

# **F a c h h o c h s c h u l e W e d e l**

## **S e m i n a r a r b e i t**

in der Fachrichtung

Wirtschaftsinformatik

für die Prüfungsleistung

WI-Seminar (Informatik)

## **Web Service Entwicklung mit Java**

Eingereicht von: Sven-Michael Lindow  
Heidbergwinkel 2b  
24558 Henstedt-Ulzburg  
Tel 04193 / 892038

Erarbeitet im: 8. Fachsemester

Abgegeben am: 29. November 2006

Referent: Prof. Dr. Sebastian Iwanowski  
Fachhochschule Wedel  
Feldstraße 143  
22880 Wedel

---

## Inhaltsverzeichnis

Inhaltsverzeichnis.....	II
Abbildungsverzeichnis.....	II
1. Einleitung.....	1
1.1. Web Services Definition.....	1
1.2. Web Services Definition Language (WSDL).....	2
1.3. SOAP.....	4
1.4. Universal Description, Discovery and Integration.....	6
1.5. Representational State Transfer (REST).....	6
2. Web Services mit Java.....	8
2.1. Java Web Service Developer Pack (JWSDP).....	8
2.1.1. JAXP (Processing).....	8
2.1.2. JAXB (Binding).....	8
2.1.3. JAX-RPC.....	10
2.1.4. JAX-WS.....	10
2.1.5. SAAJ.....	11
2.2. AXIS.....	11
3. Öffentliche Web Services nutzen.....	12
3.1. Google Web Service.....	12
3.1.1. Google Java API.....	13
3.1.2. Google mit SOAP.....	14
3.2. Komplexe Web Services.....	17
4. Web Services bereitstellen.....	19
4.1. Entwicklung eines einfachen Web Services.....	19
4.2. Entwicklung mit Netbeans 5.5.....	20
5. Java Web Services im Vergleich zu .NET.....	25
6. Interessante Java-WS Projekte.....	27
6.1. Project Glassfish.....	27
6.2. XINS (XML Interface for Network Services).....	27
6.3. Xfire.....	28
7. Literaturverzeichnis.....	29

## Abbildungsverzeichnis

Abbildung 1: GUI für die Google Web Services.....	13
Abbildung 2: Projekterstellungsdialog.....	20
Abbildung 3: Projekteinstellungen.....	21
Abbildung 4: Neues Java Application Projekt.....	22





## 1. Einleitung

Die Arbeit behandelt das Thema „Web Service Entwicklung mit Java“. Dafür wird einleitend auf die bestehenden Standards eingegangen, um alle Grundbegriffe von Web Services zu erläutern.

Danach erfolgt eine Beschreibung der benötigten Java-APIs, die für die Web Service Entwicklung benötigt werden. Darauf aufbauend werden einige Praxisbeispiele gezeigt und dargestellt wie gut dabei die IDE-Unterstützung in Java ist.

Abschliessend wird ein kurzer Vergleich zur Web Service Entwicklung mit .NET Programmiersprachen gezogen, und es werden interessante Java Open-Source Projekte vorgestellt, die die Web Service Entwicklung weiter vereinfachen sollen.

### 1.1. Web Services Definition

Bei der Suche nach einer eindeutigen Definition von Web Services fällt einem schnell auf, dass es diese so nicht gibt. Jeder Autor oder auch Konsortien drücken sich meist unterschiedlich aus. So wird zwar oft probiert das Gleiche zu beschreiben, doch gelingt es nicht immer gleich gut. Eine etwas genauere Definition ist z.B. diese:

*„WebServices are encapsulated, loosely coupled contracted functions offered via standard protocols. Hereby the term encapsulated means the inability of a user to gather information about the internal service realization. Loosely Coupled refers to the possibility of changing the implementation of one function without requiring changes to clients invoking the function. Encapsulation builds a prerequisite of loosely coupled systems. If a description of a function is publicly available the interface offered by the function to invoke it can be considered to be contracted. This means the behavior of the function and its input and output parameters are fixed. Finally, the usage of standard protocols refers to a technical infrastructure which relies on open, widely published and freely available protocols.“ - WebService.org*

Im Vergleich dazu ist folgende Definition nicht sehr gut, da auf konkrete Implementationen eingegangen wird, die in einer allgemeinen Definition nicht zu finden sein sollten.

*„ [...] Web-Services basieren auf den drei Standards WSDL,*

*SOAP und UDDI: Mit WSDL wird die Schnittstelle eines Web-Services spezifiziert, via SOAP werden Prozedurfernaufrufe übermittelt und mit UDDI, einem zentralen Verzeichnisdienst für angebotene Web Services, können andere Web-Services aufgefunden werden.“ - Arbeitskreis Web Services der Gesellschaft für Informatik*

Diese zweite Definition beschreibt die momentane verbreitetste Welt der Web Services, aber sobald es neue Standards gibt, verliert die Definition an Gültigkeit. Im Folgenden werden die drei eben genannten Standards etwas näher beschrieben, da diese später für die Entwicklung der Web Services mit Java verwendet werden.

## 1.2. Web Services Definition Language (WSDL)

WSDL ist eine XML-basierte Metasprache zur Beschreibung von Web Services. Eine WSDL-Dokument enthält funktionelle Angaben zu der Schnittstelle, dem Zugangsprotokoll und hauptsächlich zu den Operationen, deren Parametern und Rückgabewerten. WSDL ist bisher noch nicht als Standard vom W3C veröffentlicht wurden. Zur Zeit steht die WSDL 2.0 als Draft zur Verfügung, mit dem Ziel danach als Standard veröffentlicht zu werden.

Die Struktur eines WSDL-Dokuments sieht folgendermaßen aus:

```
<definitions>

  <types>
    Definition der verwendeten Datentypen.
  </types>

  <message>
    Definition der verwendeten Nachrichten.
  </message>

  <portType>
    Definition der Operationen.
  </portType>

  <binding>
    Definition des Protokolls für den Web Service.
  </binding>

</definitions>
```

Ein konkretes Beispiel für ein WSDL-Dokument soll die verschiedenen Elemente etwas deutlicher darstellen. Ein einfacher „HalloWelt“-Web Service wird dazu implementiert, auf den später per SOAP zugegriffen wird.

