specify method calls.
5. **SOAP HTTP Binding**:
   — how to transport SOAP messages via HTTP.

Only the SOAP Envelope is mandatory,
all other components can be replaced by other specifications independently.

SOAP Messages

The root element of a SOAP message:

<Envelope>
Content: ⟨*Header*⟩ ?
⟨*Body*⟩
</Envelope>

The optional header carries information about the processing of the message by intermediary SOAP nodes. (not handeled here)

The mandatory body element:

- contains any number of children elements.
- which are web service-specific (i.e., not described by SOAP!).

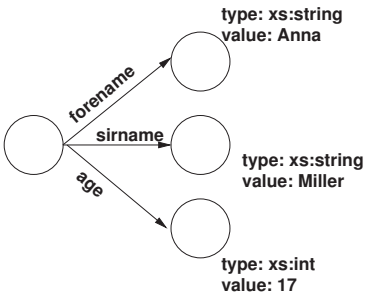
SOAP Data Model (1/2)

The SOAP data model models data items as **directed labeled graphs**.

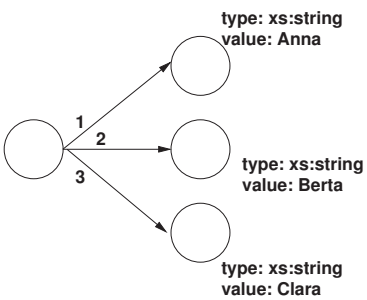
It contains the following data items:

simple value: a simple lexical value.
a node with a **lexical value** and an optional **XML schema type**.

struct: a compound of named parts.
a node with **labeled outgoing edges**.

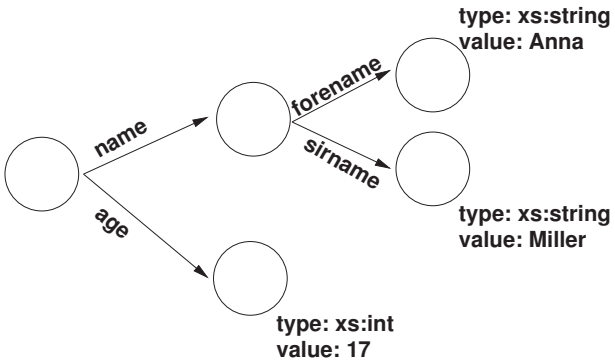


array: a compound of indexed parts.
a node with **numbered outgoing edges**.



SOAP Data Model (2/2)

The target nodes of structs and arrays can be any valid SOAP data items, i.e., simple values as well as themselves structs or arrays.



SOAP Encoding

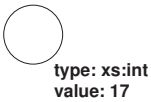
The SOAP encoding provides a representation for SOAP data instances as XML:

- Each **edge** of a SOAP data instance is represented as **element**. Its name is the label of the edge (structs) or arbitrary (arrays).
- The **SOAP type** of the target node (optionally) can be expressed by the attribute

  nodeType

  as: “simple”, “struct” or “array”.
- **Simple values** of target nodes are expressed as **character content** of the edge element, their type by the attribute

  xsi:type



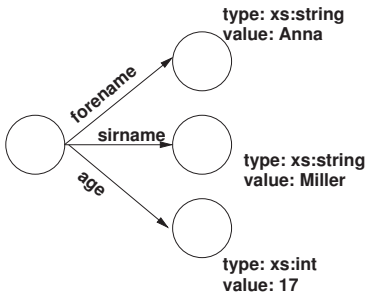

```
1 <XXX enc:nodeType="simple" xsi:type="xs:int">17</XXX>
```

The names for the SOAP encoding primitives belong to the namespace

http://www.w3.org/2003/05/soap-encoding

SOAP Encoding / Structs

– **Structs** as target nodes are expressed as **sequence of outgoing named edges**.




```
1 <XXX enc:nodeType="struct" xmlns:app="http://www.ismll.de/examples/soap/encoding1">
2  <app:forename enc:nodeType="simple" xsi:type="xs:string">Anna</app:forename>
3  <app:sirname enc:nodeType="simple" xsi:type="xs:string">Miller</app:sirname>
4  <app:age enc:nodeType="simple" xsi:type="xs:int">17</app:age>
5 </XXX>
```

