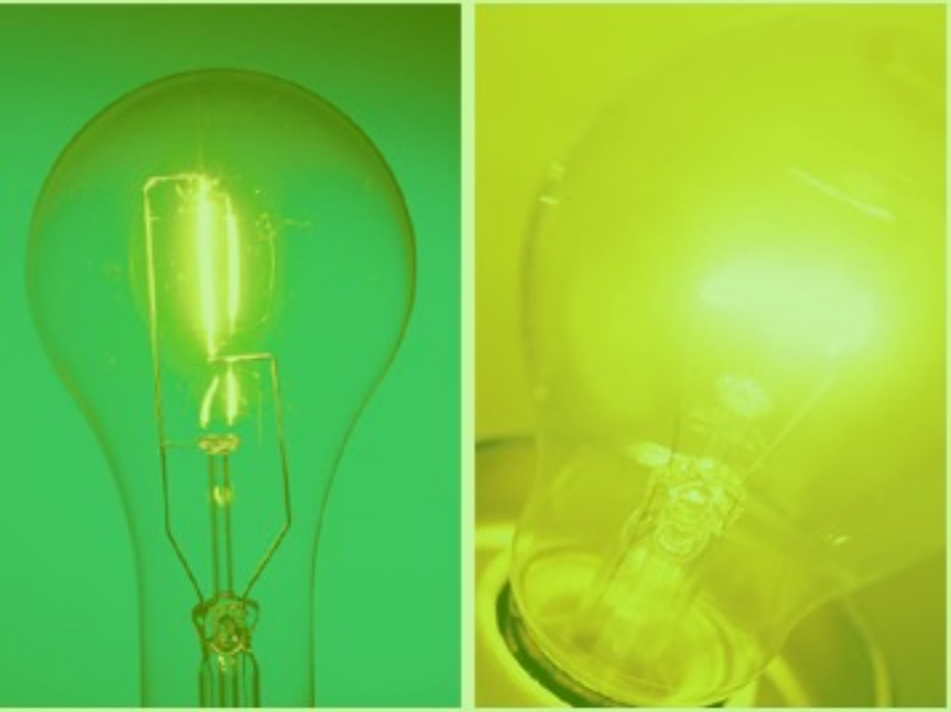
Web Services



Alfred Schmidt

Why Web Services?

The idea of Web services is easy:

Make (business) applications available over the Internet, so that they can easily be used in other applications.

We had comparable approaches before in the form of

- RPC (Remote Procedure Calls),
- DCE (OSF Distributed Computing Environment),
- CORBA (Common Object Request Broker Architecture),
- RMI (Java Remote Method Invocation) or
- DCOM (Distributed Component Object Model by Microsoft).

Why Web Services?

These technologies have some significant weak points:

- no sufficient acceptance for one of the standards
- close coupling
- high complexity
- programming language dependence
- binary data exchange

Why Web Services?

The advantages of Web services against these older technologies are

- easy to handle (especially against CORBA and DCOM)
- usually communication runs on HTTP (Hypertext Transfer Protocol), so that Web Services can be used on the Internet and in Intranet and Extranet environments. Other protocols like SMTP (Simple Mail Transfer Protocol) are possible.
- Web services are based on accepted standards
- Web services are accepted extensively by the software industry (for example Microsoft, Sun and IBM)
- almost every programming language is supported
- Web services are based on XML

Web Services Definition

Web services are encapsuled, distributed, loosely coupled and self-describing modular and reusable pieces of business logic, identified by a URI*, which are accessible through standard protocols such as HTTP and SMTP. Most often messages are exchanged using a family of XML interfaces like **SOAP**, **WSDL** and **UDDI**.

* **U**niform **R**esource **I**dentifier (URLs and URNs)

Who Uses Web Services?