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

<definitions name="HalloWelt"
  targetNamespace="http://localhost/HalloWelt"
  xmlns:tns="http://localhost/HalloWelt"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <message name="halloWeltResponse">
    <part name="Result" type="xsd:string"/>
  </message>

  <portType name="HalloWeltPortType">
    <operation name="halloWelt">
      <output message="tns:halloWeltResponse"/>
    </operation>
  </portType>

  <binding name="HalloWeltBinding" type="tns:HalloWeltPortType">
    <soap:binding
      style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>

    <operation name="halloWelt">
      <soap:operation soapAction="urn:hallowelt#halloWelt"/>
      <output>
        <soap:body
          use="encoded"
          namespace="urn:hallowelt"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
        </output>
      </operation>
    </binding>

  <service name="HalloWeltService">
    <port name="HalloWeltPort" binding="HalloWeltBinding">
      <soap:address location="http://localhost/HalloWelt"/>
    </port>
  </service>
</definitions>
```

### 1.3. SOAP

SOAP ist ein leichtgewichtiges XML-basiertes Protokoll für den Austausch strukturierter und getypter Informationen zwischen Anwendungen in einer dezentralisierten, verteilten Umgebung. Dabei soll durch SOAP kein Programmiermodell oder implementationsspezifische Semantik vorgegeben werden, sondern lediglich ein Mechanismus, um die interne Semantik einer Anwendung zu beschreiben.

Eine SOAP-Nachricht ist nach dem Head-Body-Pattern modelliert. Ein SOAP-Envelope ist ein Container, der ein optionales Header-Element und ein Body-Element enthält. Weiterhin wird hier der verwendete Namensraum festgelegt.

Im Head-Bereich der Nachricht werden die Metainformationen der Nachricht untergebracht. Diese können Informationen über das Routing der Nachricht, über eine eventuelle Verschlüsselung und / oder über die Zugehörigkeit zu einer Transaktion umfassen.

Im Body der Nachricht sind die Nutzdaten untergebracht. Diese Daten müssen vom Empfänger der Nachricht interpretiert werden, mögliche Zwischenstationen können diese auch ignorieren. Die Daten können dabei unter anderem für entfernte Methodenaufrufe (RPC), (Fehler-)Meldungen oder reine Daten stehen.

Anschließend können mögliche Anhänge folgen, diese werden abhängig von dem Transportprotokoll an die Nachricht angehängt. Binärdateien (Sound, Video, Grafik etc.) können durch Nutzung von MIME-Mechanismen angebunden werden.

Als einfaches Beispiel für einen SOAP-Request soll der eben beschriebene „HalloWelt“-Web Service angesprochen werden.

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

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

  <SOAP-ENV:Body>
    <ns1:halloWelt xmlns:ns1="http://localhost" />
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Dieser SOAP-Request ruft per RPC die Methode „halloWelt“ auf. Hierbei handelt es sich um einen Aufruf ohne Parameter, der also einfacher nicht sein könnte. Als Ergebnis wird nur ein String zurück gegeben, wie es in der WSDL-Datei definiert wurde.

Hier zeigt sich, dass das SOAP Protokoll schon bei einer kleinen Anfrage relativ viel Overhead hat. Dies ist auch ein Nachteil von SOAP gegenüber anderen RPC Protokollen, da relativ viel Zeit für das Parsen der Nachrichten verwendet werden muss.

Für ein weiteres Beispiel eines SOAP Requests, der eine SOAP Response erhält, soll der Google Web Service dienen. Google bietet eine Funktion für das Vorschlagen eines Wortes bei Tippfehlern. Dieser Service soll nun genutzt werden.

```
<?xml version='1.0' encoding='UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <S:Body>
    <ns1:doSpellingSuggestion xmlns:ns1="urn:GoogleSearch"
S:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <key xsi:type="xsd:string">google key</key>
      <phrase xsi:type="xsd:string">britney spears</phrase>
    </ns1:doSpellingSuggestion>
  </S:Body>
</S:Envelope>
```

Dieser Request ruft die Methode „doSpellingSuggestion“ auf und übergibt „britney spears“ als Parameter für die Anfrage. Das <key>-Element ist ein Schlüssel der für die Nutzung der Google Web Services benötigt wird.

Als Antwort auf diese Anfrage erhält man folgende SOAP-Response:

```
<?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/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doSpellingSuggestionResponse
xmlns:ns1="urn:GoogleSearch" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:string">britney spears</return>
    </ns1:doSpellingSuggestionResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

## 1.4. Universal Description, Discovery and Integration

UDDI ist ein Verzeichnisdienst für Unternehmen, um deren Web Services zentral verfügbar zu machen. Dabei werden drei Arten von Informationen angeboten: Die „White Pages“, eine Art Telefonbuch, die „Yellow Pages“, die elektronische Entsprechung der gelben Seiten und die „Green Pages“, die Informationen über das Unternehmen und zu den technischen Details der angebotenen Web Services enthalten.

Ein Problem von UDDI ist, dass es nicht so genutzt wird, wie es ursprünglich gehofft wurde. Das liegt vor allem daran, dass die meisten Web Services, die entwickelt werden spezielle B2B Lösungen sind, die nicht der Öffentlichkeit zugänglich gemacht werden sollen, also müssen diese auch in kein Verzeichnis aufgenommen werden.

## 1.5. Representational State Transfer (REST)

Neben SOAP gibt es weitere Alternativen für die Realisierung von Web Services. Roy Fielding hat in seiner Dissertation einen Architekturstil beschrieben, den er REST nennt.

REST basiert auf den Prinzipien des Internets. Das World Wide Web stellt selbst eine gigantische REST Anwendung dar. Viele Suchmaschinen, Shops oder Buchungssysteme sind bereits ohne Absicht als REST basierter Web Service verfügbar.

REST ist kein Standard. REST beschreibt, wie Web Standards in einer Web gerechten Weise eingesetzt werden können. Für die Übertragung wird HTTP eingesetzt, und mittels der vier HTTP-Kommandos GET, POST, PUT und DELETE werden die Web Services verwendet. Für das Prüfen, ob ein Kommando erfolgreich ausgeführt wurde, werden die HTTP-Statusmeldungen verwendet.