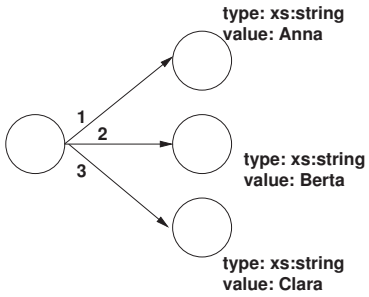

SOAP Encoding / Arrays

– **Arrays** as target nodes are expressed as **sequence of unnamed edges**.

– The element name is arbitrary, the position denotes the index.

– The type of the array element and the size of the array can be specified by the attributes

itemType
arraySize




```
1 <XXX enc:nodeType="array" enc:itemType="xs:string" enc:arraySize="3">
2  <Y>Anna</Y>
3  <Y>Berta</Y>
4  <Y>Clara</Y>
5 </XXX>
```

SOAP Encoding / References

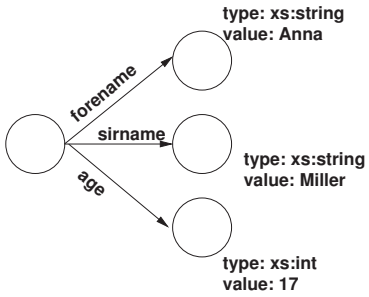
– Instead of provided as element content, target nodes can also be pointed to by the attribute

ref

– For each ref, there must an element with attribute

id

having the same value in the same envelope.




```
1 <XXX enc:nodeType="struct" xmlns:app="http://www.ismll.de/examples/soap/encoding1">
2   <app:forename enc:ref="Annas forename"/>
3   <app:sirname enc:ref="Anna/sirname"/>
4   <app:age enc:ref="v13"/>
5 </XXX>
6 <Y enc:id="Annas forename" enc:nodeType="simple" xsi:type="xs:string">Anna</Y>
7 <Y enc:id="Anna/sirname" enc:nodeType="simple" xsi:type="xs:string">Miller</Y>
8 <Y enc:id="v13" enc:nodeType="simple" xsi:type="xs:int">17</Y>
```


SOAP Encoding / env:encoding-style

To signal that a data item has been encoded using the SOAP encoding, one has to set the attribute `env:encoding-style` to the namespace URI

http://www.w3.org/2003/05/soap-encoding

Do not confuse this setting with the namespace setting for the names of the SOAP encoding namespace.

SOAP Remote Procedure Calls (RPC)

To invoke an SOAP RPC, the following information is needed:

- The address of the target SOAP node.
- A procedure or method name.
- Arguments passed to the procedure as identity/value pairs.
- Property values of the binding.
- Header data (optional).

The namespace for SOAP rpc primitives is

http://www.w3.org/2003/05/soap-rpc

SOAP Remote Procedure Calls (RPC)

An **RPC invocation** is encoded as single struct with the in or in/out arguments as parts, i.e.:

- Encoded as element in the SOAP body.
  The name of the element is the name of the procedure called.
- Each in or in/out argument as outgoing edge named by the argument name, i.e., as nested element.

An **RPC response** is encoded as single struct with the out or in/out arguments and the result as parts:

- Encoded as element in the SOAP body.
  The name of the element is arbitrary.
- Each out or in/out argument as edge named by the argument name.
- If the result type is not void, an outgoing edge named

  rpc:result

Example / Request

Assume there is a webservice at the address “http://localhost:8080/axis2/services/CalculatorService” offering a procedure “add” that takes two integer arguments “i1” and “i2” and returns the sum of both values.