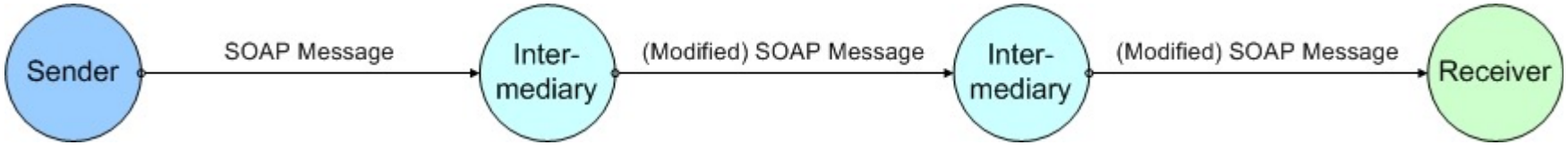
- Google (Web Search)
- Amazon (Search & Shopping Cart)
- SAP (SAP Enterprise)
- Microsoft (.NET Framework)
- IBM (Lotus Notes/Domino)
- almost any major company worldwide

Some Interesting Pages

- www.xmethods.net
- www.webserviceX.net
- www.soapclient.com/soaptest.html
- www.w3schools.com/webservices/
- w3.org
- ws.apache.org

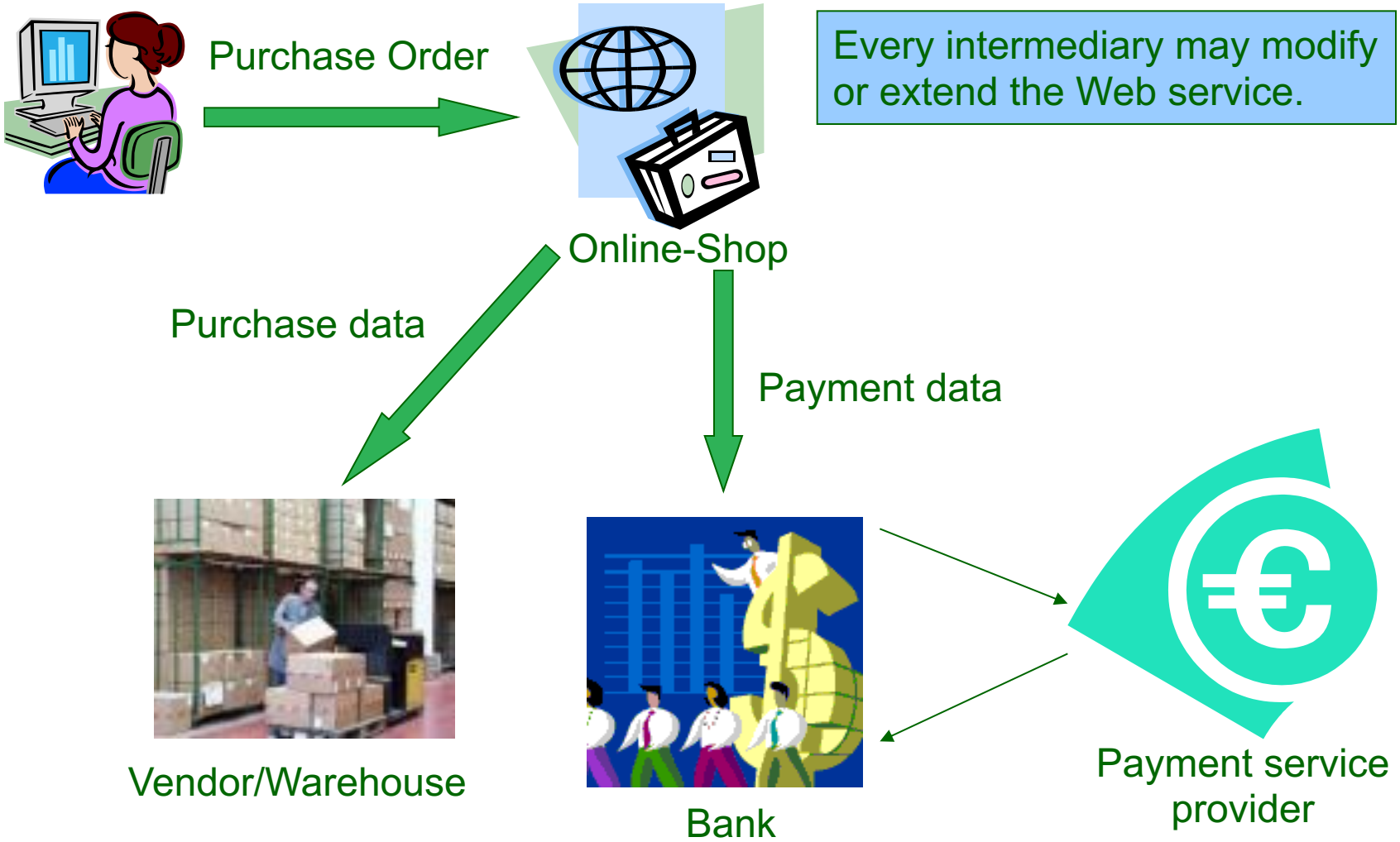
Web Services Actors

- Who‘s acting in a Web Services business scenario?



