Web Services

Dr. Ingo Melzer
DaimlerChrysler AG

18. Januar 2004

Syllabus

- Die Vorlesung findet im Wesentlichen alle zwei Wochen statt.
- Übungen werden in die Vorlesung integriert.
- Inhalt:
  - XML
  - SOAP
  - WSDL
  - UDDI
  - Sicherheit bei Web Services
  - Performanz
  - Evtl.: BPEL4WS, Transactions, Grid, . . .
- Die meisten Beispiele werden in Perl geschrieben sein. Grundlegende Kenntnisse sollten allerdings reichen. Etwas Java ist hilfreich.

Literatur

- Dieses Skript (wird veröffentlicht).
- Web Services with Perl, R. Ray und P. Kulchenko, O'Reilly
- Java Web Services, D. Chappell und T. Jewell, O'Reilly
- Web Services Essentials, E. Cerami, O'Reilly
- Mehr wird noch kommen ...

Service Oriented Architecture

- Idee der SOA: Dienste werden in einem Netz angeboten
- Diese können von anderen Diensten genutzt werden.
- Solche Dienste werden veröffentlicht und in Verzeichnissen aufgenommen.
- Andere Dienste können diese dann finden und bekommen Informationen, wie man den gefundenen Dienst nutzen kann.
- SOA wird oft im Zusammenhang mit EAI unter Interoperabitlität genannt. Dabei geht es um Integration von alten und neuen Systemen.
- Beispiel: Buche einen Flug, reserviere einen Sitzplatz, bezahle den Flug, ...
- Beispiel: Automatisches Lager, das automatisch nachbestellt und dabei den billigsten Anbieter sucht.
- Beispiel: Kühlschrank mit Barcode.
- Ein möglicher Ansatz zur Umsetzung einer SOA sind Web Services.

Historie

- Bereits einige Ansätze in diese Richtung: RPC, RMI, DCOM, CORBA, ...
- Nachteil: Abhängigkeiten von Betriebssystem oder Sprache.
- Verwendung oft sehr komplex.
- Softwaresysteme bestehen mehr und mehr aus kleinen Komponenten, die über immer mehr Rechner verteilt sind.
- Fast keine Integration von externen Systemen bei anderen Institutionen (Zuliefererintegration).

Probleme

- Wie finde ich die Funktionalität, die ich gerade brauche?
- Wie nutze ich ein Modul, wenn ich es gefunden habe?
- Brauche ich bestimmte Soft- oder Hardware, um das Angebot zu nutzen?
- Wie kann ich sicherstellen, dass andere meinen Dienst überhaupt finden können?

Entsprechende Funktionalitäten werden heute meist als *Dienste* oder *Services* bezeichnet.

Rollen bei einer SOA

1. Dienstanbieter:
   Implementiert den Dienst, publiziert ihn auf einem meist über das Internet zugänglichen Server, dokumentiert die Aufrufschnittstelle (das Interface) in einer computerlesbaren Form und ist verantwortlich für die Bereitstellung (hosting).

2. Dienstverzeichnis:
   Hier werden die von den Anbietern veröffentlichten Schnittstellen in einer katalogisierten Form verwaltet und angeboten. Ein solches Verzeichnis kann mit den gelben Seiten verglichen werden.

3. Dienstnutzer:
   Er nutzt den angebotenen Dienst, nachdem dieser meist über einen Verzeichnisdienst gefunden worden ist.

Ablauf

In einer SOA laufen meist folgende Schritte ab:

1. Ein Anbieter (Service Provider) implementiert einen Dienst.

2. Der Dienst wird inklusive der Beschreibung veröffentlicht (deployment).

3. Der Dienst wird bei einem Verzeichnisdienst (Service Broker) registriert.

4. Ein Nutzer sucht im Verzeichnis nach einem Diest und erhält eine Liste möglicher Kandidaten inkl. Schnittstellenbeschreibung.

5. Der Nutzer ruft einen Diest mit Hilfe der erhaltenen Informationen auf.

Sämtliche Schritte laufen mit standardisierten Protokollen und einem einheitlichen Datenformat ohne menschliche Interaktion ab.

Web Services Übersicht

- Web Services sind der zur Zeit aussichreichste Versuch, eine Service Oriented Architecture umzusetzen.
- Web Services können als technische Realisierung einer SOA gesehen werden.
- Allerdings erging es den Web Services wie den Portalen, sie wurden schnell berühmt, tauchten im Gartner Hype Cycle auf, und jeder versteht darunter etwas anderes.
- Sehr viele verschiedene Definitionen, je nach Anbieter.

Was sind Web Services?

Verschiedene Zitate:

- Nichts wirklich neues.
- Aber ein neuer Ansatz, um die Fehler der Vorgänger zu mindern.
- Sie basieren auf modernen und offenen Standards.
- Eignen sich nur für eine Computer-Computer Kommunikation.
- Benötigen ein Transport-Protokoll.
- Sind unabhängig von der Programmiersprache und dem Betriebssystem.

Entwicklung

- Email: Mensch-Mensch, es wird uninterpretierter Text verschickt.
- WWW: Mensch-Computer, ein Mensch konsumiert und gibt an, was er als nächstes lesen möchte. Der Computer bleibt passiv.
- Web Servics: Computer-Computer, ein Computer erbringt für einen anderen eine Dienstleistung.
- Semantic Web: Die Vision? Die beteiligten Computer verstehen, was ihnen angeboten wird. Die Dieste bieten semantische Information, die vom Nutzer verstanden werden können.

