In diesem Kapitel
- XML als Message Format
  - *Umschläge*
  - *Daten-Repräsentation*
- http als Transport-Protokoll
  - *Wie funktioniert http?*
  - *http und Java*
- XML-RPC
  - *Datenstrukturen*
  - *Fehlerbehandlung*
  - *Validation*
- SOAP
  - *Ein Beispiel*
  - *SOAP Headers*
  - *SOAP Grenzen*
  - *Validation*

2. *XML Protokolle: XML-RPC und SOAP*

2.1. Einleitung

XML hat sich nicht nur zur Beschreibung der Daten und Dokumente durchgesetzt, auch in der Kommunikation lösen XML basierte Systeme ältere beispielsweise EDI/EDIFACT basierte Systeme ab.

Eine der ersten Anwendungen in diesem Bereich war XML-RPC (remote procedure call) von Userland. Später wurde daraus SOAP, unter Mithilfe von Microsoft und IBM. Diese beiden Firmen waren wesentlich an der Verbreitung der XML Technologien beteiligt, Sun schlief ihren Server Schlaf und träumte von Java Applikationen für Kaffee-Maschinen oder intelligente Fingerringe.

W3C nahm sich dem Thema an und gründete eine XML Protokoll Gruppe (siehe auch meine Unterlagen zum Thema Enterprise Computing aus dem Jahr 2000/2001ff).

XML Protokolle verwenden http oder SMTP als Basis-Protokolle, warum werden wir noch sehen (Anfrage-Antwort basierte Protokolle: SMTP mit automatischer Antwort [gelesen], http POST [Bestätigung]).

Zusätzlich werden wir uns mit RDF (Resource Description Framework) befassen, nicht zuletzt wegen meinem Interesse für den Semantic Web.

2.2. XML als Message Format

Der Vorteil von XML als Message Format besteht unter anderem darin, dass

1) XML auf den üblichen Plattformen zur Verfügung steht
2) Das Format damit einheitlich, also ohne lange Diskussionen über interne Darstellungen (Big und Little Endian), verwendet werden kann.
3) XML ist Programmiersprachen-unabhängig
4) XML ist Betriebsystem-unabhängig
5) XML Dokumente können über http, SMTP, PIM's ausgetauscht werden.

XML steht in vielen Varianten zur Verfügung, Spezialisierungen für bestimmte Themenbereiche oder Arbeitsgebiete:

- Finanzsektor : Open Financial Exchange (OFX): `www.ofx.net`.
- Bio-Sektor : von Rosetta (wie vieles mehr) : `http://www.rosettabio.com/products/conductor/geml/resources.htm` die Gene Expression Markup Language.
- Weitere XML Spezialisierungen für fast alle Wirtschaftsbereiche.
- News-Sektor: International Press Telecommunications Council `http://www.newsml.org/`
- ...

2.2.1. Umschläge - Envelopes

Wie im Briefverkehr hat es sich auch beim elektronischen Austausch als nützlich erwiesen, zwischen dem Inhalt einer Nachricht und den Informationen über Empfänger, usw. zu unterscheiden. Der Inhalt kann im einfachsten Fall, falls die Nachrichten nur zwischen zwei Partnern ausgetauscht werden, ohne Umschlag ausgetauscht werden. Falls mehrere unterschiedliche Sender und Empfänger involviert sind, bietet der Umschlag viele Vorteile.

Wie die Umschläge definiert werden, hängt vom Protokoll ab:

- XML-RPC : alle Markup Informationen sind im Umschlag; der Textinhalt wird durch die Daten im Umschlag repräsentiert
- SOAP : Inhalt und Umschlag sind XML- Dokumente.

2.2.2. Daten Repräsentation

In internationalen Firmen und im Web werden Informationen über Landes- und Kontinent-Grenzen ausgetauscht. Wie sieht in diesem Fall die Datumsausgabe aus? Ist 3/7/09 der 3. Juli 2003 oder der 7. März 3003?

All diese Format-Probleme werden in XML mithilfe eines XML- Schemas gelöst. XML Schema gilt als sehr komplex und der Standard als inkonsequent und schwer lesbar, was Sie selber leicht verifizieren können. Aber wenigstens werden verschiedene Basisdatentypen ein für alle Male festgelegt.

Primitive Datentypen nach W3C XML Schema

Typ	Bedeutung
`xsd:string`	Eine Unicode Zeichenkette (eine DTD #PCDATA).
`xsd:boolean`	true, false, 1, 0
`xsd:decimal`	Dezimalzahl 12.14345 oder -0.1234. Grösse und Länge sind beliebig; analog zur `java.math.BigDecimal` Klasse.
`xsd:float`	4-Byte IEEE-754 Fliesskommazahl; entspricht Java `float`
`xsd:double`	8-Byte IEEE-754 Fliesskommazahl; entspricht Java `double`
`xsd:integer`	Ganzzahlig, analog zur `java.math.BigInteger` Klasse
`xsd:nonPositiveInteger`	Ganze nicht positive Zahl (0, -1...)
`xsd:negativeInteger`	Ganze negative Zahl (-1,-2,...)
`xsd:nonNegativeInteger`	Ganze, nicht negative Zahl (0, 1,2,...)
`xsd:long`	Ganze Zahl zwischen -9223372036854775808 und +9223372036854775807 inklusive; entspricht Java `long`
`xsd:int`	Ganze Zahl zwischen -2147483648 und 2147483647 inklusive; entspricht Java `int`
`xsd:short`	Ganze Zahl zwischen -32768 und 32767 inklusive; entspricht Java `short`
`xsd:byte`	Ganze Zahl zwischen -128 und 127 inklusive; entspricht Java `byte`
`xsd:unsignedLong`	Ganze Zahl zwischen 0 und 18446744073709551615.
`xsd:unsignedInt`	Ganze Zahl zwischen 0 und 4294967295
`xsd:unsignedShort`	Ganze Zahl zwischen 0 und 65535
`xsd:unsignedByte`	Ganze Zahl zwischen 0 und 255
`xsd:positiveInteger`	Ganze Zahl grösser 0.
`xsd:duration`	Zeitdauer gemäss ISO 8601 extended format: `PnYnMnDTnHnMnS`. Sekunden: dezimal oder Integer; sonst Integer.
`xsd:dateTime`	Tageszeit gemäss ISO 8601 Format: *`CCYY-MM-DD`*`Thh:mm:ss`.
`xsd:time`	Uhrzeit gemäss ISO 8601 Format: *`hh:mm:ss.sss`*.
`xsd:date`	Datum gemäss ISO 8601 Format: *`YYYYMMDD`*;
`xsd:gYearMonth`	Monat und Jahr.
`xsd:gYear`	Jahr im Gregorianischen Kalender
`xsd:gMonthDay`	Monat im Gregorianischen Kalender
`xsd:gDay`	Tag ohne Monatsangabe
`xsd:gMonth`	Monatsangabe ohne Jahr
`xsd:hexBinary`	Hexa-Darstellung von Daten

Typ	Bedeutung
xsd:base64Binary	Base-64 codierte Daten
xsd:anyURI	Absolute oder relative URL oder URN
xsd:QName	"Qualified Name": hier ein ein Präfix, wie SOAP-ENV:Body.
xsd:NOTATION	Name einer NOTATION im aktuellen Schema.
xsd:normalizedString	Zeichenkette, bei der \r, \n und \t als Leerzeichen gesehen werden
xsd:token	Zeichenkette, welche mehrere Whitespaces als *ein* Leerzeichen betrachten.
xsd:language	RFC 1766 Sprach-Kennzeichen: en, fr-CA (Canada)
xsd:NMTOKEN	XML Name Token
xsd:NMTOKENS	Liste von durch Whitespace getrennten XML Name Tokens
xsd:Name	XML Name
xsd:NCName	XML Name ohne Präfix
xsd:ID	Eindeutiger Name
xsd:IDREF	Eindeutiger Name im XML Dokument
xsd:IDREFS	Liste von durch Whitespaces getrennten IDREF's
xsd:ENTITY	Ungeparste Entity in einer DTD
xsd:ENTITIES	Liste von durch Whitespace getrennten ENTITY Namen

