## Web Services

Die Definition eines **Web Services** laut **W3C (World Wide Web Consortium)** ist:
*„Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks."*

Web Services kommunizieren über den Austausch von **XML-Dateien (eXtensible Markup Language)**. Somit sind sie **unabhängig** von der Wahl der **Programmiersprache**. Übertragen werden die Dateien mittels **HTTP** über **TCP/IP**.
Es wurde ein Wandel vollzogen von der **Objektorientierung** hin zur **Serviceorientierung**. Services kapseln Daten und Funktionen und werden dynamisch gebunden.

Ein Web Service ist folgendermaßen aufgebaut:

1. **Kommunikationsprotokoll**: Der meistverwendete Standard ist **SOAP (Simple Object Access Protocol)**, basierend auf HTTP und XML *(W3C Recommendation, 24. Juni 2003)*
2. **Instrumente für die Beschreibung der Services**: Der meistverwendete Standard ist **WSDL (Web Service Description Language)** *(W3C Recommendation 10. Februar 2004)*
3. **Instrumente für die Katalogisierung und Auffindung von Services**: Der meistverwendete Standard ist **UDDI (Universal Descritption Discovery and Integration)**, der ein Registry (indiziertes Archiv von Registrierungen) ist, welcher auf XML basiert und somit plattformunabhängig ist. Er erlaubt das Auffinden (**discovery**) und die Abfrage (**query**) von Services.

Web Services können auf 2 Arten verwendet werden:

1. **Wiederverwendung von Anwendungskomponenten**: Web Services bieten einzelne Funktionen an, die **anderen Anwendungen behilflich** sein können, z.B. Währungsrechner, Übersetzer, Wetter, usw. Somit müssen die Anwendungen diese Funktionen nicht noch einmal implementieren.
2. **Verbinden mit bestehender Software**: Web Services können als **Schnittstelle** gesehen werden **zu den Daten des Anbieters**. Somit werden Probleme in der Interoperabilität behoben werden.

Web Services haben somit folgende Eigenschaften:

- Es sind **Komponenten** einer **Anwendung**
- Kommunizieren über **offene Protokolle (SOAP)**
- Können über das **Web** mittels **UDDI** gefunden werden
- Können von **anderen Anwendungen** verwendet werden
- Verwenden **XML** um die Daten zu kodieren und dekodieren

Web Services können in **verschiedenen Sprachen** geschrieben werden und werden wie jede Web-Anwendung über einen **Web Server** veröffentlicht.