Net Services wäre eigentlich ein viel besserer Begriff gewesen!

XML

- Die eXtensible Markup Language ist ein Ableger von SGML.
- XML ist eine Metasprache.
- XML ist die Grundlage von Web Services.
- XML ist eine Möglichkeit, strukturierte Dokumente zu erzeugen.
- XML ist mächtig und trotzdem sehr einfach.
- Eine der Ideen von XML ist die Trennung von Daten und Semantik.
- XML ist ein W3C Standard. Die Verwendung ist daher frei.
- Es gibt viele Erweiterung wie zum Beispiel XML-Signature.

SOAP

- SOAP 1.0 wurde im Herbst 1999 von DevelopMentor, IBM, Microsoft, Lotus und UserLand Software veröffentlicht.
- Version 1.1 wuerde Anfang 2001 als W3C Note eingereicht.
- Die Idee, entfernte Funktionsaufrufe mit einem XML Vokabular zu ermöglichen, wurde von XML-RPC üebernommen.
- Damals stand SOAP für „Simple Object Access Protocol“. SOAP ist aber weder einfach noch erlaubt es einen echten Zugriff auf Objekte; daher steht es auch nur noch für SOAP.
- Die aktuelle Version, SOAP 1.2, ist eine W3C Recommendation.
- SOAP unterstützt das Versenden von synchronen und asynchronen Nachrichten an Dienste.
- SOAP ist unabhängig vom Transportprotokoll.

WSDL I

- Web Services Description Language
- Ein XML Vokabular zum Beschreiben von Schnittstellen
- Bei Web Services ist eine maschinenlesbare Schnittstellenbeschreibung nicht erzungen, aber doch empfehlenswert.
- WSDL helfen Tools zur Codeerzeugung.
- Entsprechende Beschreibungen dienen auch als Doku!

WSDL II

Die wichtigsten Elemente eines WSDL Dokuments:

- Interface aller exportierter Funktionen
- Definition der Ein- und Ausgabedatentypen
- Binding Information (also wie rufe ich den Dienst auf)
- Adressangaben (also wie genau finde ich den Dienst)

UDDI

- Universal (Service) Description, Discovery, and Integration.
- WSDL bietet eine rein technische Beschreibung der Schnittstelle.
- UDDI ist ein SOAP basierter Dienst, der dies erweitern soll.
- UDDI kann mit gelben Seiten verglichen werden.
- Meta-Daten zu den Diensten (z. B. um was für einen Dienst handelt es sich?) können erfasst werden.
- Suchmöglichkeiten nach Diensten werden spezifiziert.

Definition 1 Web Services

Die Web Services Architecture Arbeitsgruppe vom W3C definiert Web Services als „ein durch eine URI (nach RFC2396) identifiziertes Softwaresystem, dessen öffentliche Schnittstellen und Protokollbindungen durch XML definiert und beschrieben sind."

Definition 2 Web Services

Bei DaimlerChrysler wird meist folgende Definition verwendet:

„A Web Service is a piece of server-side software that provides a certain functionality (as a black box) and is accessible through Internet protocols using XML/SOAP messages with a described and published interface (typically by means of WSDL).

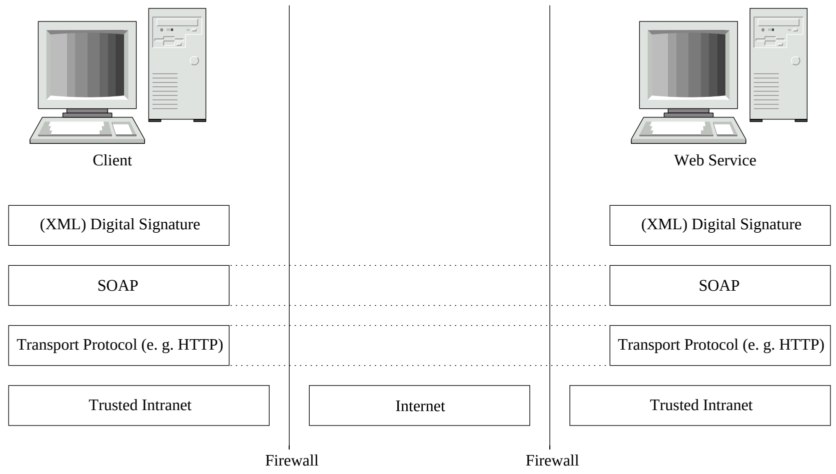
Those interface descriptions should be registered in a (global) registry such as UDDI."

Eigenschaften von Web Services

Web Services . . .

- ermöglichen die Kommunikation von verschiedenen Systemen.
- erleichtern die Integration von existierenden Systemen.
- agieren als Fassade für eine Menge von vergleichbaren Systemen.
- kommen in zwei Stilrichtungen vor: Dokument und RPC (depreciated).
- verwenden (nur) XML Protokolle (SOAP, . . .).
- verwenden zur Zeit meist HTTP(S) als Transport-Protokoll.
- sind eigentlich nur für eine Computer-Computer Kommunikation geeignet.
- ermöglichen leichteren Zugriff auf sonst verborgene Systeme (Security?).