Falls Sie diese Datentypen einsetzen wollen, müssen Sie entweder einen Namensraum definieren, oder den Standardnamensraum `http://www.w3.org/2001/XMLSchema-instance` verwenden, mithilfe des Präfix `xsi`.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <Order xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Customer id="c32" xsi:type="xsd:string">ACME Inc.</Customer>
  - <Product>
      <Name xsi:type="xsd:string">Kuckucks-Uhr</Name>
      <SKU xsi:type="xsd:string">1234</SKU>
      <Quantity xsi:type="xsd:positiveInteger">1</Quantity>
      <Price currency="EUR" xsi:type="xsd:decimal">50</Price>
    - <ShipTo>
        <Street xsi:type="xsd:string">Sonnenbergstrasse
          10</Street>
        <City xsi:type="xsd:string">Herrliberg</City>
        <State xsi:type="xsd:NMTOKEN">ZH</State>
        <Zip xsi:type="xsd:string">8765</Zip>
      </ShipTo>
    </Product>
  - <Product>
      <Name xsi:type="xsd:string">Solar Uhr</Name>
      <SKU xsi:type="xsd:string">998</SKU>
      <Quantity xsi:type="xsd:positiveInteger">100</Quantity>
      <Price currency="EUR" xsi:type="xsd:decimal">50</Price>
      <Discount xsi:type="xsd:decimal">.10</Discount>
    - <ShipTo>
        <GiftRecipient xsi:type="xsd:string">Samuel
          Schmid</GiftRecipient>
```

```
        <Street xsi:type="xsd:string">Bundesplatz 1</Street>
        <City xsi:type="xsd:string">Bern</City>
        <State xsi:type="xsd:NMTOKEN">BE</State>
        <Zip xsi:type="xsd:string">3000</Zip>
      </ShipTo>
      <GiftMessage xsi:type="xsd:string">Gratuliere zum
        Fernsehaufritt! Mach weiter so! Ueli und
        Christoph</GiftMessage>
    </Product>
    <Subtotal currency="EUR" xsi:type="xsd:decimal">4550</Subtotal>
    <Tax rate="7.0" currency="EUR" xsi:type="xsd:decimal">28</Tax>
    <Shipping method="Post" currency="EUR"
      xsi:type="xsd:decimal">12</Shipping>
    <Total currency="EUR" xsi:type="xsd:decimal">4590.00</Total>
  </Order>
```

2.3. *HTTP als universelles Transport Protokoll*

Daten und Dokumente kann man mit unterschiedlichsten Protokollen austauschen:

- SMTP (mail)
- FTP (File Transfer)
- Pipes
- RPC (remote procedure calls: Prozeduren, die von einem Client auf einem fernen Server ausgeführt werden)
- ...

Im Web hat sich das http Protokoll für Web Seiten und Datentransfer etabliert. Auch für viele nicht Web basierte Anwendungen bietet sich http wegen seiner Einfachheit an:

- HTTP Implementierungen gibt es in fast jeder Sprache (Java, C, Perl, Basic, …).
- HTTP gibt es auf fast allen Plattformen (Windows, Mac, Unix...).
- HTTP wird von den meisten Firewalls nicht gestört.
- HTTP ist Text-basiert, kann also leicht mithilfe von Telnet getestet werden.
- HTTP Header sind in der Lage, unterschiedlichste Informationen aufzunehmen, wie Grösse des Dokuments, allfällige Kodierung u.v.m.
- HTTP ist bestens bekannt.

Daher wird sehr oftHTTP als Transport-Protokoll für XML basierte Nachrichten eingesetzt, speziell der POST Befehl. Warum gerade der POST Befehl werden Sie gleich sehen.

2.3.1. Funktionsweise von HTTP

Die Funktionsweise von http können Sie selber testen.

Beispiel:

Lesen Sie eine Web Seite oder eine Datei aus einer URL mithilfe von Telnet.

```
telnet www.hsr.ch 80
GET /robots.txt
```

Oder

```
GET /index.html HTTP/1.1
```

Als Antwort auf die `robots.txt` Anfrage erhalten Sie:

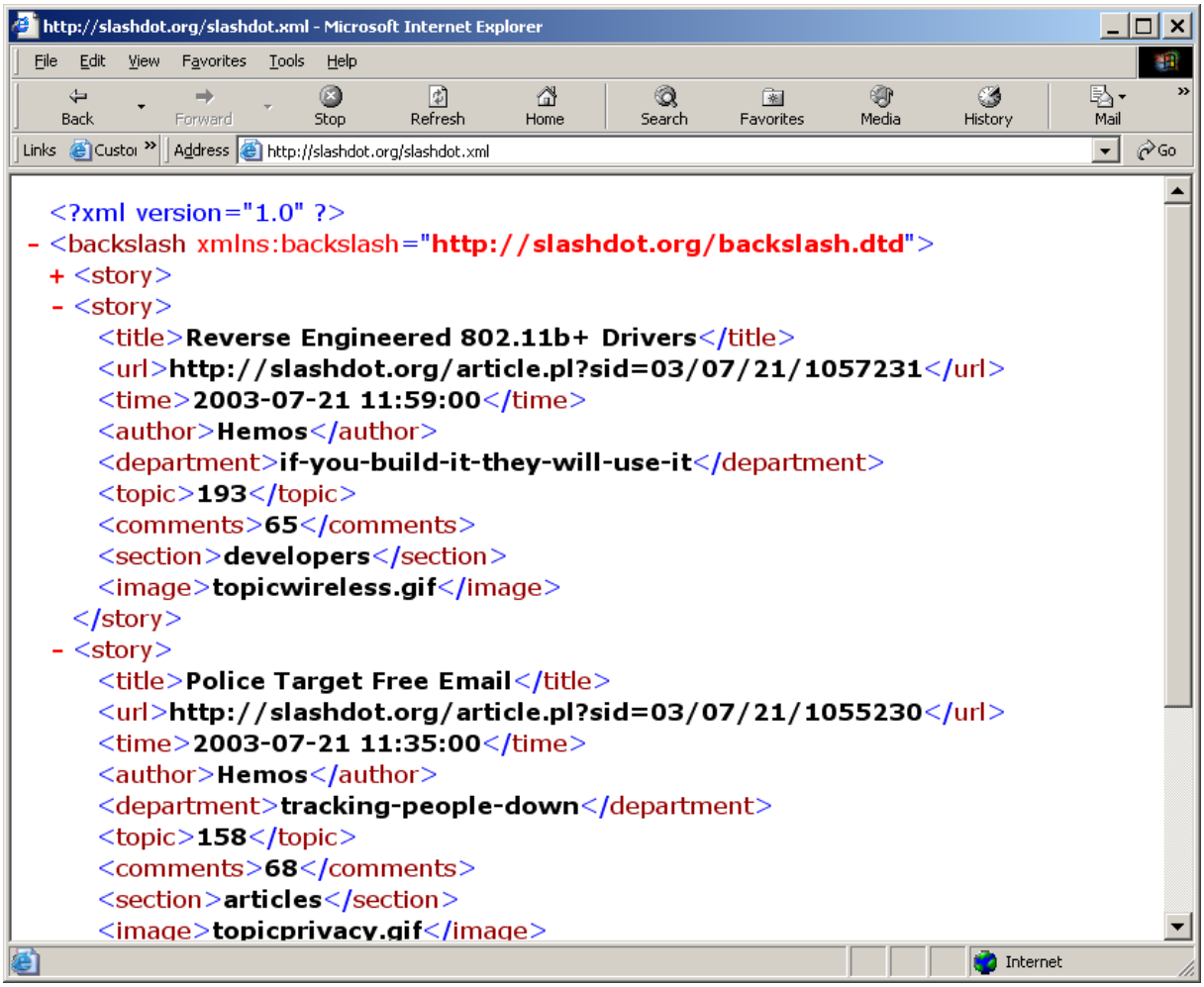
```
# 2002-07-26  webmaster@hsr.ch
# http://www.kollar.com/robots.html
# http://info.webcrawler.com/mak/projects/robots/norobots.html

# valid for all robots and individuals
User-agent: *
```

```
# mostly mirrored or copied
Disallow: /icons                # too many for robot searching
Disallow: /pictures
Disallow: /downloads
Disallow: /images

# changing information
Disallow: /intern               # mostly internal use
Disallow: /cgi-bin              # only indirect access welcome
```

Hier noch ein interessanter Link auf einen Web, welcher aktuelle Informationen als XML Dokument zur Verfügung stellen:

`http://slashdot.org/slashdot.xml.`

Als Antwort erhalten Sie:



Der Browser sendet im Hintergrund folgende Anfrage:

```
GET /slashdot.xml HTTP/1.1
Host: www.slashdot.org
User-Agent: Mozilla/5.0 (Windows; U; WinNT4.0; en-US; rv:0.9.2)
Accept: text/xml, text/html;q=0.9, image/jpeg, */*;q=0.1
Accept-Language: en, fr;q=0.50
Accept-Encoding: gzip,deflate,compress,identity
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Keep-Alive: 300
Connection: keep-alive
```

Und der Server antwortet mit:

```
HTTP/1.1 200 OK
Date: .... GMT
Server: Apache/... (Unix) mod_perl/1.24
Last-Modified: ... GMT
ETag: "...."
Accept-Ranges: bytes
Content-Length: 3678
Connection: close
Content-Type: text/xml
...
```

Das http Protokoll ist ein Text-basiertes Protokoll, Sie können also alle Befehle und den gesamten Dialog als Klartext lesen und eingeben. Falls die Seite, die Sie lesen wollen, Bilder enthält, werden diese Hexa-codiert übermittelt.

2.3.2. HTTP in Java

Alles was wir im Telnet gemacht haben, lässt sich auch leicht mit Java realisieren. Java enthält viele http Klassen, im Package `java.net`.

Das folgende Programm zeigt eine von vielen Möglichkeiten, Texte aus einer URL zu lesen. Dazu werden die Klassen:

- `java.net.URL` mit (u.a.) den Methoden
  `getDocumentAsInputStream()` (liefert den Inhalt der URL ohne http Header)
  `getDocumentAsString()` (liest das gesamte Dokument in eine Zeichenkette)

Beispiel:

```
package kapitel02;

import java.io.IOException;
import java.net.MalformedURLException;


public class URLGrabberTest {

  public static void main(String[] args) {
    System.out.println("[URLGrabberTest]Start");
    int iDocs=args.length;
    String docs[] = new String[Math.max(iDocs,3)];

    if (iDocs<3) {
      docs[0] = "http://www.sdsu.edu";
      docs[1] = "http://www.mit.edu";
      docs[2] = "http://www.ethz.ch";
      iDocs=3;
    }
    for (int i = 0; i < iDocs; i++) {
      System.out.println("\tDokument["+i+"] "+docs[i]);
    }


    for (int i = 0; i < iDocs; i++) {
      try {
        String doc = URLGrabber.getDocumentAsString(docs[i]);
        System.out.println("\tDokument["+i+"] "+docs[i]+"\r\n"+doc);
      }
      catch (MalformedURLException e) {
        System.err.println(docs[i]
         + " ist keine URL.");
      }
      catch (IOException e) {
        System.err.println(" IOException: "
         + e.getMessage());
      }
    }
```

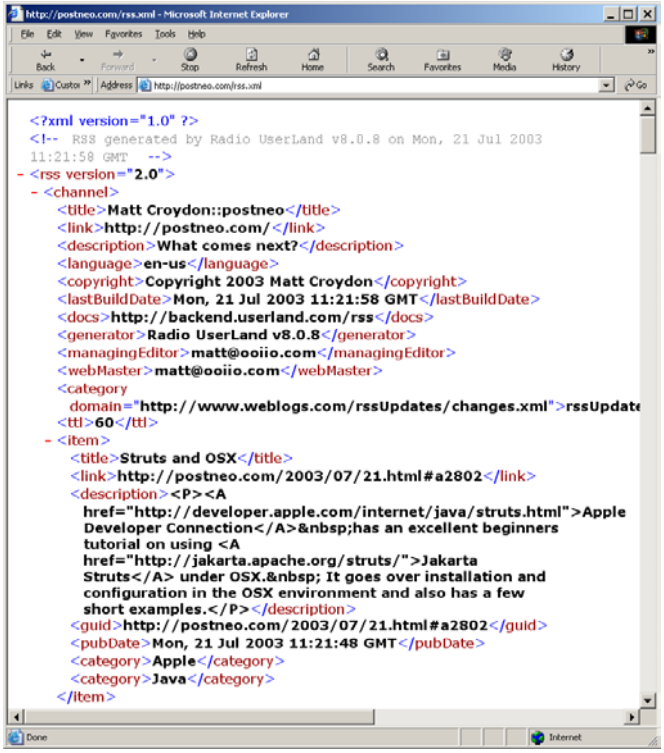
```
  }

}

package kapitel02;

import java.net.URL;
import java.io.IOException;
import java.net.MalformedURLException;
import java.io.InputStream;


public class URLGrabber {

  public static InputStream getDocumentAsInputStream(URL url)
    throws IOException {

    InputStream in = url.openStream();
    return in;

  }

  public static InputStream getDocumentAsInputStream(String url)
    throws MalformedURLException, IOException {

    URL u = new URL(url);
    return getDocumentAsInputStream(u);

  }

  public static String getDocumentAsString(URL url)
    throws IOException {

    StringBuffer result = new StringBuffer();
    InputStream in = url.openStream();
    int c;
    while ((c = in.read()) != -1) result.append((char) c);
    return result.toString();

  }

  public static String getDocumentAsString(String url)
    throws MalformedURLException, IOException {

    URL u = new URL(url);
    return getDocumentAsString(u);

  }

}
```

Beachten Sie, dass kein `Reader` (`getReaderFromURL()`) eingesetzt wurde, da der Datentyp der Daten aus der URL nicht bekannt ist. Testen Sie das Programm mit `www.slashdot.org`.

2.4. RDF Site Summary (RSS)

Im Rahmen des RDF Projekts entstand der RSS Vorschlag für den Austausch von Site Informationen. Sie finden verschiedene Web Sites, welche diesen Standard experimentell bereits einsetzen:

Wenn Sie einen dieser Web Sites auswählen, erhalten Sie eine kurze Zusammenfassung:



Den Standard finden Sie unter `http://web.resource.org/rss/1.0/spec`.

2.5. Verfeinerte Abfragen / Anfragen über HTTP

Ganze Seiten abzufragen ist beispielsweise bei Börsenanwendungen kaum sinnvoll. Sie möchten zum Beispiel die Börsendaten aus dem NASDAQ für die UBS, IBM, Sun Microsystems (SUNW) abfragen.

2.5.1. Abfragen in HTTP

Analog zu Datenbankabfragen können Sie in http Anfragen die möglichen Ergebnisse einschränken, durch *Schlüssel=Wert* Paare.

Beispiel:
`http://quotes.nasdaq.com/quote.dll?page=xml&mode=stock&symbol=UBS`

Welche Schlüsselwörter erlaubt sind, hängt von der Server-Applikation ab.
Hier einige weitere Kurz-Kennzeichen: SUNW steht für Sun Microsystems, IBM, Red Hat RHAT, ... (der Börsenticker auf n-tv hilft Ihnen weiter).

Die eigentliche Arbeit geschieht also auf dem Server!

2.5.2. Der POST Befehl

Die meisten http Anfragen betreffen das Lesen einer Web Seite. Diese Anfragen verwenden den http GET Befehl. Falls Formulare eingesetzt werden, geschieht die Übermittlung an den Server mithilfe des POST Befehls.

Beispiel:
In HTML:

```
<form name="FormName"
action="/cgi-bin/cgiemail/public/de/forms/kontakt.txt" method="post">
<input type="hidden" name="success"
value="http://www.sedrundisentis.ch/public/de/kontakt/formular/contents/
thanks.html"><input type="hidden" name="dummy" value=".">
```

Als Parameter (Eingabewerte) steht ein Query-Ausdruck.
Genauso könnte allerdings anstelle eines Query-Ausdrucks ein ganzes XML Dokument gesendet werden. XML-RPC und SOAP nutzen genau diese Möglichkeit:

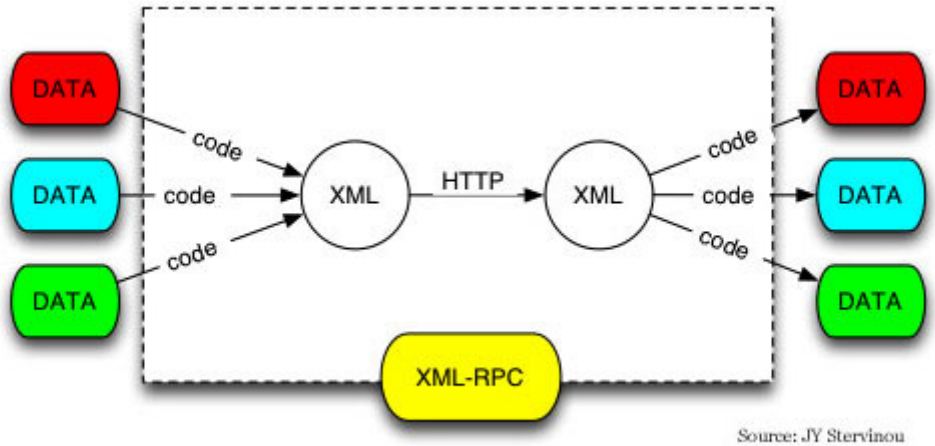
- im http Request werden ganze XML Dokumente gesendet
- mithilfe des POST Befehls können Anfrage und Antwort miteinander verknüpft werden, ohne Zusatzaufwand durch den Benutzer oder Programmierer.

Dies entspricht genau dem RPC Muster (Anfrage/Antwort). Sie finden bei den Beispielen eine Implementierung von POST, einmal mit Sockets (mit http Header und all dem Zeugs), einmal mit URLConnection (also ohne den http Header).

2.6. XML-RPC

XML-RPC wurde von Dave Winer von Userland definiert. Der Name RPC (Remote Procedure Call) wurde gewählt, weil eine Prozedur oder Funktion auf einem entfernen rechner aufgerufen wird, mit einem oder mehreren Argumenten. Als Rückgabe sind unterschiedliche Datentypen, auch `void`, erlaubt.

Winer schuf diese Art der Kommunikation, weil er ein Tool entwickelt hatte, um Web Pages publizieren zu können. Sein Anliegen war, Kunden remote helfen zu können, auf einfache Art und Weise. Existierende Technologien, wie CORBA oder RMI waren ihm zu komplex.



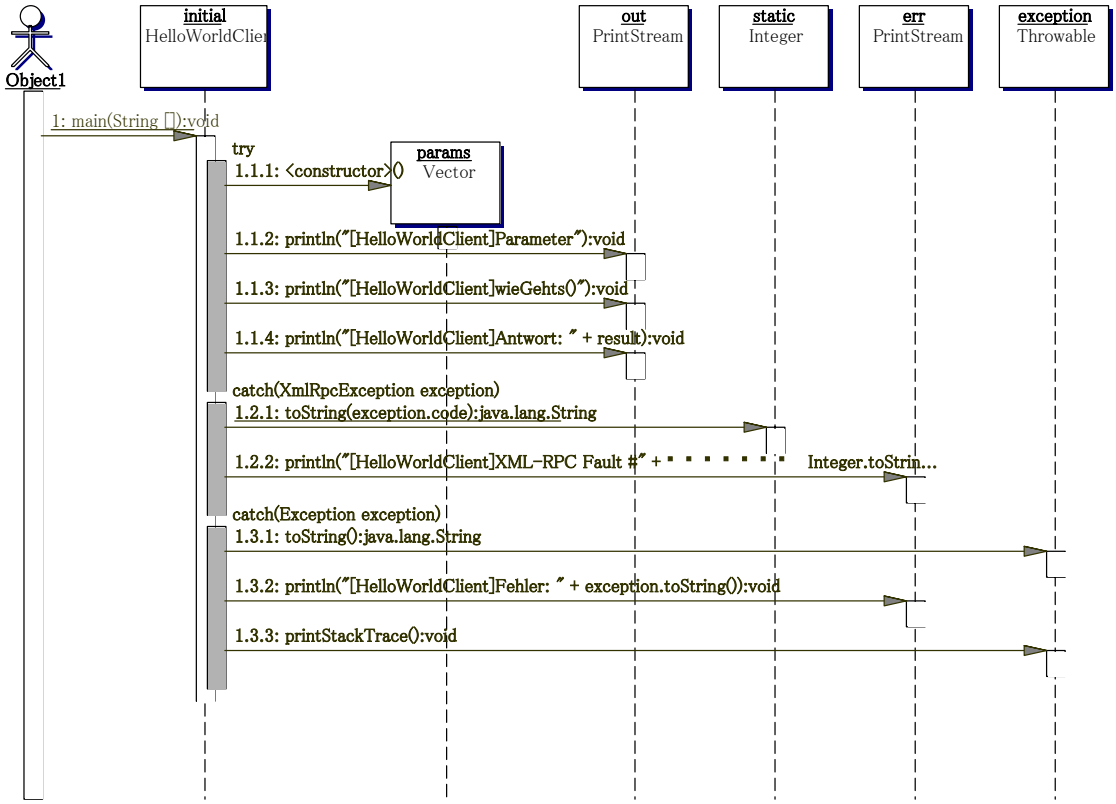
XML-RPC (die Apache Version) löst nicht alle Probleme und ist klar auf 80-90% Problemlösung ausgerichtet. Insbesondere fehlen Stubs, Skeletons und Garbage Collection genauso wie Activation, Callback u.v.m.

Grundidee:

1) ein XML Dokument enthält einen Methodennamen und Argumente dieser Methode;
2) das Dokument wird mit POST an einen Web Server gesandt;
3) der Web Server (ein Servlet oder CGI Skript) analysiert das XML,
   führt die Methode mit den Argumenten aus und verpackt das Ergebnis in XML und
   schickt das Ergebnis an den Aufrufenden zurück;
4) der Client analysiert das zurück erhaltene XML Dokument.

Beispiel:
Im Fall des HelloWorld XML-RPC Beispiels, sieht dies folgendermassen aus:

Der HelloWorldClient:



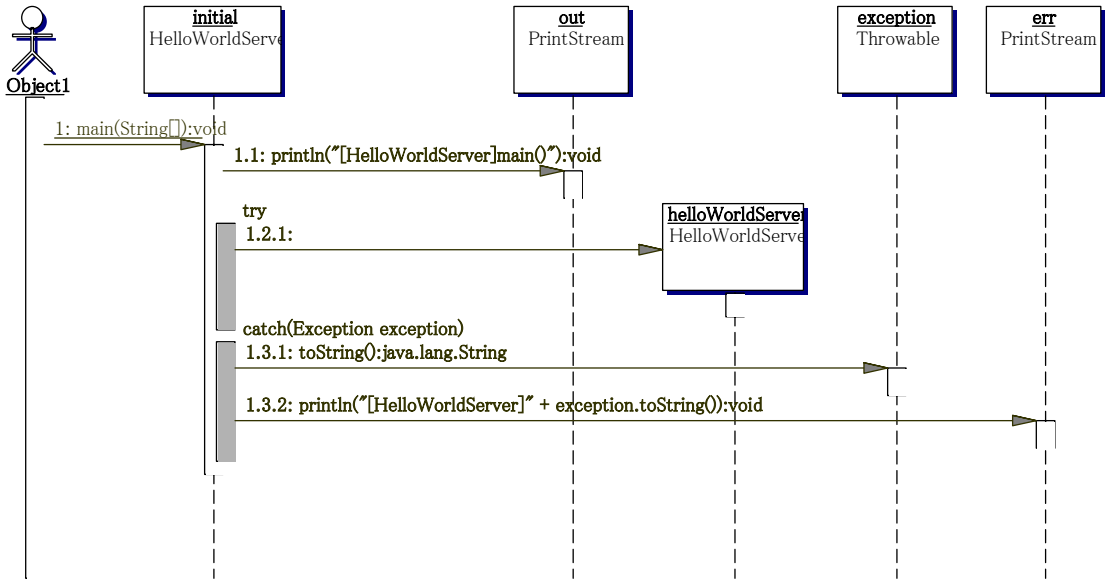
und hier als Java:

```
String server_url ="http://localhost:9090/";
// Anbindung an den Server.
XmlRpcClient srvr = new XmlRpcClient(server_url);

// Parameterliste.
Vector params = new Vector();

// Aufruf des Servers, Ausgabe des Ergenisses.
String result =(String)srvr.execute("helloWorld.wieGehts", params);
```

Hier die Ausgabe:

```
[HelloWorldClient]Parameter
[HelloWorldClient]wieGehts()
[HelloWorldClient]Antwort: Gut! Wie geht's Dir?
```

Der HelloWorldServer:



und hier als Java:

```
public String wieGehts(){
        return "Gut! Wie geht's Dir?";
}
public static void main(String[] args) {
        WebServer server = new WebServer(9090);
        server.addHandler("helloWorld", new HelloWorldServer());
…
```

Die ganze Kommunikation wird also kaum sichtbar abgewickelt. Damit man etwas transparenter sieht, was so passiert, habe ich zwei Programme geschrieben:

1) `SendToURL` : senden eines XML-RPC Requests an eine URL (Server)
2) `SocketToURL` : senden eines XML-RPC Requests inklusive http Header

Schauen wir uns die beiden Teile (XML-Dokument, vollständige http Message) genauer an.

Beispiel: XML-Dokument

`SendToURL:`

XML-Dokument zu `XMLRPCBerechnungsClient` und `XMLRPCBerechnungsServer`

```
<?xml version="1.0" ?>
- <methodCall>
    <methodName>computer.add</methodName>
  - <params>
    - <param>
      - <value>
          <int>3</int>
        </value>
      </param>
    - <param>
      - <value>
          <int>123456789</int>
        </value>
      </param>
    </params>
  </methodCall>
```

1) Das Wurzel-Element des XML-RPC Request Dokuments ist `<methodCall>.`
2) Der Methodenname steht im Element `<methodCall>`
3) Die Parameter der Methode stehen in `<params>`
4) Jeder Parameter steht in einem eigenen `<param>`
5) Jeder Parameter besitzt einen XMLRPC Datentyp `<int>`
   Falls der Parametertyp nicht mit dem Methodenparametertyp übereinstimmt, wird eine Exception geworfen (mehr darüber später).
   Mögliche Datentypen bei Apache XML-RPC sind `boolean`, `double`, `dateTime`, `base64`, `int` und `string`.

Beispiel: ganzer http Call

`SocketToURL`

XML-RPC Call zu `XMLRPCBerechnungsClient` und `XMLRPCBerechnungsServer`

```
POST /computer HTTP/1.1
Content-Length: 254
Content-Type: text/xml


<?xml version="1.0" encoding="ISO-8859-1"?>
      <methodCall>
            <methodName>computer.add</methodName>
            <params>
                  <param>
                        <value>
                              <int>12</int>
                        </value>
                  </param>
                  <param>
                        <value>
                              <int>15</int>
                        </value>
                  </param>
            </params>
      </methodCall>
```

Die Länge des Dokuments muss dabei exakt bestimmt werden. Im Programm geschieht dies:

```
String http_POST_header =     "POST "+path+" HTTP/1.1\r\n"+
                              "Content-Length: "+
                              xml_rpc_call.length()+"\r\n"+
                              "Content-Type: text/xml\r\n"+
```

In der String Variable xml_rpc_call steht das aus einer Datei eingelesene XML Dokument.

Beispiel: http Response
`SocketToURL`
XML-RPC Call zu `XMLRPCBerechnungsClient` und `XMLRPCBerechnungsServer`

```
HTTP/1.1 200 OK
Server: Apache XML-RPC 1.0
Connection: close
Content-Type: text/xml
Content-Length: 201

<?xml version="1.0" encoding="ISO-8859-1"?>
      <methodResponse>
            <params>
                  <param>
                        <value>
                              <struct>
                                    <member>
                                          <name>add</name>
                                          <value>
                                                <int>27</int>
                                          </value>
                                    </member>
                              </struct>
                        </value>
                  </param>
            </params>
      </methodResponse>
```

Soweit ein erfolgreicher Methodenaufruf. Später werden wir noch einige Fehler einbauen und die Fehlermeldungen genauer ansehen.

Gemäss Apache XML-RPC bestehen folgende Umsetzungsvorschriften bzgl. Datentypen:

XML-RPC Datentyp	Vom Parser generiert	Vom Handler erwartet
<i4> oder <int>	java.lang.Integer	int
<boolean>	java.lang.Boolean	boolean
<string>	java.lang.String	java.lang.String
<double>	java.lang.Double	double
<dateTime.iso8601>	java.util.Date	java.util.Date
<struct>	java.util.Hashtable	java.util.Hashtable
<array>	java.util.Vector	java.util.Vector
<base64>	byte[ ]	byte[ ]

2.6.1. Datenstrukturen

Neben den Basisdatentypen kann man in XML-RPC auch zusammengesetzte Daten übermitteln. Dazu stehen Arrays und Structures zur Verfügung. Beide Datenstrukturen können auch verschachtelt werden (Array von Arrays, Array aus Structures). Daraus lassen sich fast beliebig komplexe Datenstrukturen aufbauen.

2.6.1.1. Arrays

Im Apache XML-RPC entsprechen XML-RPC Arrays `java.util.Vector`.

Beispiel:

`XMLRPCArrayMessage.xml` aus `XMLRPCArrayClient XMLRPCArrayServer`

```
<?xml version="1.0"?>
	<methodCall>
	<methodName>arrayServer.setAll</methodName>
		<params>
			<param>
				<value>
					<array>
						<data>
							<value>
							<string>Bert Simpson</string>
							</value>
							<value>
							<string>Seestrasse</string>
							</value>
							<value>
							<int>12</int>
							</value>
							<value>
							<string>Herrliberg</string>
							</value>
							<value>
							<int>100101</int>
							</value>
							</data>
						</array>
					</value>
				</param>
			</params>
	</methodCall>
```

In Java, auf der Server Seite:

```
	public void setAll(Vector pVec){
		name = (String)pVec.elementAt(0);
		strasse = (String)pVec.elementAt(1);
		strassen_nr = ((Integer)pVec.elementAt(2)).intValue();
		wohnort = (String)pVec.elementAt(3);
		matrikel_nr = ((Integer)pVec.elementAt(4)).intValue();
```

bzw. als Antwort:

```
	public Vector getAdressArray() {
		Vector vec = new Vector();
		vec.add(0,name);
		vec.add(1,strasse);
		vec.add(2,wohnort);
		return vec;
	}
```

und hier ein Antwort (Hinweis: ich habe jeweils entweder nur Werte gesandt oder abgefragt, nicht beides gleichzeitig: XML-RPC hat damit Probleme):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <methodResponse>
  - <params>
    - <param>
      - <value>
        - <array>
          - <data>
              <value>Bert Simpson</value>
              <value>Seestrasse</value>
              <value>Herrliberg</value>
            </data>
          </array>
        </value>
      </param>
    </params>
  </methodResponse>
```

2.6.1.2. Structs

In XML-RPC sind Structs Sammlungen von <Name, Wert> Paaren. Die in Java angepasste Datenstruktur sind die Hash-Tabellen.

Beispiel:

XMLRPCStructMessage.xml aus XMLRPCStructClient XMLRPCStructServer

```
<?xml version="1.0"?>
      <methodCall>
            <methodName>structServer.setAll</methodName>
                  <params>
                        <param>
                              <value>
                                    <struct>
                                    <member>
                                          <name>name</name>
                                          <value>
                                          <string>Bert Simpson</string>
                                          </value>
                                          </member>
                                          <member>
                                          <name>strasse</name>
                                          <value>
                                          <string>Seestrasse</string>
                                          </value>
                                          </member>
                                          <member>
                                          <name>strassen_nr</name>
                                          <value>
                                          <int>12</int>
                                          </value>
                                          </member>
                                          <member>
                                          <name>wohnort</name>
                                          <value>
                                          <string>Herrliberg</string>
                                          </value>
                                          </member>
                                          <member>
                                          <name>matrikel_nr</name>
                                          <value>
                                          <int>100101</int>
                                          </value>
                                          </member>
                                    </struct>
                              </value>
```

```
                        </param>
                    </params>
        </methodCall>
```

In Java, auf der Server Seite:

```
        public void setAll(Hashtable pHash){
              name = (String)pHash.get("name");
              strasse = (String)pHash.get("strasse");;
              strassen_nr = ((Integer)pHash.get("strassen_nr")).intValue();
              wohnort = (String)pHash.get("wohnort");
              matrikel_nr = ((Integer)pHash.get("matrikel_nr")).intValue();
```

bzw. als Antwort:

```
        public Hashtable getAdressStruct() {
              Hashtable ht = new Hashtable();
              ht.put("name",name);
              ht.put("strasse", strasse);
              ht.put("wohnort", wohnort);
              return ht;
```

und hier ein Antwort (Hinweis: ich habe jeweils entweder nur Werte gesandt oder abgefragt, nicht beides gleichzeitig: XML-RPC hat damit Probleme):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <methodResponse>
  - <params>
    - <param>
      - <value>
        - <struct>
          - <member>
              <name>wohnort</name>
              <value>Herrliberg</value>
            </member>
          - <member>
              <name>name</name>
              <value>Bert Simpson</value>
            </member>
          - <member>
              <name>strasse</name>
              <value>Seestrasse</value>
            </member>
          </struct>
        </value>
      </param>
    </params>
  </methodResponse>
```

2.6.2. Fehler

Die Fehlermeldungen in XML-RPC bestehen aus <`faultCode, faultString`>.

Beispiel:

`XMLRPCStructMessage.xml` aus `XMLRPCStructClient XMLRPCStructServer`