Als Beispielanwendung soll ein Onlineshop dienen bei dem Produktdaten abgerufen, hinzugefügt und gelöscht werden und Bestellung getätigt werden. Als erstes wird ein Warenkorb mittels ID angefragt:

```
GET /warenkorb/5873
```

Wie das Ergebnis einer Anfrage repräsentiert wird, ist bei REST nicht spezifiziert. Zwischen Client und Server muss ein gemeinsames Verständnis über die Bedeutung der Repräsentation vorhanden sein. Die Verwendung von XML macht es leicht, die Repräsentation sowohl für Menschen als auch für Maschinen einfach zu gestalten.

```
HTTP/1.1 200 OK Content-Type: text/xml
<?xml version="1.0"?>
<warenkorb xmlns:xlink="http://www.w3.org/1999/xlink">
<kunde xlink:href="http://shop.oio.de/kunde/5873">5873</kunde>
<position nr="1" menge="5">
  <artikel xlink:href="http://shop.de/artikel/4501" nr="4501">
    <beschreibung>Dauerlutscher</beschreibung>
  </artikel>
</position>
<position nr="2" menge="2">
  <artikel xlink:href="http://shop.de/artikel/5860" nr="5860">
    <beschreibung>Earl Grey Tea</beschreibung>
  </artikel>
</position>
</warenkorb>
```

Wie aus dem Ergebnis zu entnehmen ist, enthält der abgefragte Warenkorb 2 Produkte. Auf die Ressourcen der Produkte wird mittels XLink verwiesen, so daß eine einfache Abfrage der Detail-Daten möglich wäre.

Soll nun ein weiterer Artikel hinzugefügt werden müssen die Daten mittels POST gesendet werden:

```
POST /warenkorb/5873
artikelnummer=961
```

Diese Anfrage fügt den Artikel 961 zum bestehenden Warenkorb hinzu.

Das Hinzufügen und Löschen von Daten gestaltet sich ähnlich einfach. Mit PUT können z.B. neue Artikel angelegt werden:

```
PUT /artikel
<artikel>
  <beschreibung-kurz>Rooibusch Tee</beschreibung-kurz>
  <beschreibung>Feiner namibischer Rooibusch Tee</beschreibung>
  <preis>2,80</preis>
  <einheit>100g</einheit>
</artikel>
```

Das Löschen von Artikel funktioniert mit dem Kommando DELETE:

```
DELETE /artikel/6005
```

Im Vergleich zu SOAP verwendet REST mit den URIs einen globalen Adressraum, über den jede Resource adressiert werden kann. Bei SOAP werden die Nachrichten immer an den zentralen Service geschickt und vor dort weiter verteilt.

## 2. Web Services mit Java

Für die Entwicklung von Web Services mit Java werden zusätzliche Java-APIs benötigt. SUN stellt dafür das Java Web Service Developer Pack zur Verfügung, in dem alle benötigten APIs enthalten sind.

### 2.1. Java Web Service Developer Pack (JWSDP)

Im Folgenden werden die wichtigsten APIs kurz vorgestellt.

#### 2.1.1. JAXP (*Processing*)

Das Bearbeiten von XML Dateien kann DOM- oder SAX-basiert oder per XSLT erfolgen.

Beim Document Object Model (DOM) wird die gesamte XML Datei in den Speicher geladen und so kann auf jedes beliebige Element zugegriffen werden. Dabei hat man auch die Möglichkeit mit XPath in der Datei nach bestimmten Attributen zu suchen.

Die Simple API for XML (SAX) erlaubt die ereignisgesteuerte Verarbeitung von XML Dateien. Im Gegensatz zu DOM wird bei SAX die Datei nicht komplett in den Speicher geladen, so dass es sich auch für große Datenmengen problemlos anwenden lässt.

XSLT steht für XSL Transformations und ist Teil der Extensible Stylesheet Language (XSL). XSLT ist eine Programmiersprache zur Transformation von XML-Dokumenten. Es baut auf der logischen Baumstruktur eines XML-Dokumentes auf und erlaubt die Definition von Umwandlungsregeln.

Im späteren Beispiel der Nutzung von Web Services wird XSLT verwendet, um die Antworten in das gewünschte Format zu wandeln.

#### 2.1.2. JAXB (*Binding*)

JAXB ermöglicht das Generieren von Java Klassen und Interfaces aus XML Schema Definitionen. So erhält man eine einfache Möglichkeit auf die Inhalte der XML Dateien zuzugreifen ohne die eigentliche Struktur der Dateien kennen zu müssen.

Wenn man JAXB in seiner Anwendung verwenden möchte, so läuft das in zwei Schritten ab. Zunächst hat man die XML Schema Definitionen, die mittels dem JAXB Compilers in Packages, Klassen und Interfaces umgewandelt werden. Danach muss man die generierten Daten in sein Projekt einbinden und kann mit Hilfe des Bindings auf die XML-Dokumente zugreifen.

JAXB stellt einen einfachen Mechanismus zur Konvertierung zwischen XML und Java zur Verfügung. Ein kurzes Beispiel soll dies verdeutlichen. Dazu wird eine einfache DTD Datei für ein Buch verwendet.

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

<!ELEMENT book (chapter+)>
<!ATTLIST book title CDATA #REQUIRED>
<!ATTLIST book author CDATA #REQUIRED>
<!ATTLIST book pageCount CDATA #IMPLIED>

<!ELEMENT chapter (content?)>
<!ATTLIST chapter caption CDATA #REQUIRED>

<!ELEMENT content (#PCDATA)>
```

Wenn man diese Datei nun an den JAXB Compiler übergibt wird für jedes Element eine Java Klasse erzeugt, so dass man eine Book, Chapter und Content Klasse erhält. Somit hat man nun die Möglichkeit direkt mit den Klassen zu arbeiten. Als nächstes wird jetzt ein Buch in Java erzeugt.