- Each actor is called a „SOAP node“
- The receiver is also called „Ultimate Receiver“ (the ulimate receiver processes the payload of a SOAP message)
- Each intermediary acts both as sender and receiver

Intermediaries Sample Scenario



Web Services Standards

- **XML**

  A meta language for the definition of markup languages.
- **SOAP**

  A lightweight protocol intended for exchange of structured and typed information in distributed networks.
- **WSDL**

  XML-based interface description language for Web services.
- **UDDI**

  The standard discovery service for Web services.

Standardization committees

- XML
  Version 1.1
  W3C (www.w3.org/XML)

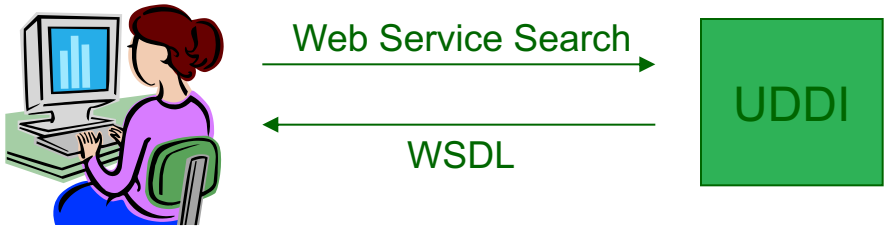
- SOAP
  Version 1.2
  W3C (www.w3.org/xp/Group)

- WSDL
  Version 2.0
  W3C (www.w3.org/ws/desc)

- UDDI
  Version 2 and Version 3.0.2
  OASIS (www.oasis-open.org)

Web Services Protocols

UDDI (Universal Description, Discovery and Integration)

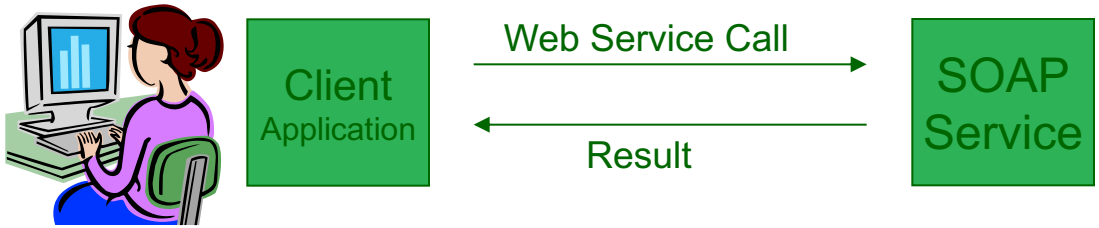


Web Services Directory

WSDL (Web Services Description Language)

Web Services Description

SOAP (Simple Object/Open Access Protocol*)