```
<member>
      <name>matrikel_nr</name>
      <value>
```

```
            <String>100101</String> <!--ist als int deklariert //-->
        </value>
</member>
```

Fehlermeldung:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <methodResponse>
  - <fault>
    - <value>
      - <struct>
        - <member>
            <name>faultString</name>
            <value>java.lang.Exception:
              java.lang.ClassCastException</value>
          </member>
        - <member>
            <name>faultCode</name>
          - <value>
              <int>0</int>
            </value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

2.6.3. Validation von XML-RPC

Es gibt keine offizielle DTD oder ein Schema, welches XML-RPC beschreibt. Aber der Aufbau des XML-RPC zugeordneten XML Dokuments ist recht einfach.

2.6.3.1. DTD für XML-RPC

Mogliche DTD:

```
<!ELEMENT methodCall (methodName, params)>
<!ELEMENT methodName (#PCDATA)>
<!ELEMENT params     (param*)>
<!ELEMENT param      (value)>
<!ELEMENT value
   (i4|int|string|dateTime.iso8601|double|base64|struct|array)>

<!ELEMENT i4               (#PCDATA)>
<!ELEMENT int              (#PCDATA)>
<!ELEMENT string           (#PCDATA)>
<!ELEMENT dateTime.iso8601 (#PCDATA)>
<!ELEMENT double           (#PCDATA)>
<!ELEMENT base64           (#PCDATA)>

<!ELEMENT array            (data)>
<!ELEMENT data             (value*)>
<!ELEMENT struct           (member+)>
<!ELEMENT member           (name, value)>
<!ELEMENT name             (#PCDATA)>

<!ELEMENT methodResponse   (params | fault)>
<!ELEMENT fault            (value)>
```

Die Integer Variablen müssen zwischen -2,147,483,648 und 2,147,483,647 sein.

2.6.3.2. XML Schema für XML-RPC

XML Spy verhilft auch hier sofort zu einem Vorschlag:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!—Mögliche Roots: methodResponse und
       methodCall -->

  <xsd:element name="methodCall">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="methodName">
          <xsd:simpleType>
            <xsd:restriction base="ASCIIString">
              <xsd:pattern value="([A-Za-z0-9]|/|\.|:|_)*" />
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="params" minOccurs="0" maxOccurs="1">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="param"  type="ParamType"
                           minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
         </xsd:element>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
```

```
<xsd:element name="methodResponse">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="params">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="param" type="ParamType"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="fault">
        <!—Inhalt einer Fehlermeldung -->
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="value">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="struct">
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:element name="member"
                                     type="MemberType">
                        </xsd:element>
                        <xsd:element name="member"
                                     type="MemberType">
                        </xsd:element>
                      </xsd:sequence>
                    </xsd:complexType>
                  </xsd:element>
                </xsd:sequence>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

<xsd:complexType name="ParamType">
  <xsd:sequence>
    <xsd:element name="value" type="ValueType"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ValueType" mixed="true">
  <!-- I need to figure out how to say that this
       is either a simple xsd:string type or that
       it contains one of these elements; but that otherwise
       it does not have mixed content -->
  <xsd:choice>
    <xsd:element name="i4"            type="xsd:int"/>
    <xsd:element name="int"           type="xsd:int"/>
    <xsd:element name="string"        type="ASCIIString"/>
    <xsd:element name="double"        type="xsd:decimal"/>
    <xsd:element name="Base64"        type="xsd:base64Binary"/>
    <xsd:element name="boolean"       type="NumericBoolean"/>
    <xsd:element name="dateTime.iso8601" type="xsd:dateTime"/>
    <xsd:element name="array"         type="ArrayType"/>
    <xsd:element name="struct"        type="StructType"/>
  </xsd:choice>
```

```
  </xsd:complexType>

  <xsd:complexType name="StructType">
    <xsd:sequence>
      <xsd:element name="member" type="MemberType"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="MemberType">
    <xsd:sequence>
      <xsd:element name="name"  type="xsd:string" />
      <xsd:element name="value" type="ValueType"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="ArrayType">
    <xsd:sequence>
      <xsd:element name="data">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="value"  type="ValueType"
                         minOccurs="0" maxOccurs="unbounded"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="ASCIIString">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([ -~]|\n|\r|\t)*" />
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="NumericBoolean">
    <xsd:restriction base="xsd:boolean">
      <xsd:pattern value="0|1" />
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

2.7. SOAP

Nachdem XML-RPC seine ersten Erfolge hatte, wurde von Microsoft und IBM eine verbesserte Form der Idee gesucht: Simple Object Access Protocol wurde geboren.

XML-RPC ist bezüglich Datentypen sehr limitiert. Auch konzeptionell war XML-RPC das Werk eines Einzelnen, des Besitzers von Userland. SOAP dagegen ist das Produkt eines Teams. Aber auch SOAP wurde rasch in einer verbesserten Version publiziert.

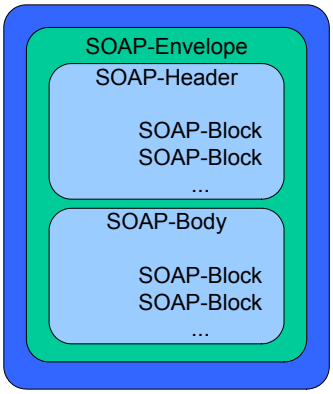
SOAP unterstützt mehr Datentypen als XML-RPC und der Aufbau einer SOAP Message ist formal klarer geregelt.

2.7.1. S wie Simple(r)

Der grundlegende Kommunikationsmechanismus von SOAP ist http, wobei aber die "Bindung" an http nicht zwingend ist. http ist einfach, also...SOAP Dokumente mit Attachments versteht man viel besser, wenn man an Stelle des http das SMTP Protokoll als Modell verwendet. SOAP Messages sind im Grund Einwegnachrichten. Aber in der Praxis werden mehrere Messages kombiniert, zu einer Request / Response Pattern Lösung.

2.7.2. Ein SOAP Beispiel

Als erstes betrachten wir lediglich die SOAP Message. Eine SOAP Message steckt in einem Umschlag (*SOAP-Envelope*); die eigentliche Message kann mehrere Headers sowie mehrere Rümpfe enthalten. Mehrere Header können immer dann sehr sinnvoll sein, wenn Sie eine Nachricht über verschiedene Zwischenknoten an einen Kunden oder einen Server senden müssen. Die Header können in solchen Anwendungen bei den Zwischenknoten mutiert werden.



Das folgende Beispiel übermittelt eine Bestellung an ein Lager. Die Bestellung enthält alle relevanten Business-Informationen.

Beispiel:
(basiert auf Apache SOAP:
`GenericHTTPSoapClient_MitTunnel, http://localhost:5555/JavaWebServices/servlet/apachesoap.SimpleHTTPReceive`
mit dem Apache Tunnel
[`javaw org.apache.soap.util.net.TcpTunnelGui 5555 localhost 8080`] damit Client und Server Message besser erkennbar werden).

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Header>
      <jaws:MessageHeader xmlns:jaws="urn:jaws-beispiele"
      SOAP-ENV:mustUnderstand="1" xmlns:SOAP-ENV="urn:jaws-beispiele">
      <From>From_Client</From>
      <To>To_Client</To>
      <MessageId>9999</MessageId></jaws:MessageHeader>
</SOAP-ENV:Header>
```

```
<SOAP-ENV:Body>
<!--ab hier folgt das XML Dokument -->
      <PurchaseOrder xmlns="urn:jaws-beispiele">
      <shipTo country="CH">
            <name>Josef M. Joller</name>
            <street>Seestrasse 10</street>
            <city>Bluemlisalp</city>
            <state>ZH</state>
            <zip>1234</zip>
      </shipTo>
      <items>
            <item partNum="872-AA">
                  <productName>Cortex Schuhe</productName>
                  <quantity>1 Paar</quantity>
                  <price>168</price>
                  <comment>Keine nassen Fuesse mehr!</comment>
            </item>
      </items>
      </PurchaseOrder>
<!-- hier endet das XML Dokument -->
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Die Transformation eines XML Dokuments in eine SOAP Message.

1) das XML Dokument wird in einen SOAP Body eingepackt.
2) Dieser SOAP Body muss in einen SOAP Envelope eingepackt werden.
3) Die SOAP Headers sind optional.
4) Der Namensraum muss deklariert werden (SOAP 1.1).
5) Encoding Style falls serialisiert werden soll.
6) Und schliesslich erfolgt die Bindung an ein Protokoll

Nun zu den einzelnen Komponenten.

2.7.2.1. SOAP Umschlag

Der Umschlag ist der äusserste XML Tag einer SOAP Nachricht:

```
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Zum Aufbau: insgesamt werden drei Namensräume deklariert

1) der Namensraum SOAP-ENV (SOAP Namensraum)
2) der Namensraum xsi (SOAP Instanzen Namensraum)
3) der Namensraum xsd (Schema Namensraum)

2.7.2.2. SOAP Header

In SOAP 1.1 und 1.2 ist eigentlich völlig offen, was im SOAP Header eingepackt werden soll.

```
<SOAP-ENV:Header>
      <jaws:MessageHeader xmlns:jaws="urn:jaws-beispiele"
      SOAP-ENV:mustUnderstand="1" xmlns:SOAP-ENV="urn:jaws-beispiele">
      <From>From_Client</From>
      <To>To_Client</To>
      <MessageId>9999</MessageId></jaws:MessageHeader>
</SOAP-ENV:Header>
```

2.7.2.3. SOAP Protokoll Bindung

In unserem Beispiel verwenden wir eine http Bindung:

```
POST /JavaWebServices/servlet/apachesoap.SimpleHTTPReceive HTTP/1.0
```

```
Host: localhost:5555
Content-Type: text/xml; charset=utf-8
Content-Length: 961
SOAPAction: "urn:jaws-beispiele"

<?xml version='1.0' encoding='UTF-8'?>
...
```

Zum Aufbau:

1) SOAPAction : in SOAP 1.1 war dieses Feld zwingend erforderlich. Es war Teil der http Bindung.
2) In SOAP 1.2 ist die Angabe freiwillig.

2.7.3. POST eines SOAP Dokuments

Die folgende Ausgaben wurden mit dem oben erwähnten Tunnel bestimmt.
Die Ausgabe des Clients haben Sie oben schon gesehen, inklusive dem POST Header:

```
POST /JavaWebServices/servlet/apachesoap.SimpleHTTPReceive HTTP/1.0
Host: localhost:5555
Content-Type: text/xml; charset=utf-8
Content-Length: 961
SOAPAction: "urn:jaws-beispiele"
…
```

Der Jakarta SOAP Server antwortet darauf mit:

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 0
Date: Thu, 07 Aug 2003 19:22:10 GMT
Server: Apache Coyote/1.0
Connection: close
```

2.7.4. Fehlerbehandlung

Sie können wie bei XML-RPC oder `java.net.Socket` Fehler auf unterschiedlichen Ebenen erhalten:

1) der Server kennt das Servlet nicht
2) der Zugriff auf das Servlet ist geschützt
3) die Parameter in der SOAP Message sind nicht korrekt.

Als Fehlermeldung sendet der Server ein XML Dokument an den Client mit einer Fehlermeldung analog zu XML-RPC (Fehlercode, Fehlerbeschreibung, evtl. Fehlerdetails).

2.7.5. Encoding Styles

Damit die Inhalte einer SOAP Message korrekt interpretiert werden können, muss festgelegt werden, wie ein SOAP Inhalt in einen Java Inhalt umgewandelt werden kann. Dazu können natürlich auch XML Schemas eingesetzt werden.

SOAP Datentyp	Java Datentyp
SOAP-ENC:string	`java.lang.String`
SOAP-ENC:boolean	`Boolean`
SOAP-ENC:decimal	`java.math.BigDecimal`
SOAP-ENC:float	`Float`
SOAP-ENC:double	`Double`
SOAP-ENC:integer	`java.math.BigInteger`
SOAP-ENC:positiveInteger	`java.math.BigDecimal`
SOAP-ENC:nonPositiveInteger	`java.math.BigInteger`
SOAP-ENC:negativeInteger	`java.math.BigInteger`
SOAP-ENC:nonNegativeInteger	`java.math.BigInteger`
SOAP-ENC:long	`Long`
SOAP-ENC:int	`Int`
SOAP-ENC:short	`Short`
SOAP-ENC:byte	`Byte`
SOAP-ENC:unsignedLong	`double` oder `java.math.BigInteger`
SOAP-ENC:unsignedInt	`Long`
SOAP-ENC:unsignedShort	`Int`
SOAP-ENC:unsignedByte	`Int`
SOAP-ENC:duration	Spezielle Klasse (selber implementieren)
SOAP-ENC:dateTime	`java.util.Date`
SOAP-ENC:time	`java.sql.Time`
SOAP-ENC:date	`java.sql.Date`
SOAP-ENC:gYearMonth	(selber implementieren)
SOAP-ENC:gYear	(selber implementieren), `int`, oder `java.math.BigInteger`
SOAP-ENC:gMonthDay	(selber implementieren)
SOAP-ENC:gDay	(selber implementieren) `int`
SOAP-ENC:gMonth	(selber implementieren) oder `int`
SOAP-ENC:hexBinary	`byte[]`
SOAP-ENC:base64Binary	`byte[]`
SOAP-ENC:anyURI	`java.net.URL`, `java.lang.String`, oder (selber

SOAP Datentyp	Java Datentyp
	implementieren)
SOAP-ENC:QName	`java.lang.String` oder (selber implementieren)
SOAP-ENC:NOTATION	`org.w3c.dom.Notation`
SOAP-ENC:normalizedString	`java.lang.String`
SOAP-ENC:token	`java.lang.String`
SOAP-ENC:language	`java.lang.String` oder (selber implementieren)
SOAP-ENC:NMTOKEN	`java.lang.String` oder (selber implementieren)
SOAP-ENC:NMTOKENS	`java.lang.String` oder (selber implementieren)
SOAP-ENC:Name	`java.lang.String`
SOAP-ENC:NCName	`java.lang.String`
SOAP-ENC:ID	`java.lang.String`
SOAP-ENC:IDREF	`java.lang.String`
SOAP-ENC:IDREFS	ein Array oder eine Liste aus `java.lang.String`s oder (selber implementieren)
SOAP-ENC:ENTITY	`org.w3c.dom.Entity`
SOAP-ENC:ENTITIES	eine `org.w3c.dom.NodeList` mit `org.w3c.dom.Entity` Objekten.

2.7.5.1. Structs

Strukturen sind in SOAP analog zu XML-RPC definiert:

```
<Aktion xmlns="http://namespaces.migros.ch/xmljava/">
  <Artikel>Trauben</Artikel>
  <Preis ISO="CHF">2.50</Preis>
</Aktion>
```

2.7.5.2. Das `mustUnderstand` Attribut

Falls dieses Attribut im Header einer SOAP Message gesetzt ist (Wert: 1), muss der Empfänger den Header verarbeiten. Dieser Mechanismus kann beispielsweise bei der Weiterleitung einer Nachricht eine Rolle spielen.

2.7.6. SOAP's Grenzen

SOAP schreibt, wie so viele W3C "Standards", sehr wenig zwingend vor. Beispielsweise wird weder eine DTD nach ein Schema für die SOAP Message verlangt. Mit andern Worten:

- eine SOAP Message kann ein ungültiges XML Dokument im Body enthalten.
- DTD's sind explizit nicht gestattet, als Teil der Nachricht.

SOAP 1.1 ist in dieser Beziehung (und einigen anderen) eher schlecht und versucht zu allgemein zu sein. In SOAP 1.2 wurde einiges korrigiert. Aber die nächste Version kommt bestimmt.

2.7.6.1. Validation von SOAP

Da die Spezifikation kein DTD zulässt, muss die Validation mithilfe von Schemas geschehen.

2.8. Eigene XML Protokolle

XML-RPC ist eher zu einfach; SOAP eher zu komplex. Im Java Web Services Development Kit wurden Versionen vorgestellt, und in Apache Axis realisiert, welche den Einsatz von SOAP wesentlich vereinfachen.

Beispiel:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Vector;


public class SimpleGenericHTTPSoapClient_2 {
…
      public void sendSOAPMessage(){
      try {
            // SOAP Body für den Umschlag
            FileReader fr = new FileReader (m_dataFileName);
            javax.xml.parsers.DocumentBuilder xdb =
            org.apache.soap.util.xml.XMLParserUtils.getXMLDocBuilder();
            org.w3c.dom.Document doc =
                        xdb.parse (new org.xml.sax.InputSource (fr));
…
            //Vector mit den header Elementen
            Vector headerElements = new Vector();
            // Header Element Namespace
            org.w3c.dom.Element headerElement =
                        doc.createElementNS(URI,"jaws:MessageHeader");
…
            // SOAP Umschlag
            org.apache.soap.Envelope envelope =
                        new org.apache.soap.Envelope();
            // SOAP Body als Vector Elemente
            Vector bodyElements = new Vector();
            // DOM Root
            bodyElements.add(doc.getDocumentElement ());
            // SOAP Body Elemente
            org.apache.soap.Body body = new org.apache.soap.Body();
            body.setBodyEntries(bodyElements);
            // SOAP Body Elemente in den Umschlag stecken
            envelope.setBody(body);
            // SOAP Message aufbauen
            org.apache.soap.messaging.Message msg =
                        new org.apache.soap.messaging.Message();
      try {
            java.net.URL u = new java.net.URL(m_hostURL);
            msg.send (new java.net.URL(m_hostURL), URI, envelope);
      } catch(NullPointerException npe) {npe.printStackTrace();}
      // Ausgabe der Antwort
      org.apache.soap.transport.SOAPTransport st = msg.getSOAPTransport ();
      BufferedReader br = st.receive ();
……
```

Wir werden sehen, wie viel einfacher die selbe Aufgabe mit Jax (und Axis) zu lösen ist.

2.9. Zusammenfassung

XML Protokolle benutzen in der Regel http als Transport-Protokoll (zum Teil SMTP). Im Vergleich zu CORBA oder RMI sind die Protokolle wesentlich transparenter. Allerdings wurde mit SOAP bereits wieder eine Komplexität und Inkompatibilität erreicht, die den Erfolg von SOAP gefährdeten. Zur Zeit sind mehrere Teams am Erarbeiten von Interoperabilitäts-Standards beschäftigt (IBM, Sun, Microsoft, Bea, ..). Bleibt zu hoffen, dass etwas dabei herauskommt.

XML-RPC ist als erste Idee sehr bemerkenswert, weil damit ein Paradigmawechsel eingeleitet wurde. SOAP startet eventuell zu tief. Aber historisch ist dies verständlich. Erst die Web Services Explosion führte zu einer grossen Verbreitung von SOAP, in eine Richtung, die ursprünglich nicht vorauszusehen war.

2.10. Anhang – Web Services Architekturen

Das Thema "Architektur von Web Services" ist in Bearbeitung, bei verschiedenen Firmen und W3C, sowie weiteren Organisationen. Auf die einzelnen Inhalte gehen wir noch ein. Hier geht es um die Einordnung von SOAP, ... in einen Gesamtkontext.

2.10.1.1. Architektur gemäss WebServices.org

Layer	Beispiel
Service Negotiation	Trading Partner Agreement
Workflow, Discovery, Registries	UDDI, ebXML registries, IBM WSFL, MS XLANG
Service Description Language	WSDL/WSCL
Messaging	SOAP/XML Protocol
Transport Protocols	HTTP, HTTPS, FTP, SMTP
Business Issues	Management, Quality of Service, Security, Open Standards

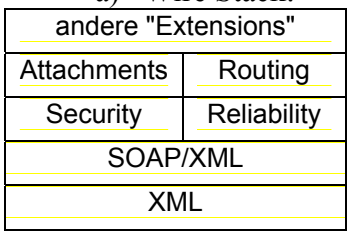
2.10.1.2. IBM

Tools	Layer		
TPA (Trading Partner Agreement)	Service Negotiation		
WSFL	Service Flow		
UDDI+WSEL	Service Description	Service Publication (Direct UDDI) Service Directory (Static UDDI)	Endpoint Description
WSDL	Service Implementation		
SOAP	XML-Based Messaging		
HTTP, FTP, email, MQ, IIOP	Network		
Quality of Service, Management, Security	Business Issues		

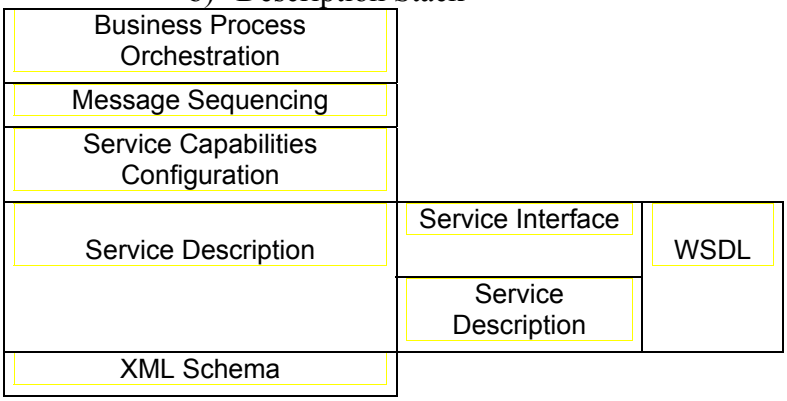
2.10.1.3. W3C

Die Arbeitsgruppe definiert verschiedene Stacks:

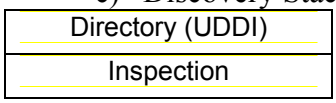
a) Wire Stack:



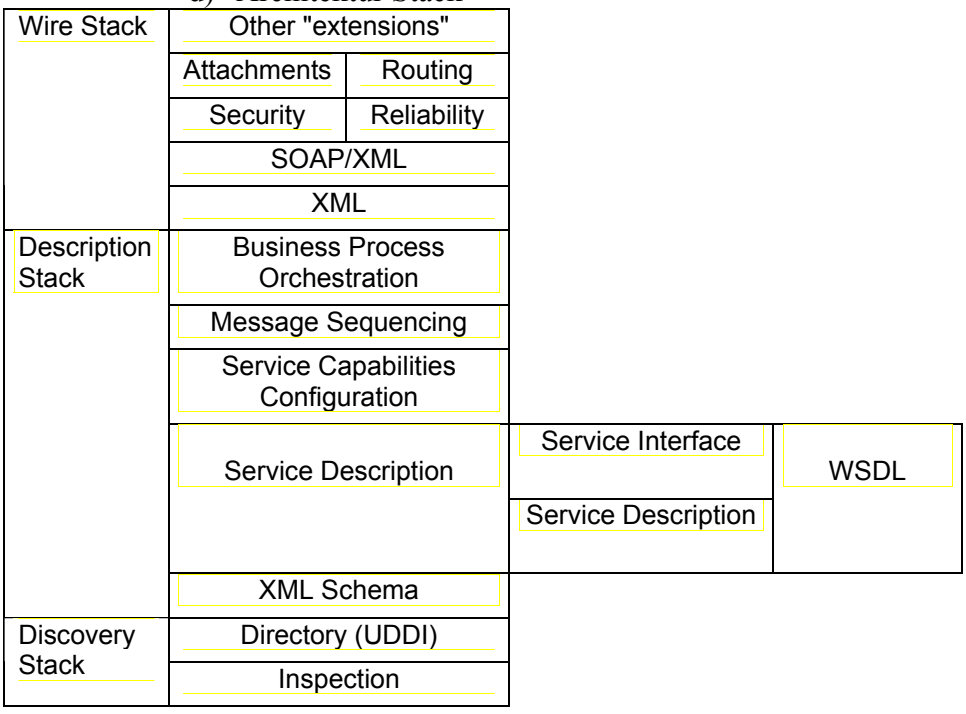
b) Description Stack



c) Discovery Stack



d) Architektur Stack



2. XML PROTOKOLLE: XML-RPC UND SOAP