```
Book book = new Book();
book.setTitle("JAXB Sample");
book.setAuthor("Sven Lindow");
book.setPageCount(5);

Chapter chapter = new Chapter();
chapter.setCaption("Introduction");
chapter.setContent("JAXB is a simple.");
book.getChapter().add(chapter);
```

Zum Abschluss soll jetzt das Buch in eine XML-Datei geschrieben werden und danach wieder eingelesen werden.

```
OutputStream os = new FileOutputStream("book.xml");
book.validate(); // optionale Validierung gegen DTD oder Schema
book.marshal(os);
os.close();

InputStream is = new FileInputStream("book.xml");
Book book = Book.unmarshal(is);
is.close();
```

### **2.1.3. JAX-RPC**

Die Java API for XML-Based RPC (JAX-RPC) erlaubt die plattformübergreifende Programmierung von Web Service Clients oder Endpoints basierend auf SOAP 1.1. Für die Beschreibung von Endpoints wird die Web Service Definition Language (WSDL) verwendet.

JAX-RPC wird mit dem Release des Java EE Application Servers „Glassfish“ durch die neue JAX-WS 2.0 Implementation abgelöst. SUN stellt die JAX-RPC API aber weiterhin zur Verfügung, damit alle bestehenden Web Services weiter genutzt werden können, da es einige Zeit dauern wird, bis sich die neue API komplett durchgesetzt hat.

Die Anwendung der API wird bei der Verwendung von Web Services näher beschrieben.

### **2.1.4. JAX-WS**

Die Java API for XML Web Services (JAX-WS) ist eine Neuentwicklung der JAX-RPC API. Es sollte eine einfachere API geschaffen werden, um Web Services bereitzustellen und zu nutzen. Zusammen mit JAXB und SAAJ stellt JAX-WS die Basis für Web Service Programmierung dar. Diese drei APIs stellen ebenfalls die Basis für den Application Server „Glassfish“ dar.

Die APIs wurde dahingehend vereinfacht, dass zuvor überlappende Funktionalität nun klar getrennt wurde. Z.B. waren in JAX-RPC auch Binding-Funktionen enthalten, die nun nur noch in JAXB zu finden sind. Eine weitere wichtige Umstellung ist die Konfiguration mittels Annotations. Zuvor wurde eine Menge von XML-Daten benötigt, um Web Services zu definieren und zu konfigurieren. Dies wurde sehr schnell unübersichtlich und ist nun deutlich einfacher geworden.

Beispiele zur Verwendung und Bereitstellung von Web Services mit JAX-WS folgen in Kapitel 3 und 4.

### 2.1.5. SAAJ

Die SOAP with Attachements API for Java (SAAJ) erlaubt das Versenden von SOAP Nachrichten über das Internet. Ein kurzes Beispiel soll die einfache Anwendung der API verdeutlichen. Dazu wird eine SOAP Nachricht mit nur einem Text Element erstellt.

```
// Getting a Connection
SOAPConnectionFactory factory =
SOAPConnectionFactory.newInstance();
SOAPConnection connection = factory.createConnection();

// Creating a Message
MessageFactory messageFactory = MessageFactory.newInstance();
SOAPMessage message = messageFactory.createMessage();

// Populating a Message
SOAPPart soapPart = message.getSOAPPart();
SOAPEnvelope envelope = soapPart.getEnvelope();
SOAPBody body = envelope.getBody();
SOAPBodyElement bodyElement =
body.addBodyElement( envelope.createName("text", "something",
"http://someurl.com/products"));
bodyElement.addTextNode("some-xml-text");

// Sending a Message
SOAPMessage response = connection.call(message, endpoint);
```

Die erstellte Nachricht sieht nun folgendermaßen aus:

```
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <something:text
xmlns:something="http://someurl.com/products">some-xml-
text</something:text>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

### 2.2. AXIS

AXIS ist eine Implementation des SOAP Protokolls und stellt ein SOAP Framework zur Verfügung. Die AXIS API verbirgt alle SOAP und WSDL Details vor dem Entwickler, so dass sich dieser damit kaum auseinandersetzen muss. Wenn man einen Web Service bereitstellen möchte, kann man diese sehr einfach mit AXIS erledigen und die Anwendung als Tomcat Web Application bereitstellen. Auf Client-Seite nimmt AXIS einem das meiste XML-Handling und vor allem das SOAP-Handling ab, so dass die Entwicklung stark beschleunigt wird.

## 3. Öffentliche Web Services nutzen

Im Folgenden soll die Implementation von Web Services auf Client-Seite näher betrachtet werden.

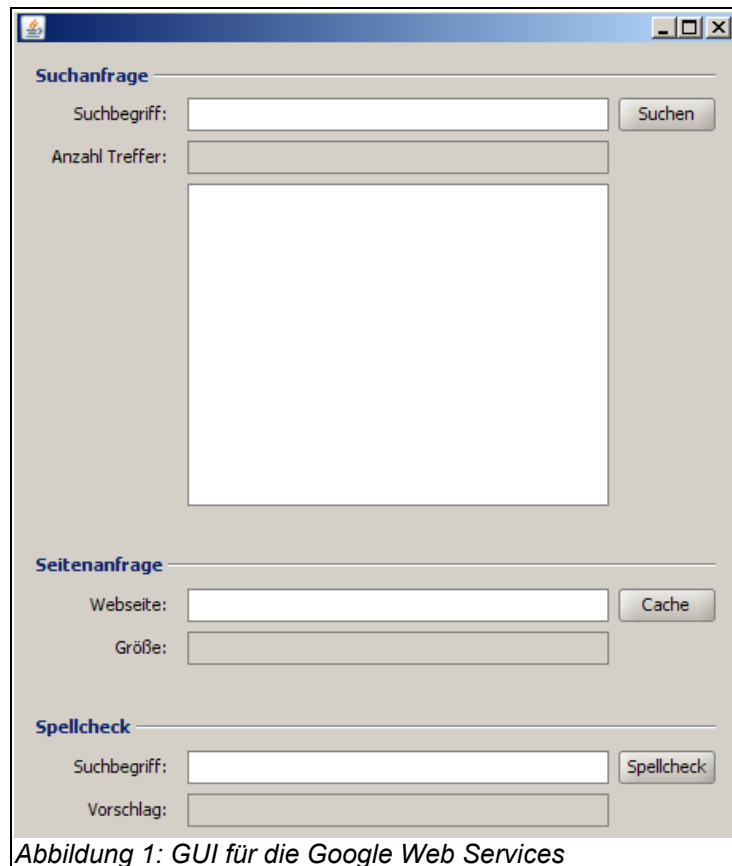
### 3.1. Google Web Service

Google stellt für die Nutzung der Web Services eine Java-API zur Verfügung, die die Entwicklung stark vereinfachen soll. Im Vergleich dazu wird auch gezeigt, wie man mit JAX-RPC oder JAX-WS auf den Google Web Service zugreifen kann.