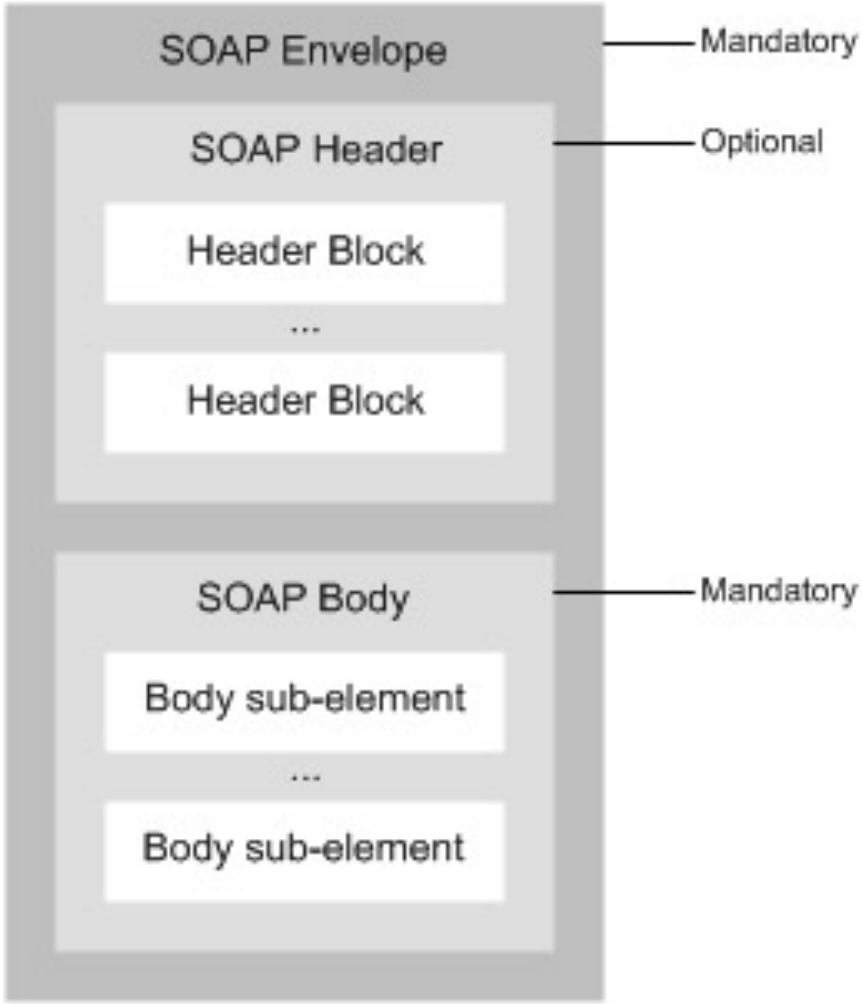
* since SOAP 1.2 no longer an acronym

SOAP

SOAP, the core technology in the area of Web services, is a lightweight protocol intended for exchanging structured and typed information in distributed networks.

SOAP works on top of well-known standards like HTTP and XML. This allows SOAP-enabled applications to traverse conventional* firewalls.

The SOAP header may contain routing information and therefore cannot be encrypted. The SOAP body contains the message payload.



* XML-enabled firewalls are necessary to check SOAP content.

SOAP Sample Message: Header

```
<!-- ENVELOPE -->
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope">

 <!-- HEADER -->
 <SOAP-ENV:Header>
   <r:reservation
       xmlns:r="http://flightbooking.example-air.com/reservation"
       SOAP-ENV:role="http://www.w3.org/2003/05/soap-envelope/role/next"
       SOAP-ENV:mustUnderstand="true">
     <r:reference>BH5NAB</r:reference>
     <r:dateAndTime>2004-06-03T22:19:00</r:dateAndTime>
   </r:reservation>
   <p:passenger xmlns:p="http://test.example-air.com/passengers"
       SOAP-ENV:role="http://www.w3.org/2003/05/soap-envelope/role/next"
       SOAP-ENV:mustUnderstand="true">
     <p:name>Alfred Schmidt Mr.</p:name>
     <p:name>Gisela Teichert Mrs.</p:name>
   </p:passenger>
 </SOAP-ENV:Header>
```