Typischer Web Service Aufruf



- Ein Client nutzt einen Web Service.

Libraries für Web Services

- Apache AXIS
- Perls SOAP::Lite
- .NET Framework
- Systinet
- IBM: ETTK
- Sun: WSDP
- Cape Clear

XML

- Dieses Thema wurde am 11. November 2003 von Prof. Mario Jeckle präsentiert.
- Unter `http://www.jeckle.de/vorlesung/xml/script.html` findet man sein Skript zu seiner XML Vorlesung.
- Behandelt wurde im Wesentlichen Kapitel 1.
- Für diese Vorlesung sind wichtig: XML, Information Set (oft auch nur InfoSet), Namensräume, Schema, XPath, Signature und Encryption.

XML Information Set

- Der XML Information Set ist eine mögliche Darstellung von XML Dokumenten.
- Jedes XML Dokument entspricht genau ein Information Set.
- Die vollständige Information des Dokuments wird als Baum dargestellt.
- Die Wurzel des Baums ist das Document Information Item. Diesen gibt es daher auch genau einmal.
- Vorteil: gut zum Verständnis und bei einer weitern Darstellung muss nicht umgedacht werden (z. B. binär statt spitzen Klammern).

XML Namensräume

- Eigene Spezifikation (Recommendation): Namespaces in XML
- Die Recommendation wurde ein Jahr nach der ersten XML Version veröffentlicht
- Mögliche Lösung für das Problem der identischen Bezeichner
- Vergleichbar mit identischen Variablennamen
- Problem bei Wiederverwendung von XML-Strukturen
- Idee: Eine URI als Präfix (URLs sind echte Teilmenge der URIs)

XML Schema

- XML Schema Description Language (XSD) vermutlich wichtigeste Sprache zur Formulierung von bel. XML Strukturen
- Bildet zusammen mit XML 1.0 (2nd Edition) und den Namensräumen die Basis für zukünftige Entwicklungen um XML
- Ein XML Schema ist ein XML Dokument, das die Struktur von XML Dokumenten beschreibt.
- Ein XML Dokument heißt daher gültig hinsichtlich eines Schemas, wenn es über ein solches verfügt und einen zu diesem konfromen Aufbau hat.

XML XPath

- XPath erlaubt es, den „Ort“ eines Elements in einem XML Dokument anzugeben
- Damit kann ein Teil des Dokuments lokalisiert oder extrahiert werden
- XPath kann dadurch für lesende Anfragen genutzt werden
- XPath wird oft in Verbindung mit XSLT verwendet

Ein XML Beispiel

```
<addresses>
   <person>
      <name>
         <first>Ingo</first>
         <last>Melzer</last>
      </name>
      <city>Laupheim</city>
      <country>Germany</country>
   </person>
   <person>
      <name>
         <first>Andreas F.</first>
         <last>Borchert</last>
      </name>
      <city>Lonsee</city>
      <country>Germany</country>
   </person>
</addresses>
```

- Dieses einfache Beispiel zeigt bereits die hierarchische Struktur von XML.

Perl und XML

- Der Umgang mit XML in den meisten Sprachen ist relativ einfach, da es umfangreiche Bibliotheken gibt.
- Wie einfach der Umgang sein kann, soll hier mit Perl und dem Modul `XML::Simple` gezeigt werden.
- Der Import von `XML::Simple` mit dem Befehl `use` flutet den Namensraum der Programms mit den Routinen `XMLin` und `XMLout`.
- Wie der Name „Simple“ schon nahe legt, ist das Modul zwar einfach, hat aber auch seine Einschränkungen. Da die Adressen zum Beispiel als Hash abgelegt wurden, ist die Reihenfolge verloren.
- Das Ziel von `XML::Simple` ist die einfache Konvertierung von XML in Perl Datenstrukturen und zurück. Dadurch entstehen einige Einschränkungen.

XML::Simple

```
#!/usr/local/bin/perl

use strict;
use warnings;
use XML::Simple;
use Data::Dumper;

my $file = XMLin("adr.xml", forcearray => 0);
print Dumper($file);
```

```
$addresses = {
   'person' => [ {
      'city' => 'Laupheim',
      'country' => 'Germany',
      'name' => {
         'first' => 'Ingo',
         'last' => 'Melzer'
      }
   }, {
      'city' => 'Lonsee',
      'country' => 'Germany',
      'name' => {
         'first' => 'Andreas F.',
         'last' => 'Borchert'
      }
   } ]
};
```

Ein Perl Programm und die Ausgabe.

XML::Simple

- Die erste Zeile gibt der Shell an, wo die lokale Perl-Installation zu finden ist.
- Mit `use strict` werden einige Möglichkeiten von Perl, die oft zu Fehlern führen, verboten.
- `use warnings` erzeugt Meldungen, die meist durch einen Programmierfehler entstehen (zum Beispiel Verwendung von Variablen, die noch keinen definierten Wert haben). In früheren Perl Versionen war dafür in Zeile 1 die Option `-w` notwendig.
- Zeile 5 und 6 laden zwei Module.
- In Zeile 8 wird die XML Datei geladen und in eine Perl Datenstruktur umgewandelt.
- In der letzten Zeile wird diese Struktur einfach auf STDOUT ausgegeben.
- Die Ausgabe wurde etwas komprimiert (Leerzeichen gelöscht).