Google bietet drei verschiedene Web Services an: Das Stellen von Suchanfragen, das Abrufen von gecacheten Webseiten und eine Rechtschreibprüfung. Für diese drei Services wurde eine einfache GUI geschrieben, die für die jeweilige Anfrage einen Parameter entgegennimmt.

Bei einer Suchanfrage wird als Ergebnis die Anzahl der gesamten Treffer und die ersten 10 Suchergebnisse ausgegeben. Die Suche nach einer gecacheten Webseite gibt nur die Größe der Webseite in kb an. Theoretisch könnte man die Webseite mittels eines HTML-Renderers sich anzeigen lassen. Die Antwort ist mittels BASE64 codiert und muss bevor sie ausgegeben werden kann dekodiert werden. Die Rechtschreibprüfung schlägt automatisch ein Wort vor, das gemeint sein könnte. Dies klappt meist nur, sofern man nicht mehr als einen Buchstaben falsch schreibt.

Die Anwendung ist nach dem MVC Muster aufgebaut, und Model und View sind per Observer miteinander verbunden, so daß der View automatisch aktualisiert wird, sobald das Model die Antwort von Google erhält.



### 3.1.1. Google Java API

Die Google API stellt eine komfortable Abstraktion der Web Services zur Verfügung. Dabei muss man sich um keine SOAP Nachrichten kümmern, die im Hintergrund ausgetauscht werden. Für alle Attribute und Operationen stehen Java-Methoden und Klassen zur Verfügung. Eine minimale Suchanfrage gestaltet sich folgendermaßen:

```
// Initialisierung und setzen des Google Developer Keys
GoogleSearch googleSearch = new GoogleSearch();
googleSearch.setKey( _key );

// Übergabe des Suchworts
googleSearch.setQueryString( _searchWord );

// Suche durchführen und gewünschte Ergebnisse speichern
GoogleSearchResult searchResult = googleSearch.doSearch();
_numberResults = searchResult.getEstimatedTotalResultsCount();
_searchResults = searchResult.getResultElements();
```

```
// Observer benachrichtigen
notifySearchObservers();
```

Die anderen beiden Web Services sind genauso einfach zu verwenden. Dabei ist es bei jeder Anfrage erforderlich den Google Developer Key zu übergeben. Der eigentliche Web Service Aufruf besteht dann in beiden Fällen nur noch aus einer Zeile. Nach der jeweiligen Abfrage werden dann wie zuvor die Observer benachrichtigt, um die Ergebnisse anzuzeigen.

```
// Abfrage der Größe einer Webseite
_cachedSize = googleSearch.doGetCachedPage( _website ).length;

// Abfrage für einen Wortvorschlag
_suggestion = googleSearch.doSpellingSuggestion( _spellcheck );
```

### 3.1.2. Google mit SOAP

Die andere Möglichkeit, die Google bietet die Web Services zu nutzen ist das manuelle Senden von SOAP Anfragen. Da Google eine enkodiertes SOAP Nachrichtenformat verwendet, was man normalerweise nicht tun sollte, kann man die WSDL Datei nicht nach Java importieren, so daß man sich nicht die Klassen für den Web Service generieren lassen kann. So muss man selbst die SOAP Antworten parsen, um die gewünschten Ergebnisse zu erhalten. Dazu kann man entweder die SOAP Elemente mittels Java durchlaufen oder man verwendet XSLT, um die Datei zu verarbeiten. In diesem Fall wird XSLT eingesetzt, da es einfach und schnell zu realisieren ist.

Für die Anfragen werden die SOAP Nachrichten nicht dynamisch erstellt sondern aus einer XML-Datei gelesen, um das Beispiel einfacher zu machen. Eine minimal Suchanfrage mit SOAP sieht so aus:

```
<?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/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doGoogleSearch xmlns:ns1="urn:GoogleSearch"
      SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <key xsi:type="xsd:string">googlekey</key>
      <q xsi:type="xsd:string">fh wedel</q>
      <start xsi:type="xsd:int">0</start>
      <maxResults xsi:type="xsd:int">10</maxResults>
```

```

    <filter xsi:type="xsd:boolean">true</filter>
    <restrict xsi:type="xsd:string"></restrict>
    <safeSearch xsi:type="xsd:boolean">false</safeSearch>
    <lr xsi:type="xsd:string"></lr>
    <ie xsi:type="xsd:string">latin1</ie>
    <oe xsi:type="xsd:string">latin1</oe>
  </ns1:doGoogleSearch>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Im Vergleich zur Java API von Google muss man hier mehr Parameter übergeben.

Eigentlich interessiert nur das „q“ Element, das das Sucherwort (Query) beschreibt.

Die Java API vervollständigt die SOAP Nachricht automatisch um die fehlenden Elemente mit den Default-Werten, wie sie bei dieser Anfrage auch verwendet werden.

Eine Nutzung des Service mit JAX-RPC gestaltet sich so:

```

// First create the connection
SOAPConnectionFactory sCF = SOAPConnectionFactory.newInstance();
SOAPConnection connection = sCF.createConnection();

// Next, create the actual message from file
MessageFactory messageFactory = MessageFactory.newInstance();
SOAPMessage message = messageFactory.createMessage( null, new
FileInputStream( soapFile ) );

// Set the destination
URL dest = new URL( "http://api.google.com/search/beta2" );

// Send the message
SOAPMessage reply = connection.call( message, dest );

// Close the connection
connection.close();

```

Im Vergleich dazu ist der Abruf mit JAX-WS ein deutlich anderer, da dort ein neues Konzept bei der Nutzung umgesetzt worden ist. Für jeden Aufruf eines Web Service wird dort ein Dispatcher eingesetzt, der die Verbindung herstellt und auch beendet.