SOAP Sample Message: Body

```
<!-- BODY -->
<SOAP-ENV:Body>
  <i:itinerary xmlns:i="http://flightbooking.example-air.com/itinerary">
    <i:departure>
      <i:departing>Hamburg Luebeck (LBC)</i:departing>
      <i:arriving>Skavsta Stockholm (NYO)</i:arriving>
      <i:departureDate>2004-07-19</i:departureDate>
      <i:departureTime>15:45</i:departureTime>
    </i:departure>
    <i:return>
      <i:departing>Skavsta Stockholm (NYO)</i:departing>
      <i:arriving>Hamburg Luebeck (LBC)</i:arriving>
      <i:departureDate>2004-07-24</i:departureDate>
      <i:departureTime>13:55</i:departureTime>
    </i:return>
  </i:itinerary>
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

WSDL

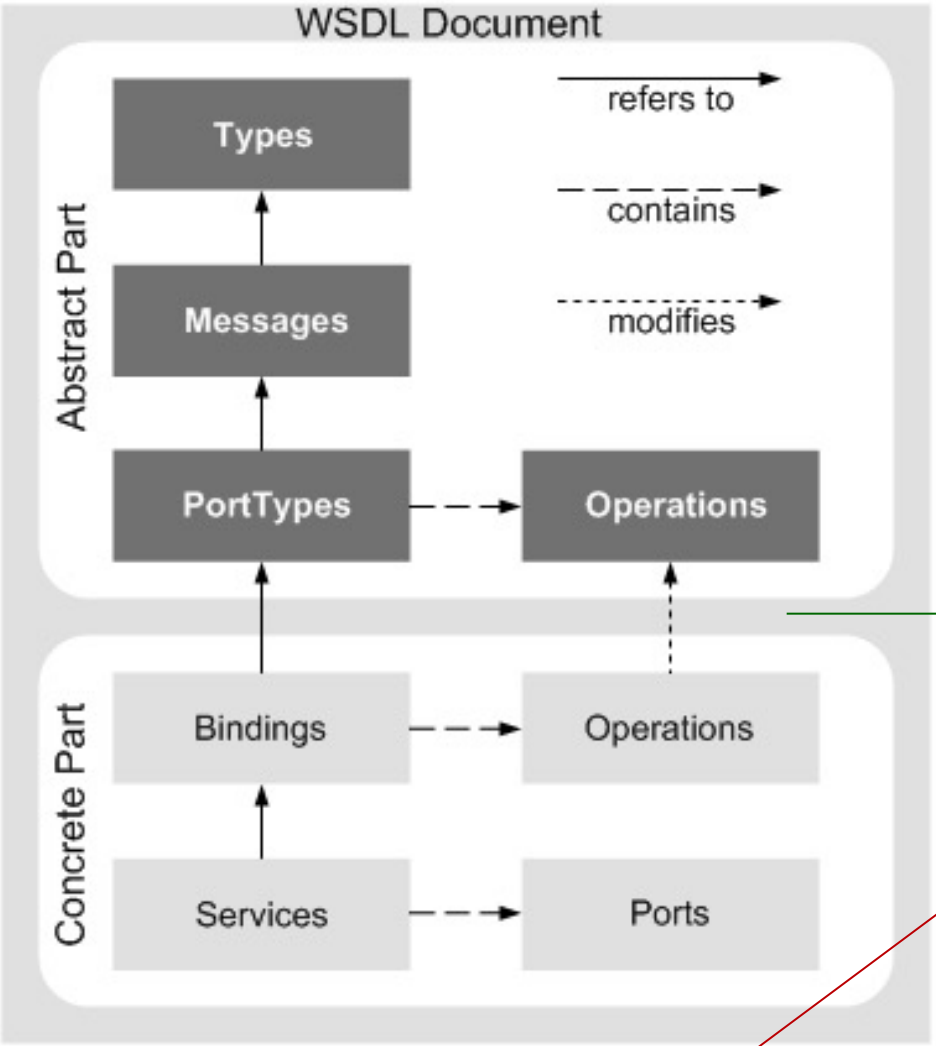
- Web Services Description Language

WSDL is (like IDL for CORBA) an interface description language. It consists of an

- abstract part describing the data types (optional*), messages, and input/output port types, and a
- concrete part describing the network protocols for invocation (binding) and the physical location (URL) of the service.

* <Types> contains only complex data types; primitive (built in) types are covered by XSD (XML Schema)

WSDL




```
<definitions>

    <types>
    </types>
    <messages>
    </messages>
    <portType>
        <operation>
        </operation>
    </portType>

    <binding>
        <operation>
        </operation>
    </binding>
    <service>
        <port>
        </port>
    </service>