SOAP

- Das XML-Vokabular SOAP stellt die Basis von Web Services dar.

- SOAP ist ein vollständiges Protokoll; es sieht nicht nur Nutzdaten sondern auch Verwaltungsinformationen (wie zum Beispiel für das Routing) vor.

- SOAP benötigt ein Transport-Protokoll. Dies ist meist HTTP. Je nach Anwendung bieten sich unter anderem auch SMTP (asynchron), FTP (umfangreiche Nutzdaten), TCP (Performanz), BEEP, . . . , an.

- Die aktuelle Version, SOAP 1.2, ist eine W3C Recommendation.

- Siehe `http://www.w3.org/TR/soap12-part0/` sowie Teile 1 und 2.

Ein einfaches Beispiel

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:hello xmlns:n="http://example.org/hello">
      <n:expires>2003-10-28T18:00:00+01:00</n:expires>
      <n:someint>1</n:someint>
    </n:hello>
  </env:Header>
  <env:Body>
    <my:message xmlns:my="http://www.mycompany.com/myns">
      <my:text>Hello World!</my:text>
    </my:message>
  </env:Body>
</env:Envelope>
```

- Diese einfache SOAP Nachricht zeigt bereits deutlich die wesentliche Struktur solcher Dokumente.

Struktur von SOAP Nachrichten

- Jede SOAP Nachricht besitzt (unabhängig von synchron bzw. asynchron und Anfrage bzw. Antwort)
  - Einen Envelope mit dem Namensraum, bei SOAP 1.2 ist dies `http://www.w3.org/2003/05/soap-envelope` Dort gibt es auch das Schema.
  - Einen optionalen Header mit Verwaltungsinformationen und evtl. benutzerdefinierten Feldern.
  - Einen Body mit mindestens einem Element. Der Body enthält die gesammten Nutzdaten.

Der SOAP Header

Der SOAP Header enthält Verwaltungsinformationen und Metadaten. Für die Header Elemente gilt:

- Sie enthalten keine Nutzdaten
- Sie sind oft nicht nur für den (entgültigen) Empfänger
- Müssen von SOAP-Knoten nicht verarbeitet werden (außer `mustUnderstand == true` siehe später)
- Typische Header Elemente sind Routinginformationen, Transaktionsdaten, Zugriffsdaten wie Name und Passwort (sollte aber verschlüsselt sein), . . .

Der SOAP Header

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:ext xmlns:n="http://example.org/ext"
        env:mustUnderstand="true">
      <n:someint>1</n:someint>
    </n:ext>
    <m:ext2 xmlns:m="http://example.org/ext2"
env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
        env:mustUnderstand="true">
      <m:someint>1</m:someint>
    </m:ext2>
  </env:Header>
  <env:Body>
    <my:message xmlns:my="http://www.mycompany.com/myns">
      <my:text>Hello World!</my:text>
    </my:message>
  </env:Body>
</env:Envelope>
```

- Ein wichtiges Attribut von Kopfelementen ist `mustUnderstand`.

Der SOAP Header: mustUnderstand

- Die Beachtung und Verarbeitung von Header-Element ist optional, ausser das Attribut `mustUnderstand` ist logisch wahr, also `true` oder `1`.
- Kann ein SOAP-Knoten ein Header-Element in einem solchen Fall nicht bearbeiten, so muss dieser einen SOAP-Fault erzeugen und an den Sender zurückschicken.
- In diesem Fall ist ein SOAP-Knoten jeder Rechner vom Sender zum Empfänger, der SOAP versteht.
- Einschränkungen sind durch das Attribut `role` möglich. Im gegebenen Beispiel muss das Element `ext2` nur vom Knoten `http://example.org/ext2` verstanden werden. Dieser Knoten wird nach Bearbeitung das Element aus der Nachricht löschen.
- Häufige Rollen sind auch `ultimateReceiver` und `none`.

Der SOAP Body

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:env="http://www.w3.org/2003/05/soap-envelope"
   xmlns:b="http://www.example.org/AddService>
   <env:Body>
      <b:add env:encodingStyle="http://www.w3.org/2003/05/soap-encoding
      <b:a>2</b:a>
      <b:b>3</b:b>
   </b:add>
</env:Body>
</env:Envelope>
```

- Jede SOAP Nachricht muss einen Body mit mindestens einem Element haben.
- Dieses Element muss aus einem anderen Namensraum sein. Dies ist für eine eventuelle Überprüfung des Schemas notwendig.

Die SOAP Antwort

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
   xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
   xmlns:b="http://www.example.org/AddService">
   <env:Body>
      <b:result env:encodingStyle="http://www.w3.org/2003/05/soap-encoc
      5</b:result>
   </env:Body>
</env:Envelope>
```

- Die Struktur der Antwort ist identisch zu der des Aufrufs.

SOAP Fault

- Entsteht bei einem Web Services Aufruf ein Fehler, so muss ein sog. Soap-Fault an den Sender geschickt werden.
- Der `body` eines Faults besteht nur aus einem `fault` Element.
- Dieses `fault` Element muss ein `Code` und ein `Reason` Element enthalten. Weitere Elemente wie `Detail` sind optional.
- HTTP Fehlermeldungen wie 403 oder ein redirect sind nicht zulässig.