```

// Initialisierung des Dispatchers mit Service, Portname und WSDL
QName serviceName = new QName( "urn:GoogleSearch",
"GoogleSearchService" );
QName portName = new QName( "urn:GoogleSearch",
"GoogleSearchPort" );
Service service = Service.create( new URL(
"http://api.google.com/GoogleSearch.wsdl" ), serviceName );
Dispatch dispatch = service.createDispatch( portName,
SOAPMessage.class, Service.Mode.MESSAGE );

// Erstellen der SOAP Nachricht

```

```

MessageFactory mf = MessageFactory.newInstance();
SOAPMessage req = mf.createMessage( null, new FileInputStream(
soapFile ) );

```

```

// Dispatcher ausführen und SOAP Antwort speichern
SOAPMessage res = (SOAPMessage) dispatch.invoke( req );

```

Die Antwort, die man in beiden Fällen erhält enthält die ersten 10 Suchergebnisse für den Begriff „FH Wedel“. Hier folgt ein kurzer Auszug aus der SOAP Antwort, die den ersten Treffer enthält.

```

<?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/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">

<SOAP-ENV:Body>
<ns1:doGoogleSearchResponse xmlns:ns1="urn:GoogleSearch" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="ns1:GoogleSearchResult">
...
<resultElements
xmlns:ns3="http://schemas.xmlsoap.org/soap/encoding/"
xsi:type="ns3:Array" ns3:arrayType="ns1:ResultElement[10]">
<item xsi:type="ns1:ResultElement">
  <URL xsi:type="xsd:string">http://www.fh-wedel.de/</URL>
  <cachedSize xsi:type="xsd:string">13k</cachedSize>
  <directoryCategory xsi:type="ns1:DirectoryCategory">
    <fullViewableName xsi:type="xsd:string"></fullViewableName>
    <specialEncoding xsi:type="xsd:string"></specialEncoding>
  </directoryCategory>
  <directoryTitle xsi:type="xsd:string"></directoryTitle>
  <hostName xsi:type="xsd:string"></hostName>
  <relatedInformationPresent
xsi:type="xsd:boolean">true</relatedInformationPresent>
  <snippet xsi:type="xsd:string">International, praxisnah und
interdisziplinär studieren - an der Fachhochschule &lt;br&gt;
&lt;b&gt;Wedel&lt;/b&gt;, direkt vor den Toren Hamburgs!
Entscheiden Sie sich für die &lt;b&gt;...&lt;/b&gt;</snippet>
  <summary xsi:type="xsd:string"></summary>
  <title xsi:type="xsd:string">Fachhochschule
&lt;b&gt;Wedel&lt;/b&gt; - Willkommen</title>
</item>
...
</resultElements>
...
</return>
</ns1:doGoogleSearchResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Mittels XSLT wird die SOAP Nachricht so, transformiert, dass nur noch Titel, URL und Kurzbeschreibung in der Textarea der GUI erscheinen.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"
xmlns:ns1="urn:GoogleSearch">

<xsl:output method="text"/>

// Selektiert alle „item“ Elemente
<xsl:template match="/">
  <xsl:apply-templates select="//item"/>
</xsl:template>

// Selektiert bei jedem „item“ Element das Titel, URL und
// Snippet Element. &#10 steht für einen Zeilenumbruch
<xsl:template match="item">
  <xsl:value-of select="title" disable-output-escaping="yes"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="URL"/>
  <xsl:text>&#10;</xsl:text>
  <xsl:value-of select="snippet"/>
  <xsl:text>&#10;&#10;</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

Die Nutzung von JAX-WS im Vergleich zu JAX-RPC ist bei diesem einfachen Beispiel nicht viel kürzer, aber sobald es mehr als ein Aufruf wird, spart man sich u.a. den Verbindungsaufbau und -abbau. Durch die klare Trennung von JAX-WS, JAXB und SAAJ in den neusten Versionen ist die API für den Entwickler einfacher und strukturierter geworden, um Web Services zu nutzen.

## 3.2. Komplexe Web Services

Der Ebay Web Service ist in Vergleich zu Google ein sehr komplexer Web Service. Wenn man sich aus der WSDL Datei die benötigten Java Klassen generieren lässt, so erhält man 777 Klassen. Die Dokumentation für die Ebay API umfasst ca. 1800 Seiten und verdeutlicht damit den großen Funktionsumfang, den Ebay zur Verfügung stellt.

Diese Komplexität stellt auch gleichzeitig das Hauptproblem dar. Wenn man sich aus der WSDL Datei die Klassen generieren läßt so enthalten diese keinerlei Dokumentation. Das bedeutet, gäbe es keine umfangreiche Dokumentation, müßte man alle Informationen aus der WSDL Datei entnehmen. Diese wird aber sehr schnell unübersichtlich, wenn komplexe Web Services bereitgestellt werden. Häufig findet man keinerlei Dokumentation über die zur Verfügung stehenden Web Services, da das Dokumentieren bekanntlich oft als letztes gemacht wird.

Die Verwendung komplexer Web Services stellt sich ebenfalls als problematisch dar, da jeder Web Service in seiner WSDL Datei eine eigene API definiert. Im Zusammenhang mit der fehlenden Dokumentation hat man somit immer eine relativ lange Einarbeitungszeit. Genau aus diesem Grund bieten große Firmen Google oder Ebay eine eigene Java API an, die dokumentiert ist und besser programmiert ist, als die automatisch aus der WSDL Datei erstellte API.

## 4. Web Services bereitstellen

Dieses Kapitel behandelt mit welchen Tools man in Java möglichst einfach und schnell einen Web Service bereitstellen kann. Als Beispiel soll ein Web Service geschrieben werden, der es erlaubt von einer Zahl ihr Quadrat oder ihre Wurzel als Ergebnis zu erhalten.

### 4.1. Entwicklung eines einfachen Web Services

Der Beispiel Web Service wird zunächst nur mit JAX-WS umgesetzt, um die einfache neue API näher darzustellen.