To invoke this service, we could sent the following SOAP message:

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://www.w3.org/2003/05/soap-encoding"
  xmlns:calc="http://ismll.de/examples/soap/Calculator">
<env:Body>
  <calc:add env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
    <calc:i1 enc:nodeType="simple" xsi:type="xs:int">7</calc:i1>
    <calc:i2 enc:nodeType="simple" xsi:type="xs:int">8</calc:i2>
  </calc:add>
</env:Body>
</env:Envelope>
```

Figure 14: A simple SOAP request.

Example / Response

The service could respond with the following SOAP message:

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://www.w3.org/2003/05/soap-encoding"
  xmlns:calc="http://ismll.de/examples/soap/Calculator"
  xmlns:rpc="http://www.w3.org/2003/05/soap-rpc">
<env:Body>
  <calc:response env:encodingStyle="http://www.w3.org/2003/05/soap-encoding">
    <rpc:result enc:nodeType="simple" xsi:type="xs:int">15</rpc:result>
  </calc:response>
</env:Body>
</env:Envelope>
```

Figure 15: A simple SOAP response.

SOAP HTTP Binding

SOAP allows different underlying protocolls for transporting the message.

The most common one is HTTP via the POST method:

- Usually with media-type **application/soap+xml** (specified in the HTTP header field **Content-Type**).

Practically, to sent our SOAP request message, we could use a download tool such as wget:

```
wget --post-file=request-manual-all.xml
  --header='Content-Type: application/soap+xml'
  http://localhost:8080/axis2/services/CalculatorService
```

(The HTTP header field “SOAPAction” from SOAP 1.1 is obsolete in SOAP 1.2. The action optionally can be encoded as the action feature of the MIME type.)

A remark about SOAP 1.1

In older examples you will also find the old SOAP v1.1 namespace identifiers

```
http://schemas.xmlsoap.org/soap/envelope/
http://schemas.xmlsoap.org/soap/encoding/
```

that should no longer be used.

1. Message Transport: HTTP
2. Message Encoding: SOAP
3. Service Publishing: WSDL
4. Implementing Web Services: Axis2 engine

WSDL: W3C Status

WSDL 2.0 - The Web Service Definition Language - is a W3C Recommendation as of 26 June 2007.

WSDL ...

- describes Services as a set of Endpoints
- defines abstract operations and messages that are anound to concrete protocols and formats

`http://www.w3.org/TR/wsdl20-primer/`
WSDL 1.1: `http://www.w3.org/TR/wsdl`

WSDL 1.1: Component overview

WSDL 1.1 consists of the following elements

- types, which provides data type definitions used to describe the messages exchanged.
- message, which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.
- portType, which is a set of abstract operations. Each operation refers to an input message and output messages.
- binding, which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
- port, which specifies an address for a binding, thus defining a single communication endpoint.
- service, which is used to aggregate a set of related ports.

WSDL 2.0: Component overview

WSDL 2.0 has a more "onion-like" structure:

- Type (same as in 1.1)
- Interfaces define the abstract interface of a Web service as a set of abstract operations
- Bindings specify concrete message formats and transmission protocol details for an interface
- Services define, where the service can be accessed

a broad overview of WSDL

A server offering Web Services using WSDL may use SOAP as message encoding.

A WSDL Document defines types and operations that "are aggregated until they are somehow accessible". In the simplest and most common case: as a Web URL.

Offers great flexibility to exchange components.

WSDL 1.1 Skeleton

```
<definitions name="StockQuote" ... >

   <types> ... </types>

   <message name="GetLastTradePriceInput"> ... </message>

   <portType name="StockQuotePortType"> ... </portType>

   <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
      ...
   </binding>

   <service name="StockQuoteService">
     ...
   </service>

</definitions>
```

Types

WSDL Types are defined by utilizing XML Schema:


```
<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
        xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="TradePriceRequest">
      <complexType>
        <all>
          <element name="tickerSymbol" type="string"/>
        </all>
      </complexType>
    </element>
    <element name="TradePrice">
      <complexType>
        <all>
          <element name="price" type="float"/>
        </all>
      </complexType>
    </element>
  </schema>
</types>
```