SOAP Fault

```
<env:Envelope xmlns:env='http://www.w3.org/2003/05/soap-envelope'>
   <env:Header>
      <a:Extension1 xmlns:a='http://example.org/ext'
         env:mustUnderstand='true' />
      <b:Extension2 xmlns:b='http://example.com/stuff'
         env:mustUnderstand='true' />
   </env:Header>
   <env:Body>
      <!--- Some Text --->
   </env:Body>
</env:Envelope>
```

- Bei obigem Aufruf muss der Empfänger beide Elemente des Headers verstehen.

SOAP Fault

```
<env:Envelope xmlns:env='http://www.w3.org/2003/05/soap-envelope'
      xmlns:xml='http://www.w3.org/XML/1998/namespace' >
   <env:Header>
      <env:NotUnderstood qname='a:Extension1'
         xmlns:a='http://example.org/ext' />
      <env:NotUnderstood qname='b:Extension2'
         xmlns:b='http://example.com/stuff' />
   </env:Header>
   <env:Body>
      <env:Fault>
      <env:Code><env:Value>env:MustUnderstand</env:Value></env:Code>
         <env:Reason>
            <env:Text xml:lang='en'>One or more mandatory
               SOAP header blocks not understood
            </env:Text>
            <env:Text xml:lang='de'>Mindestens ein Pflichtfeld
               des SOAP-Headers wurde nicht verstanden.
            </env:Text>
         </env:Reason>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

- Werden die Headerelemente nicht verstanden, so muss ein entsprechender Fault erzeugt werden.

SOAP Fault

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
   xmlns:m="http://www.example.org/timeouts"
   xmlns:xml="http://www.w3.org/XML/1998/namespace">
   <env:Body>
      <env:Fault>
         <env:Code>
            <env:Value>env:Sender</env:Value>
            <env:Subcode>
               <env:Value>m:MessageTimeout</env:Value>
            </env:Subcode>
         </env:Code>
         <env:Reason>
            <env:Text xml:lang="en">Sender Timeout</env:Text>
         </env:Reason>
         <env:Detail>
            <m:MaxTime>P5M</m:MaxTime>
         </env:Detail>
      </env:Fault>
   </env:Body>
</env:Envelope>
```

- So könnte die Fehlermeldung für einen Timeout aussehen.

Web Services Beispiele

- Ein paar erste praktische Beispiele zum Ausprobieren.
- Web Server: Apache
- Perl: SOAP:Lite
- Java: Apache Tomcat und Axis

Hello World Client in Perl

hi.pl

```
#!/usr/local/bin/perl -w

use strict;
use SOAP::Lite;

print SOAP::Lite
   -> uri('http://localhost/Demo')
   -> proxy('http://localhost/perl/hello.pl')
   -> hi()
   -> result, "\n";

print SOAP::Lite
   -> uri('http://localhost/Demo')
   -> proxy('http://localhost/perl/hello.pl')
   -> echo("Hallo Welt\n")
   -> result;
```

- Ein Aufruf eines Web Services in Perl.
- proxy ist die Adresse, die auf dem Server angesprochen wird.
- Bei proxy wird auch das Protokoll bestimmt (http, mailto, ftp, . . .).
- uri gibt an, was man ueber den proxy ansprechen möchte. Es handelt sich nur um einen Namen für den Dienst.

Echo World Client in Java

echo.java

```
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import javax.xml.namespace.QName;

public class echo {
   public static void main(String [] args) {
       try {
           String endpoint = "http://localhost/perl/hello.pl";
           Service  service = new Service();
           Call     call    = (Call)service.createCall();
           call.setTargetEndpointAddress( new java.net.URL(endpoint) );
           call.setOperationName(new QName("http://localhost/Demo", "ec
           String ret = (String) call.invoke(new Object[] { "Hello!" })
           System.out.println("Sent 'Hello!', got '" + ret + "'");
       } catch (Exception e) {
           System.err.println(e.toString());
       }
   }
}
```

- Ein Aufruf eines Web Services in Java.

Hello World Server in Perl I

hello.pl

```
#!/usr/local/bin/perl -w

use strict;
use SOAP::Transport::HTTP;
use Demo;

SOAP::Transport::HTTP::CGI
   -> dispatch_to('Demo')
   -> handle;
```

- Der Aufruf wird über das Common Gateway Interface, CGI, an den Web Service geschickt.
- `SOAP::Transport::HTTP` ermöglicht, dass keine HTTP und CGI Details beachtet werden müssen.
- `dispatch_to` gibt an, was zur Verfügung gestellt werden soll. Wird ein Modul angegeben, so werden alle Methoden angeboten.
- Mehr siehe `perldoc SOAP::Transport::HTTP`

Hello World Server in Perl II

Demo.pm

```
package Demo;

sub echo {
   return @_[1];
}

sub hi {
   return "hello, world";
}

sub bye {
   return "goodbye, cruel world";
}

1;
```

- Die eigentliche Funktionalität in einem eigenen Modul.