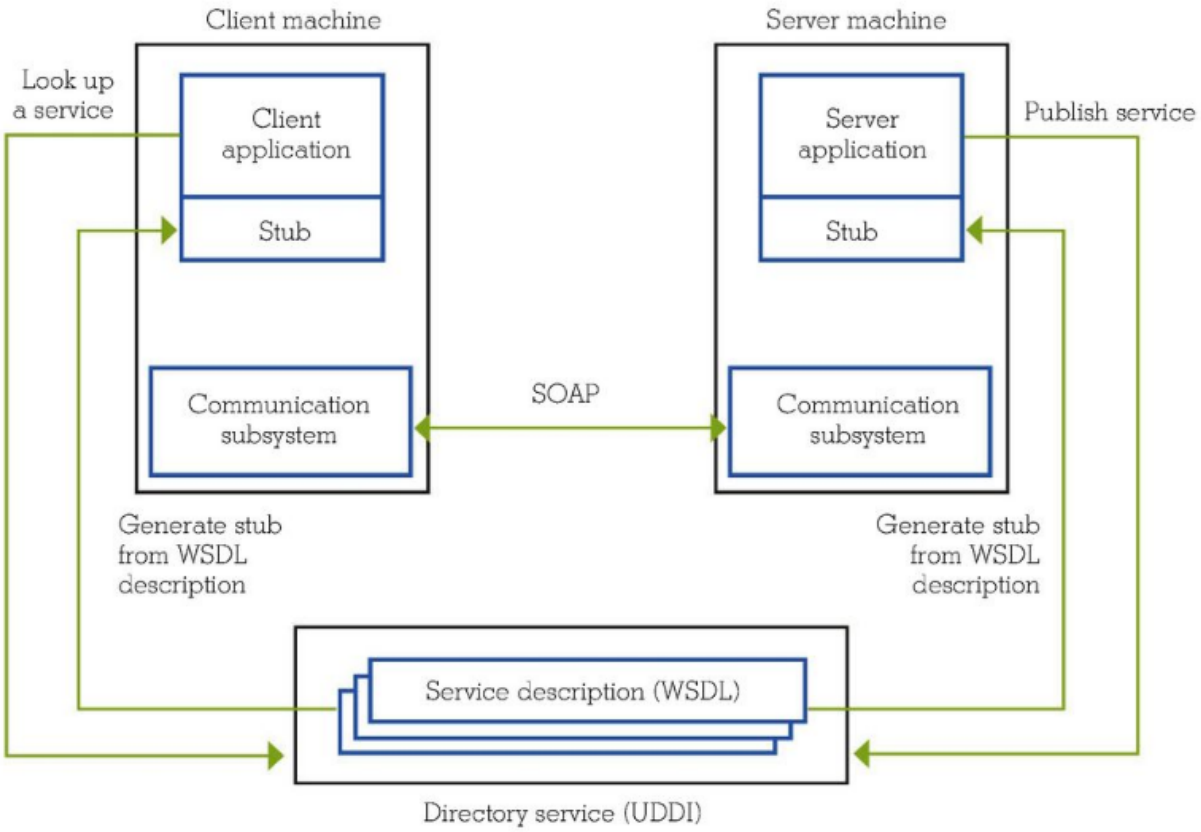


**Abb.1: Komponenten eines Web Services**

## Simple Object Access Protocol (SOAP)

**SOAP** basiert auf **XML**, welches für den **Austausch von Informationen** zwischen Rechner konzipiert wurde. Mit SOAP können Web Services verwendet werden. SOAP ist **plattform- und sprachenunabhängig**. SOAP realisiert **Remote Procedure Calls (RPC)** über Internet. HTTP ist dafür nicht gemacht, da es ein **Sicherheitsrisiko** darstellt und von **Firewalls** geblockt wird. Durch die Verpackung in SOAP können Firewalls **umgangen** werden.

Eine SOAP-Nachricht ist ein normales **XML-Dokument**, welches folgende Elemente enthält:

- **Envelope** (Briefumschlag), der die Nachricht als SOAP-Nachricht identifiziert
- **Header-Element** mit Header-Informationen
- **Body-Element** mit Aufruf und Antwort-Informationen
- **Fault-Element** mit Fehlern und Status-Informationen

Einige wichtige Syntaxregeln:

- Eine SOAP-Nachricht **muss** in **XML** geschrieben sein
- Eine SOAP-Nachricht **muss** den SOAP **Envelope namespace** verwenden
- Eine SOAP-Nachricht **muss** den SOAP **Encoding namespace** verwenden
- Eine SOAP-Nachricht muss **nicht** eine **DTD-Referenz (Document Type Definition)** enthalten
- Eine SOAP-Nachricht muss **nicht XML-Processing-Anweisungen** enthalten

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

**Abb.2: SOAP Grundgerüst**

### SOAP Envelope

Das **äußerte** Element muss das **Envelope-Element** mit ***xmlns:soap*** und ***soap:encodingStyle*** sein.