</definitions>
```

real methods

real endpoint

UDDI

- Universal Description, Discovery and Integration

is the standard discovery (directory) service for Web services. Some UDDI pages:

- uddi.ibm.com
- uddi.microsoft.com
- uddi.sap.com
- uddi.ntt.com/uddi (Asian Pacific area)

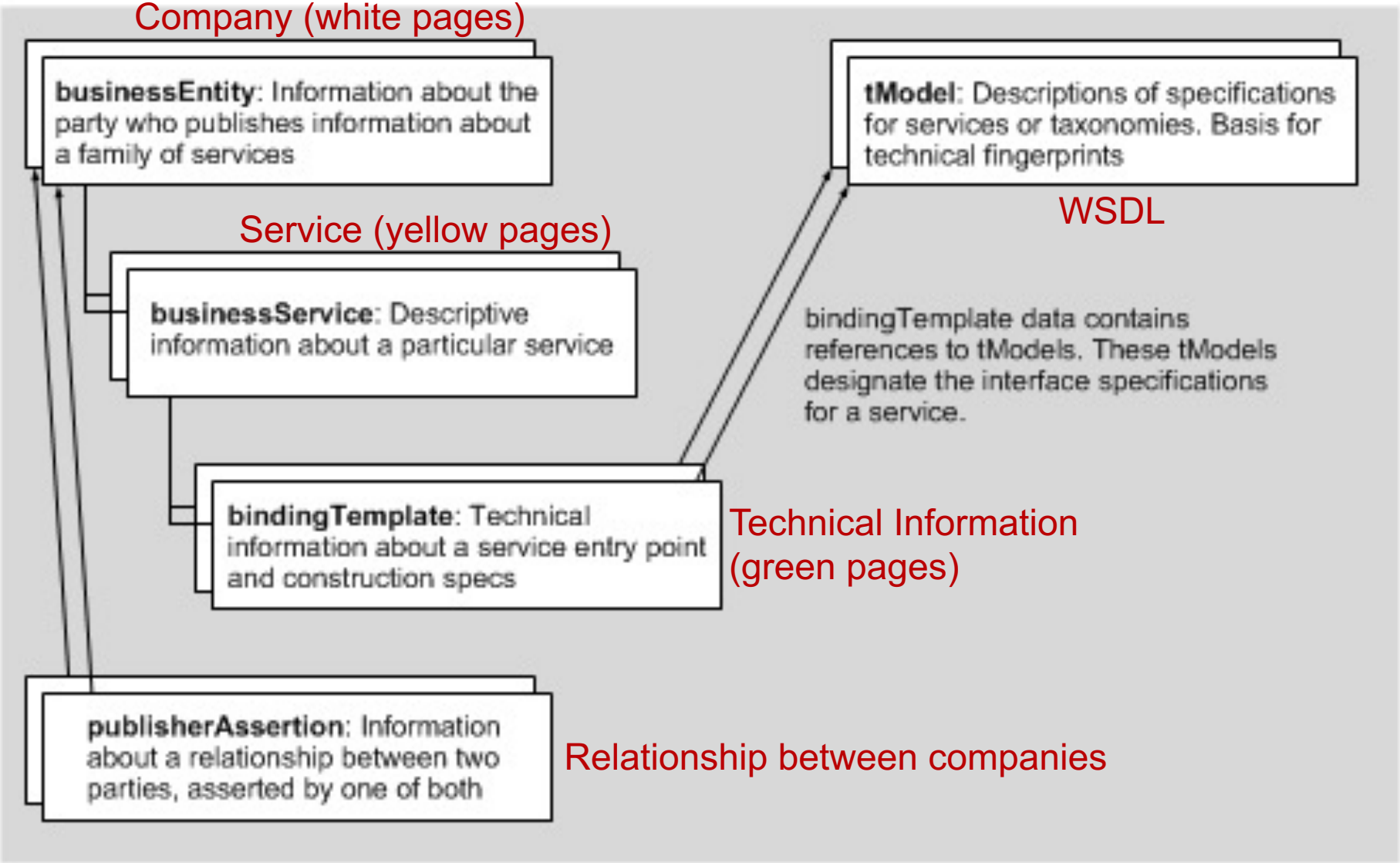
There is no broad acceptance for UDDI. As far as we know IBM quit their UDDI services.

UDDI

UDDI offers three types of inquiries:

- White Pages: names, descriptions and all contact data of companies.
- Yellow Pages: categorization like in the yellow pages for telephones.
- Green Pages: technical documentation, which describes the Web service.

UDDI


Company (white pages)
businessEntity: Information about the party who publishes information about a family of services
tModel: Descriptions of specifications for services or taxonomies. Basis for technical fingerprints
WSDL
Service (yellow pages)
businessService: Descriptive information about a particular service
bindingTemplate data contains references to tModels. These tModels designate the interface specifications for a service.
bindingTemplate: Technical information about a service entry point and construction specs
Technical Information (green pages)
publisherAssertion: Information about a relationship between two parties, asserted by one of both
Relationship between companies

SOAP Java Implementations

- IBM alphaWorks SOAP4J (not longer supported)
- Apache SOAP (based on SOAP4J; uses DOM*)
- Apache Axis (uses SAX** = better perfomance)
- Sun Java Web Services Developer Pack
- WebMethods Glue
- Systinet Server for Java (formerly WASP)
- SOAP with Attachments API for Java (SAAJ – Axis makes use of it)

* SAX = Simple API for XML
** DOM = Document Object Model

Axis

- Apache eXtensible Interaction System
- current Versions 1.4 (most often used) and 2 (completely re-designed)
- used by several companies: IBM, Borland, Macromedia, JBoss, Apple etc.
- Axis is high-grade extensible because of its handler concept

Axis offers:

- a simple stand-alone server
- a server which plugs into servlet engines such as Tomcat
- emitter tools for WSDL (WSDL2Java and Java2WSDL)
- a TCP/IP monitoring tool (tcpmon)

Axis Deployment

- two ways of deploying Web services:

- auto deployment (.jws files)
- deployment with the AdminClient using WSDD descriptors (a XML dialect), for example:

```
<deployment xmlns=… >
  <service name=„myService“ provider=… >
    <parameter name=„className“ value=„myClass“ />
    <parameter name=„allowedMethods“ value=„*“ />
  </service>
</deployment>
```

Axis Emitter Classes

- WSDL2Java and Java2WSDL
- WSDL2Java generates a Java proxy for the Web service (minimum 4 classes*):

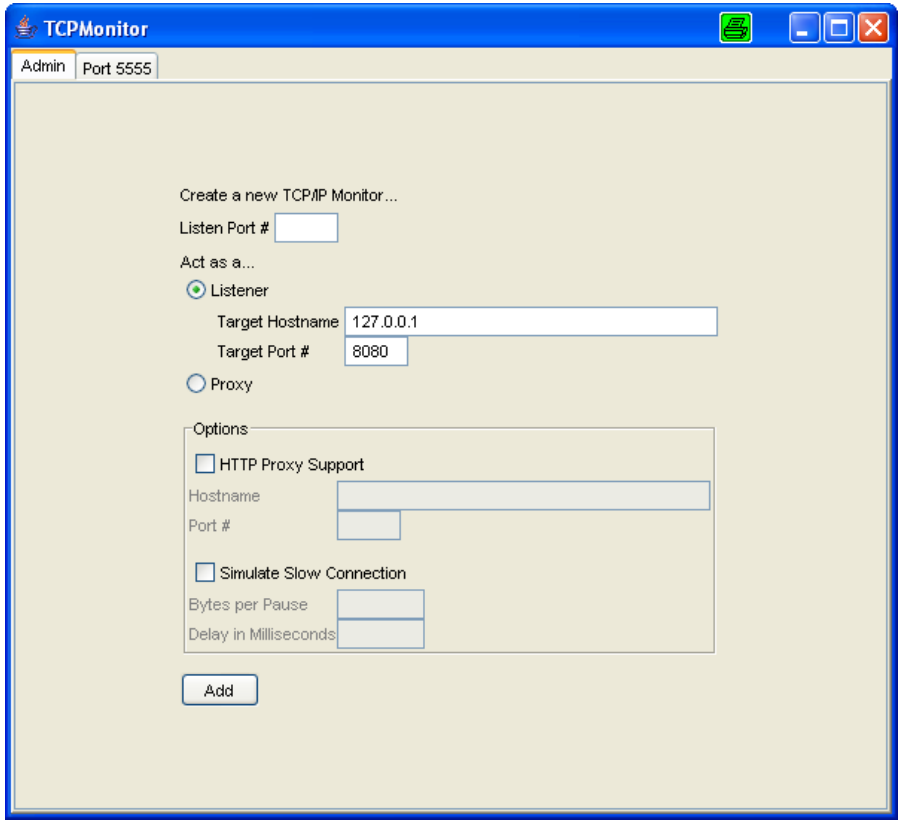
WSDL Element	Generated Class/Interface
`types`	one Java class for every type definition
`portType`	one Java interface for every portType element
`binding`	one stub class for every binding element
`service`	an extension of the interface `javax.xml.rpc.Service` and an implementation of the interface (locator)

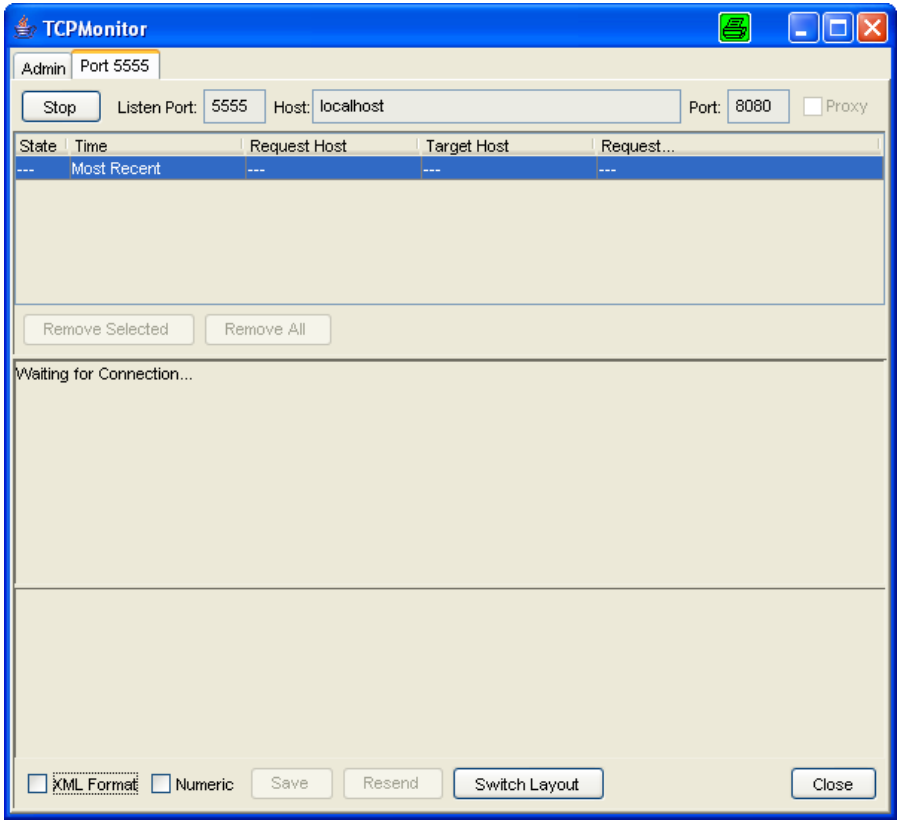
* For the Amazon Web services over 50 classes are generated!

Axis TCP/IP Monitor

org.apache.axis.utils.tcpmon

<listen on port> <host> <forward to port>





Exercises

1. Preparations 1 (Axis + WSDL2Java Plugin)
2. Barnes and Noble Book Price
3. Google Web Services API
4. Preparations 2 (Tomcat installation)
5. Create your own Web Service using Apache Tomcat and Axis
6. Test your own Web Service
7. Monitor HTTP request and response using Axis Tcpmon

Exercise #1

- Preparations 1:

Use our FTP server:

ftp://www.bwl.hs-bremerhaven.de

and download everything in pub/WebServices.

Unzip Axis, googleapi, jaf and javamail to d:\java.

Unzip com.myspotter… to the Eclipse plugin directory (d:\java\eclipse).

Exercise #2

- Barnes and Noble Book Price:

www.abundanttech.com: get the Barnes and Noble price by the ISBN book number.