Ein Taschenrechner in Java - Server

Calc.jws

```
public class Calc {
   public int add(int a, int b) {
      return a + b;
   }
}
```

- Bei Axis gibt es zwei Möglichkeiten, Dienste anzubieten.
- Als „normale“ Java-Klasse im Quelltext, diese wird mit der Endung `jws` meist unter `$TOMCAT/webapps/axis` abgelegt. Die Übersetzung findet transparent beim ersten Aufruf statt.
- Die zweite Möglichkeit ist ein *Web Services Deployment*. Hier werden übersetzte Java-Klassen veröffentlicht. Über einen entsprechenden Deskriptor kann (oder muss) man die genaue Art der Veröffentlichung spezifizieren.

Ein Taschenrechner in Java - Client

CallAdd.java

```
import java.rmi.RemoteException;
import javax.xml.rpc.ParameterMode;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;

public class CallAdd {
   public static void main(String[] args) {
      Call call = new Call(new Service());
      call.setTargetEndpointAddress("http://localhost/axis/Calc.jws");
      call.setOperationName("add");
      call.addParameter("a", XMLType.XSD_INT, ParameterMode.IN);
      call.addParameter("b", XMLType.XSD_INT, ParameterMode.IN);
      call.setReturnType(XMLType.XSD_INT);
      Object[] param = new Object[2];
      param[0] = new Integer(args[0]);
      param[1] = new Integer(args[1]);
      try {
         Object result = call.invoke(param);
         System.out.println("result="+result);
      } catch (RemoteException re) {
         re.printStackTrace();
      }
   }
}
```

- Der Client ist vergleichbar mit dem letzten Beispiel - es wurden ein paar Fehlermeldungen und die Datentypen eingefügt.

Ein Taschenrechner in Perl - Client

CallAdd.pl

```
#!/usr/local/bin/perl -w

use strict;
use SOAP::Lite;

print "result=",
   SOAP::Lite->uri("http://localhost/Calulator")
   ->proxy("http://localhost/axis/Calc.jws")
   ->add(@ARGV[0, 1])
   ->result, "\n";
```

- Das Gleiche nochmals in Perl (ohne Fehlermeldungen und Datentypen).

Ein Taschenrechner in Perl - Client mit WSDL

CallAddWSDL.pl

```
#!/usr/local/bin/perl -w

use strict;
use SOAP::Lite;

print "result=",
   SOAP::Lite->uri("http://localhost/Calulator")
   ->proxy("http://localhost/axis/Calc.jws")
   ->add(@ARGV[0, 1])
   ->result, "\n";

my $service = SOAP::Lite->service('http://localhost/axis/Calc.jws?wsdl'
print "result=", $service->add(@ARGV[0, 1]), "\n";
```

- Das Gleiche nochmals — aber mit WSDL

WSDL

- Im Gegensatz zu vergleichbaren Architekturen wie CORBA ist bei Web Services eine Schnittstellendefinition nicht zwingend erforderlich.
- Trotzdem ist diese sehr zu empfehlen. Diese Aufgabe uebernimmt die Web Services Description Language.
- WSDL entält alle wichtigen Informationen zum Aufruf eines Dienstes.
- Wenn man die Struktur des Dienstes kennt, wird aus Performanzgründen WSDL oft weggelassen.

WSDL Aufbau

- Root-Element: `<definitions>`
- Kind-Elemente bei WSDL 1.1 sind (WSDL 2.0):

  - `<types>` enthält die verwendeten Datentypen
  - `<message>` gibt an, welcher Nachrichteninhalt üebertragen werden kann.
  - `<portType>` (`interface`) Signaturen der Methoden
  - `<binding>` liefert Details zur Implementierung inklusive der Protokolle zum Aufruf.
  - `<port>` (`endpoint`, Teil von `service`)
  - `<service>` enthält eine Sammlung von einem oder mehreren Ports: wo üeberall finde ich die Services.

WSDL Beispiel

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   targetNamespace="http://localhost/axis/Calc.jws"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:impl="http://localhost/axis/Calc.jws"
   xmlns:intf="http://localhost/axis/Calc.jws"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <!-- portType -->
   <!-- message -->
   <!-- binding -->
   <!-- service -->
</wsdl:definitions>
```

- Das Root-Element mit den benötigten Namensräumen.

WSDL Beispiel

```
<!-- definitions -->
<wsdl:portType name="Calc">
   <wsdl:operation name="add" parameterOrder="a b">
      <wsdl:input message="impl:addRequest" name="addRequest"/>
      <wsdl:output message="impl:addResponse" name="addResponse"/>
   </wsdl:operation>
</wsdl:portType>
<!-- message -->
<!-- binding -->
<!-- service -->
```

- Die Signatur der Methode `add`.

WSDL Beispiel

```
<!-- definitions -->
<!-- portType -->
<wsdl:message name="addRequest">
   <wsdl:part name="a" type="xsd:int"/>
   <wsdl:part name="b" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="addResponse">
   <wsdl:part name="addReturn" type="xsd:int"/>
</wsdl:message>
<!-- binding -->
<!-- service -->
```

- Ein- und Ausgabeparameter sind also alles Integer.

WSDL Beispiel

```
<!-- definitions -->
<!-- portType -->
<!-- message -->
<wsdl:binding name="CalcSoapBinding" type="impl:Calc">
   <wsdlsoap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="add">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="addRequest">
        <wsdlsoap:body
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
           namespace="http://DefaultNamespace" use="encoded"/>
      </wsdl:input>
      <wsdl:output name="addResponse">
         <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http://localhost/axis/Calc.jws" use="encoded"/>
      </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
<!-- service -->
```

- Den Dienst spricht man im RPC-Style über HTTP an.
- Es gibt eine Methode `add`.
- Ein- und Ausgabe sind auch referenziert.

WSDL Beispiel