### SOAP Header

Der **Header** ist **optional**. Wenn ein Header-Element vorhanden ist, muss es da nächste Element sein. Darin sind App-spezifische Informationen vorhanden. Jeder unmittelbar folgende Tag muss namespace-deklariert sein. Die Nachricht kann auf drei verschiedene Weisen gelesen werden:

- ***soap:mustUnderstand***: kann 0 oder 1 sein; wenn 1, dann muss der Empfänger die Nachricht lesen.
- ***soap:actor***: ist eine URI; gibt den Endpoint an, für den diese Nachricht bestimmt ist. Auf dem Weg einer Nachricht kann es mehrere Endpoints geben.
- ***soap:encodingStyle***: ist URI; gibt style-Informationen an und kann überall in der SOAP-Nachricht angegeben werden.

### SOAP Body

Der **Body** enthält die **eigentliche Nachricht**. Direkte Folgetags können namespace-deklariert sein. Tags im Body sind App-spezifisch.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body>
  <m:GetPriceResponse xmlns:m="http://www.w3schools.com/prices">
    <m:Price>1.90</m:Price>
  </m:GetPriceResponse>
</soap:Body>

</soap:Envelope>
```

**Abb.3: SOAP-Beispiel**

Ein Tag im Body kann das **Fault-Element** sein. Darin werden Fehler aufgelistet. Es kann nur einmal in der Nachricht vorkommen.

### HTTP-Binding

SOAP-Nachrichten werden durch einfache **HTTP-Nachrichten** aufgerufen. SOAP hat dafür einen eigenen **MIME-Typ**:

```
POST /item HTTP/1.1
Content-Type: application/soap+xml; charset=utf-8
```

**Abb.4: SOAP Content Type**

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

**Abb.5: SOAP Anfrage**

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

**Abb.6: SOAP Antwort**

## Web Service Description Language (WSDL)

Mit **WSDL** werden **Web Services beschrieben**. Es handelt sich um einfache **XML-Dokumente**.

| ELEMENT | BESCHREIBUNG |
|---|---|
| **<TYPES>** | Ein Container für Definitionen von Datentypen im Web Service |
| **<MESSAGE>** | Eine Typdefinition, der gesendeten Daten |
| **<PORTTYPE>** | Eine Menge von Operationen, die der Endpoint ausführen kann |
| **<BINDING>** | Eine Spezifikation des Protokoll- und Datenformats für einen bestimmten Port-Typ |

```
<definitions>

<types>
  data type definitions........
</types>

<message>
  definition of the data being communicated....
</message>

<portType>
  set of operations......
</portType>

<binding>
  protocol and data format specification....
</binding>

</definitions>
```

Abb.7: Struktur von WSDL

- **WSDL-Ports**: Wichtigstes Element; enthält alle **Funktionen** wie eine **Bibliothek** oder **Klasse**
- **WSDL-Messages**: Definiert die **Daten** einer **Operation**. Jede Message kann aus mehreren Teilen bestehen. Diese Teile können wie **Parameter** bei einem **Funktionsaufruf** verstanden werden
- **WSDL-Types**: Definiert die **Datentypen**, die in einem Web Service verwendet werden.
- **WSDL-Bindings**: Definiert das **Protokoll** und das **Datenformat** für jeden Port-Typ.

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

Abb.8: WSDL Beispiel

In diesem Beispiel ist ***glossaryTerms*** der **Name** des **Ports** und ***getTerm*** die **Operation**. Die einzelnen Operationen sind darunter aufgelistet und weiter ober in den **message-Tags** beschrieben. ***glossaryTerms*** kann als **Bibliothek**, ***getTerm*** ist eine **Funktion** mit ***getTermRequest*** als **input**- und ***getTermResponse*** als **output-Parameter**.

Für den **portType** gibt es mehrere Typen von Operationen:

| TYP | DEFINITION |
|---|---|
| **ONE-WAY** | Die Operation kann Nachrichten erhalten, antwortet aber nicht |
| **REQUEST-RESPONSE** | Die Operation kann Nachrichten erhalten und darauf antworten |
| **SOLICIT-RESPONSE** | Die Operation kann Nachrichten erhalten und wartet auf Antwort |
| **NOTIFICATION** | Die Operation kann Nachrichten erhalten, wartet aber nicht auf Antwort |

### SOAP-Binding

Für die **Anbindung an SOAP** gibt es einen eigenen Tag. Der **binding-Tag** hat zwei **Attribute**: **name** und **type**. Der **Name** kann **beliebig** gewählt werden, der **Typ muss** mit dem **Namen des ports übereinstimmen**.
Im **binding-Tag** enthalten ist der Tag **soap:binding**. Es hat die beiden Attribute **style** und **transport**. **Style** kann **RPC** oder **document** sein, **transport** gibt das verwendete **Protokoll** (z.B. HTTP) an.
Der **operation-Tag** definiert die **Operationen**, die im **portType** möglich sind. Für jede Operation muss es eine entsprechende **SOAP-Operation** geben.