Als erstes wird dafür eine Klasse mit den beiden Methoden erstellt, die als Web Service zur Verfügung gestellt werden sollen. Damit die Klasse automatisch als Web Service erkannt wird, wird zusätzlich die Annotation `@WebService` vor die Klasse geschrieben.

```
package ws;
import javax.jws.WebService;

@WebService
public class NumberFunctions {
    public double getQuadrat(double r) {
        return r * r;
    }

    public double getWurzel(double r) {
        return java.lang.Math.sqrt( r );
    }
}
```

JAX-WS bietet nun die einfache Möglichkeit mit der statischen `publish()` Methode der Endpoint-Klasse den Web Service zu veröffentlichen.

```
public static void main(String[] args) {
    Endpoint.publish(
        "http://localhost:8080/WSEExample/numberfunctions",
        new NumberFunctions());
}
```

Der Quellcode kann nun ganz normal mit `javac` kompiliert werden. Zusätzlich muss man jetzt noch das `Wsgen`-Tool ausführen.

```
> wsgen -cp . ws.NumberFunctions
```

`Wsgen` generiert einige Quelldateien in einem Unterordner „`wsgen`“, die es anschließend automatisch kompiliert. Es werden Stubs erzeugt, die für das Veröffentlichen des Web Services erforderlich sind.

Wenn man die Anwendung nun ausführt startet die Java SE 6 einen kleinen Web Server, der den Web Service an der angegebenen Adresse veröffentlicht. Während die JVM läuft kann man mit einem Aufruf im Browser überprüfen, ob alles geklappt hat.

<http://localhost:8080/WSEExample/numberfunctions?WSDL>

Mit einem Aufruf dieser URL erscheint die automatisch erzeugte WSDL Datei. Das bedeutet, dass der Web Service erfolgreich bereitgestellt wurde.

## 4.2. Entwicklung mit Netbeans 5.5

SUN hat in der neusten Version von Netbeans die Entwicklung von Web Services weiter vereinfacht. Viele Schritte lassen sich bequem mit Hilfe der IDE ausführen, so daß man sich auf die eigentliche Entwicklung konzentrieren kann. Netbeans hat den Web Server Tomcat eingebunden, so daß man per Mausklick die Anwendung im Tomcat bereitstellen kann. Zusätzlich zum Bereitstellen soll mit Hilfe von Netbeans auch ein Client geschrieben werden, der den Service nutzt.

Als erstes muss ein neues Projekt erstellt werden. Hierfür bietet Netbeans einige Templates an, um verschiedene Projekttypen schneller erzeugen zu können. In der Kategorie „Web“ wird eine „Web Application“ als Projekttyp gewählt.

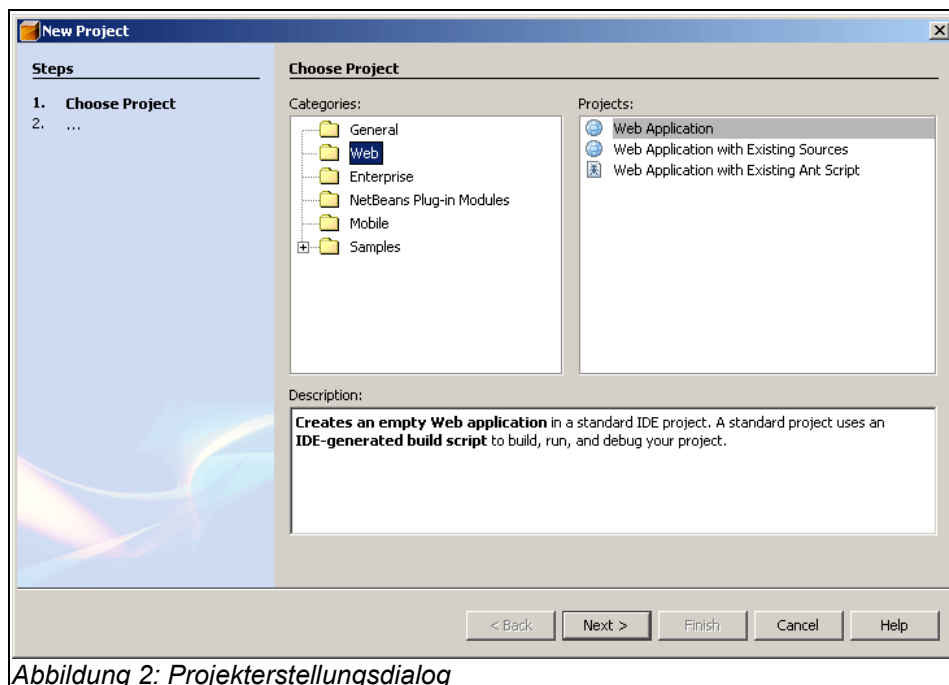
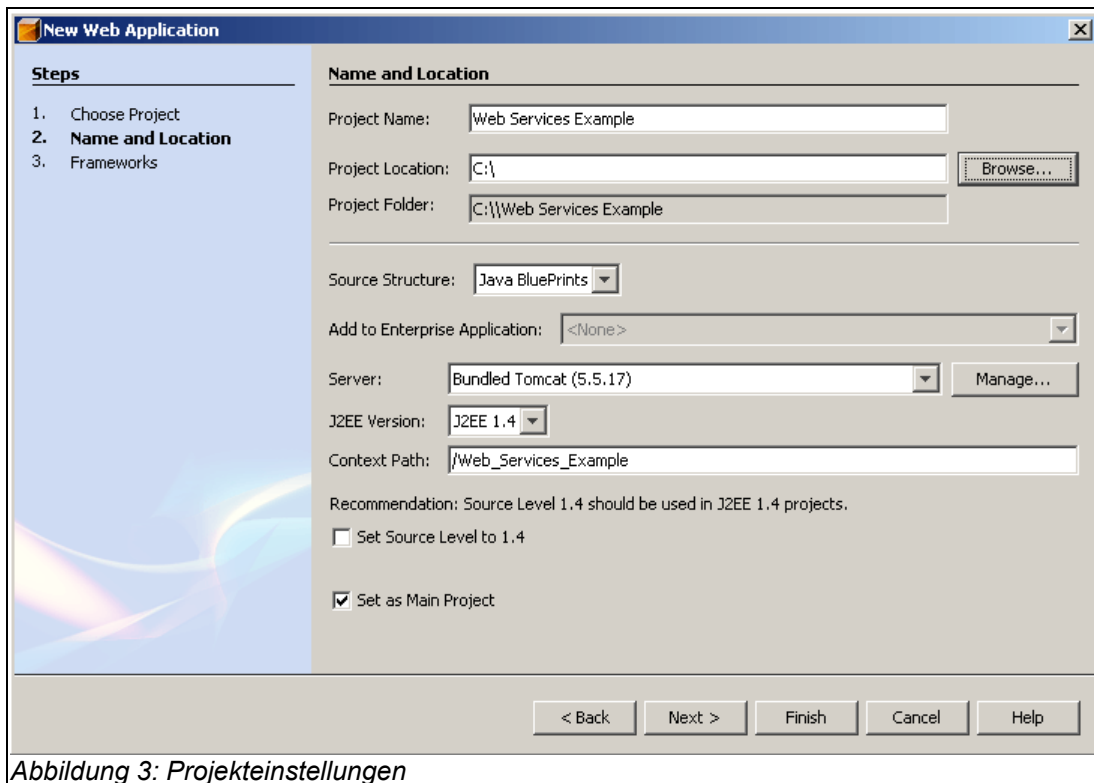