```
<!-- definitions -->
<!-- portType -->
<!-- message -->
<!-- binding -->
<wsdl:service name="CalcService">
   <wsdl:port binding="impl:CalcSoapBinding" name="Calc">
      <wsdlsoap:address location="http://localhost/axis/Calc.jws"/>
   </wsdl:port>
</wsdl:service>
```

- Der Dienst läuft also lokal als `axis.jws`.

UDDI

- UDDI, oder Universal Description, Discovery, and Integration, ist eine Umsetzung der gelben Seiten auf Web Services.
- UDDI wurde seit Herbst 2000 von Ariba, IBM und Microsoft entwickelt.
- UDDI ist ein Register für Dienste inklusive Beschreibung und Suchmethoden.
- Wichtige Daten sind Kontaktinformationen, Infos üeber die Dienste, kategorisierte Listen und Protokoll-Informationen.
- UDDI ist ein Netzwerk von (UDDI)-Knoten, die über SOAP Informationen austauschen. Dies erlaubt verteiltes veröffentlichen und (hoffentlich) finden.

Struktur

- Ein UDDI Verzeichnis hat einen Baum als interne Datenstruktur, da mal wieder XML verwendet wurde.
- In dieser Struktur gibt es vier wesentliche Elemente:
  - Firmen (Business-Entity)
  - Dienste (Business-Service)
  - Bindungen (Binding-Template)
  - Technisches Modelle (TModel)
- Die ersten drei bilden die oberste Einteilung
- TModels bieten erlauben Referenzen auf externe Informationen.

Business Entity

```
<businessEntity businessKey="uuid:F1I2R3M4A-5K6E7Y" ...>
   <name>Ich AG</name>
   <description>Wir entwickeln Web Services</description>
   <contacts>
      <contact useType="general info">
 <description>Eine Beschreibung</description>
 <personName>Ingo Melzer</personName>
 <phone>0170 1234567</phone>
      </contact>
   </contacts>
   <businessServices>
      <!-- Hier stehen meine Web Services -->
   </businessServices>
   <categoryBag/> <!-- Kategorien wie DUNS oder NAICS -->
</businessEntity>
```

- Die Business Entities beschreiben die Anbieter von Diensten.
- Enthalten sind Daten über Branche, Kontakt, Kategrorien und Listen.
- uuid steht für Universal Unique ID und wird von der jeweiligen Registry vergeben.

Business Service

```
<businessService serviceKey="uuid:S1E2R3V-I4C5E6-K5E8Y"
   businessKey="uuid:F1I2R3M4A-5K6E7Y">
   <name>Mein Web Service</name>
   <description>Noch mehr Text</description>
   <bindingTemplates>
      <!-- Das kommt auf der naechsten Folie -->
   </bindingTemplates>
   <categoryBag/>
</businessService>
```

- Business Service beschreibt einzelne Web Services.
- Jeder Business Service wird von einem Business Entity angeboten.
- Informationen zum Binding, Art und Kategorie können auch angegeben werden.

Binding-Template

```
<bindingTemplate serviceKey="uuid:S1E2R3V-I4C5E6-K5E8Y"
      bindingKey="uuid:B1I2N3D4I-N6G7-8K9E0Y">
   <description>Web Service SOAP-Bindung</description>
   <accessPoint URLType="http">
      http://www.woistderservice.de/
   </accessPoint>
   <TModelInstanceDetails>
      <!-- Verweis auf TModel-Struktur, kommt gleich -->
   </TModelInstanceDetails>
</businessService>
```

- Jedes Binding-Template eine technische Beschreibung eines Web Services. Dieser muss als Business Service existieren.
- Jeder Web Service kann bel. viele Bindungen mit evtl. verschiedenen Protokollen haben.
- BT verbindet alle tModels eines Dienstes.
- Binding-Template entspricht in etwa dem WSDL Element `service`.

tModel