```
<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

**Abb.9: WSDL Binding**

## Java Enterprise Edition platform

**Java EE** basiert auf **Java SE (Standard Edition)** und bietet die Möglichkeit **Multi-tier-Anwendungen** für **Netzwerk-Anwendungen** zu erstellen. Dabei wird eine **Java Virtual Machine (JVM)** aufgesetzt.

### Web Services

- **JAX-WS (Java API for XML Web Services)**: Erlaubt die Erstellung von relativ komplexen Web Services, die XML und SOAP verwenden. Es erlaubt message- oder RPC-orientierte Web Services.
- **JAX-RS RESTful**: Ermöglicht die Erstellung von **Representational State Transfer (RESTful) Web Services**. Es handelt sich um einfache Web Services ohne die Verwendung von XML oder WSDL.
  **REST** bietet sich für **performanceschwache Geräte** wie PDAs oder Handys an. Ein Implementation von JAX-RS ist **Jersey**.

Um das Arbeiten zu erleichtern, verwendet man **IDEs** wie **NetBeans** oder **Eclipse**. Damit lassen sich eigene Java-Web-Server starten und SOAP- und WSDL-Nachrichten automatisch erstellen.

### Webanwendungen

Java bietet verschiedene Möglichkeiten Webanwendungen zu erstellen. Die wichtigsten sind:

- **Servlet**: Java Programme, die dynamisch auf Anfragen reagieren und Antworten schicken können
- **JSP (Java Server Pages)**: Text-Dokumente, die wie Servlets ausgeführt werden.

## Application Server

Für das Ausführen von **Web Applications** braucht es neben einem einfachen **Web Server** auch einen **Application Server**, der sich um die Anwendungen auf dem **Business Tier** kümmert.
Einfach gesagt kann ein Application Server als Erweiterung des Web Servers gesehen werden. Dabei laufen alle Anwendungen in der gleichen Umgebung wie die des Web Servers.

Die Anwendungen müssen dabei für den **richtigen Typ** von **Anwendungsserver** erstellt werden, z.B. Java EE, .NET, SAP Web Application Server. Der Server bietet eine **Laufzeitumgebung** für den **Server-Teil** einer **Client-Server-Anwendung** mit folgenden Eigenschaften:

- **Explizite Eigenschaften**: Kapselung von **Datenquellen**, Zugang zu **Datenbanken** und Schnittstellen zu **höherwertigen Diensten** (**Tier-Prinzip**)
- **Implizite Eigenschaften**: **Skalierbarkeit** ohne die Anwendung modifizieren zu müssen, **Sicherheit**, **Monitoring**, **Logging** und **Kalibrierung** zur **Laufzeit**.

Beispiele von Anwendungsservern:

- Apache Geronimo
- Apache Tomcat
- ColdFusion (ältester Anwendungsserver)
- GlassFish (OpenSource, Java)
- Jetty (Apache, Eclipse)
- SAP NetWeaver Application Server
- WebLogic (Oracle, proprietär)
- WebSphere (IBM, proprietär)

### Java Application Server

Der Server setzt auf einer JVM auf und bietet alle Funktionen von Java EE:

- Servlets
- Java Server Pages (JSP)
- Enterprise JavaBeans

NET-framework verlegt.