Where can I obtain the WSDL file? (*)

How can I call this service in the address line of a browser? (**)

(*) http://www.abundanttech.com/WebServices/bnprice/bnprice.asmx?wsdl

(**) http://www.abundanttech.com/WebServices/bnprice/bnprice.asmx/GetBNQuote?sISBN=0679752455

Exercise #3

- Google Web Services API:

Google offers besides the WSDL document a special API (googleapi.jar) to access their services easily.

There are exercises for both using the generated proxy and the googleapi.

Exercise #4

- Preparations 2: Tomcat installation

- tomcat.apache.org
- actual version 5.5.17
- for miscellaneous os, incl. Windows
- Axis works as a servlet within Tomcat
- copy Axis webapps content to the Tomcat webapps directory

Exercise #5

- Creating an own Web Service using Apache Tomcat and Axis

Write a Java class „Calculator“ which allows you to add two Integer numbers.

Name it Calculator.jws and place it in TOMCAT_HOME/webapps/axis.

Call your Web service in a browser.

Exercise #6

- Test your own Web Service

Create an Eclipse project, fetch the WSDL document of your Calculator Web service, generate a proxy and write a Java client like in exercise #2.

Exercise #7

- Monitor HTTP request and response using Axis **tcpmon**

Open a DOS box (Start -> Ausführen -> cmd).

Execute (everything in one line!):

javaw -classpath <path to axis.jar>

org.apache.axis.utils.tcpmon 5555 localhost 8080

Start your Web service and change the port from 8080 to 5555 in the locator class!