```
<tModel
   tModelKey="uuid:0e727db0-3e14-11d5-98bf-001234567c89"
   operator="www.example.com/services/uddi"
   authorizedName="0100001QS1">
   <name>Mein einfacher Web Service</name>
   <description xml:lang="en">My simple Web Service</description>
   <overviewDoc>
   <description xml:lang="en">wsdl link</description>
      <overviewURL>
         http://www.example.net/tmodels/SimpleWS.wsdl
      </overviewURL>
   </overviewDoc>
   <categoryBag>
      <keyedReference
         tModelKey="uuid:c1acf23d-1234-4567-8d90-39b756e62ab4"
         keyName="uddi-org:types" keyValue="wsdlSpec" />
   </categoryBag>
</tModel>
```

- tModels nehmen bel. Beschreibungen von Diensten auf. Meist sind dies Verweise auf WSDL-Beschreibungen.
- tModels können auch Zuordnungen zu Dienstfamilien oder andere Kategorisierungen vornehmen.

Sicherheit bei Web Services

- Die SOAP-Spezifikation enthält nichts zum Thema Sicherheit.
- Es gibt verschiedene Erweiterungen und Vorschläge, wie die Sicherheit von Web Services verbessert werden kann.
- Wichtige Punkte sind diverse XML-Erweiterungen wie zur Signatur und Verschlüsselung, WS-Security und SAML.
- Achtung: Ein solches Thema wird man nie an einer Stelle lösen können.

Struktur

- Bei einer abstrakten Betrachtung des Web Services Stacks gibt es min. drei Ebenen, die getrennt behandelt werden sollten:
  - SOAP (XML basiert)
  - Transport Protokoll (wie gesagt meist HTTP)
  - Ethernet (TCP/IP)

  Die unterste Schicht sollte eigentlich in mehrere unterteilt werden. Dies wird hier aber nicht getan, da Web Services sich hier nicht von anderen netzwerkbasierten Anwendungen unterscheiden.

Struktur

- Auf jeder Schickt kann die Sicherheit erhöht werden.
- In der Regel kann eine Schicht alleine einen Dienst nicht vollständig sichern.
- XML-Schicht ist besonders wichtig (Flexibilität, Intermediaries)
  - SOAP (XML basiert): XML-Security
  - Transport Protokoll (wie gesagt meist HTTP): SSL
  - Ethernet (TCP/IP): IPSec

SSL reicht oft nicht aus

- SSL ist nur für Punkt zu Punkt Verbindungen,
- ist nur für ein paar Protokolle geeignet (im Wesentlichen HTTP),
- nur der Transport ist verschlüsselt
- nur das ganze Dokument kann verschlüsselt werden,
- Header Informationen nicht mehr lesbar, diese enthalten aber wesentliche Informationen (Routing) und sind Voraussetzung für Intermediaries.
- Was passiert bei Aufrufen, die mehr als einen einzigen Service betreffen?
- Asynchrone Aufrufe sind nicht möglich
- Daten werden bei Ankunft auf der Server automatisch Entschlüsselt.
- Absendersignatur geht verlohren.

XML-Signature

5 Punkte zur digitalen Signatur:

- Bestimme Teile, die signiert werden sollen.
- Berechne Hash-Wert (Digest) der Teile (nach Kanonisierung).
- `<SignedInfo>`, enthält Digest, verwendete Algorithmen, XPATH
- Berechne Hash-Wert des `<SignedInfo>` und signiere diesen. Das Ergebnis bildet den `<SignatureValue>`
- Füge `<SignedInfo>`, `<SignatureValue>` und `<KeyInfo>` als `<Signature>` Element in das Dokument ein.

WS-Security

- WS-Security ist eine Initiative von IMB, Microsoft und Verisign.
- Version 1.0 wurde 2002 der OASIS zur Standardisierung übergeben.
- WS-Security gibt einen Rahmen vor, mit dem die wichtigsten Sicherheitsanforderungen umgesetzt werden können.
- Wichtige Punkte sind Namenskonventionen, zum Beispiel wie XML-Signature und XML-Encryption in SOAP-Nachrichten einzubetten sind.

Auf WS-Security aufbauende Standards

- *WS-Policy:* erlaubt das erstellen von Richtlinien
- *WS-Trust:* Vertrauensbeziehungen
- *WS-Privacy:* Individuelle Sicherheitseinstellungen
- *WS-Secure Conversation:* Richtlinien zum Sitzungsverwaltung
- *WS-Federation:* Verwaltung von Beziehungen zwischen Web Services.
- *WS-Authorization:* Form von Daten und Richtlinien zur Authentifizierung

Authentisierung

- Authentisierung ist die Bestimmung einer Identität.
- Dies ist notwendig, um Zugangsbeschränkungen zu implementieren.
- Beweise einer erfolgreichen Authentisierung sind für single sign-on notwendig.
- Authentisierung wird auch benötigt, wenn eine Sammlung von Funktionalitäten als eine einzige erscheinen soll (Portale, Web-Services von verschiedenen Anbietern).

SAML - Security Assertion Markup Language

- SAML definiert ein XML Vokabular, das einen Austausch von Sicherheits-Versicherungen erlaubt.
- Üblich sind Authentisierungs- und Authorisierungs-Bestätigungen.
- So ist es zum Beispiel möglich zu bestätigen, dass ein gültiges Passwort angegeben worden war.
- Mehr `http://www.oasis-open.org/committees/security/`

Ein Beispiel

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/06/soap-envelope">
  <env:Header>
    <ns1:DeliveryNotification env:mustUnderstand="true" env:role=
      "http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver"
      xmlns:ns1="urn:xmlns:daimlerchrysler.com:research">
        <ns1:SendTo URI="MailTo:john.doe@daimlerchrysler.com"/>
    </n1:DeliveryNotification>
    <wsse:Security
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
      <wsse:UsernameToken Id="MyID">
        <wsse:Username>John</wsse:Username>
      </wsse:UsernameToken>
      <ds:Signature>
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1"/>
          <ds:Reference URI="#MsgBody">
            <ds:DigestMethod
              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>LyLsF0Pi4wPU...</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#MyID"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </env:Header>
  <env:Body Id="MsgBody">
    <ns2:QuoteRequest>
      <ns2:ID>7492653</ns2:ID>
      <ns2:Amount>10000</ns2:Amount>
      <ns2:DeliverTo>DaimlerChrysler ...</ns2:DeliverTo>
      <!-- more details omitted for brevity ... -->
    </ns2:QuoteRequest>
  </env:Body>
</env:Envelope>
```