WSDL 1.1-by-example 1: Messages

```
<message name="GetLastTradePriceInput">
   <part name="body" element="xsd1:TradePriceRequest"/>
</message>

<message name="GetLastTradePriceOutput">
   <part name="body" element="xsd1:TradePrice"/>
</message>
```


WSDL 1.1-by-example 2: portTypes

```
<portType name="StockQuotePortType">
   <operation name="GetLastTradePrice">
     <input message="tns:GetLastTradePriceInput"/>
     <output message="tns:GetLastTradePriceOutput"/>
   </operation>
</portType>
```

WSDL 1.1-by-example 3: bindings

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
   <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="GetLastTradePrice">
     <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
     <input>
        <soap:body use="literal"/>
     </input>
     <output>
        <soap:body use="literal"/>
     </output>
   </operation>
</binding>
```


WSDL 1.1-by-example 4: service

```
<service name="StockQuoteService">
   <documentation>My first service</documentation>
   <port name="StockQuotePort" binding="tns:StockQuoteBinding">
     <soap:address location="http://example.com/stockquote"/>
   </port>
</service>
```

Paradigms of Web Service Creation

WSDL only defines *what*, *where*, and *how* a service can be accessed/invoked.

Two approaches when implementing Web Services can be distinguished:

1. Contract-First
2. Code-First

1. Message Transport: HTTP

2. Message Encoding: SOAP

3. Service Publishing: WSDL

4. Implementing Web Services: Axis2 engine

Implementing a Web Service

SOAP tells you how to use a web service technically, when you know,

- where it is,
- which methods it offers and
- with which signatures.

SOAP does not tell you at all how to implement a web service.

How to implement a web service depends on the **web service engine**, e.g.,

- Apache Axis2, http://ws.apache.org/axis2/ (v1.6.0, May. 2011)

Installing Apache Axis2

```
unzip axis2-1.4-bin.zip
cd axis2-1.4/
chmod a+x bin/axis2server.sh
./bin/axis2server.sh
```

Now the Axis2 engine is running on port 8080.

You can get a list of deployed web services by visiting

http://localhost:8080/

with a web browser.

Alternatively, download the .war-file and deploy it in a servlet container of your choice.

Deploying a Web Service in Axis2 (1/2)

A minimal web service implementation is made from just two files:

1. The implementation `Calculator.java`:

```
1 public class Calculator {
2   public int add(int i1, int i2) {
3     return i1 + i2;
4   }
5
6   public int subtract(int i1, int i2) {
7     return i1 - i2;
8   }
9 }
```

2. An Axis-specific **webservice descriptor** `services.xml`:

```
1  <service name="CalculatorService" scope="application" targetNamespace="http://ismll.de/examples/soap/Calculator">
2    <description>Calculator</description>
3    <messageReceivers>
4      <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
5                       class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"/>
6      <messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out"
7                       class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
8    </messageReceivers>
9    <schema schemaNamespace="http://ismll.de/examples/soap/Calculator"/>
10   <parameter name="ServiceClass">Calculator</parameter>
11 </service>
```


Deploying a Web Service in Axis2 (2/2)

Web services can be archived in **Axis archives** (.aar; jar-archives):

- containing the classes in the root and
- the webservice descriptor `services.xml` in the subdirectory `META-INF`.

```
> jar tf Calculator.aar
META-INF/
META-INF/MANIFEST.MF
Calculator.class
META-INF/services.xml
```

Axis archives can be deployed by simply copying them to the **Axis2 services repository**:

```
> cp Calculator.aar ~/ws/axis2-1.4/repository/services/
```

References

[ZTP05] Olaf Zimmermann, Mark Tomlinson, and Stefan Peuser. *Perspectives on Web Services — Applying SOAP, WSDL and UDDI to Real-World Projects*. Springer, 2005.