Abbildung 2: Projekterstellungsdialog

Als nächstes können einige Projekteinstellungen vorgenommen werden. Dabei ist darauf zu achten, dass bei dem Punkt „Set Source Level to 1.4“ kein Haken gesetzt ist, da Web Services erst mit Java 1.5 und höher funktionieren.



Nachdem das Projekt erstellt wurde kann man ganz einfach den Web Service hinzufügen. Dazu macht man einen Rechtsklick auf den Projektnamen, wählt „New“ und dann „Web Service“. Dort kann der Name des Web Services „NumberFunctions“ und das Package „ws“ angegeben werden. Danach wird eine leere Web Service Klasse erzeugt, die mit dem gleichen Quellcode wie vorhin zu füllen ist.

```
public double getQuadrat(double r) {  
    return r * r;  
}
```

```
public double getWurzel(double r) {  
    return java.lang.Math.sqrt( r );  
}
```

Abschließend soll der Web Service im Tomcat bereitgestellt werden. Dazu ist ein Rechtsklick auf den Projektnamen und danach „Deploy“ erforderlich. Die Dateien werden automatisch kompiliert und der Web Service im Hintergrund bereitgestellt.

<http://localhost:8084/WSEExample/NumberFunctions>

Um den Web Service zu testen ist ein Aufrufen der genannten URL nötig. Zu beachten ist, dass der Tomcat Server einen anderen Port verwendet, als der Java SE 6 Web Server. Tomcat listet dort auf, welche Web Services bereitgestellt werden und bietet einen Link zu der WSDL-Datei an.

Die Nutzung des Web Services gestaltet sich mit Hilfe von Netbeans genauso einfach, wie das Bereitstellen. Dazu wird zunächst wieder ein neues Projekt erzeugt.

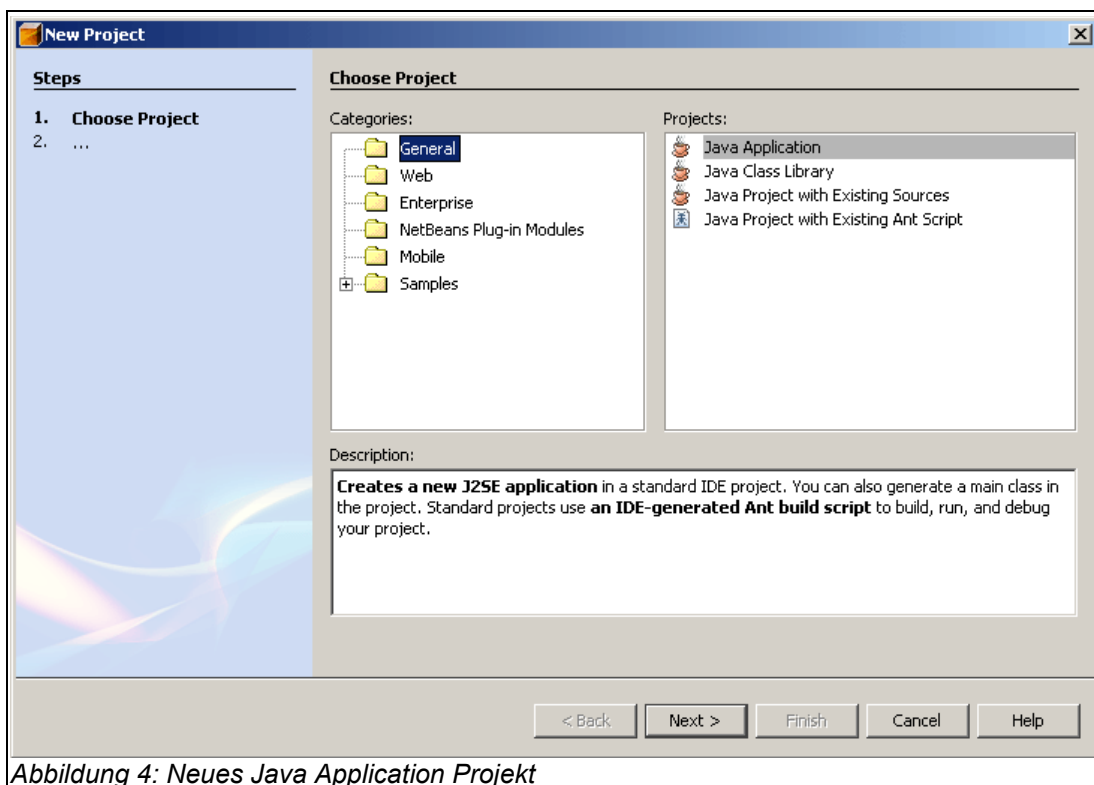
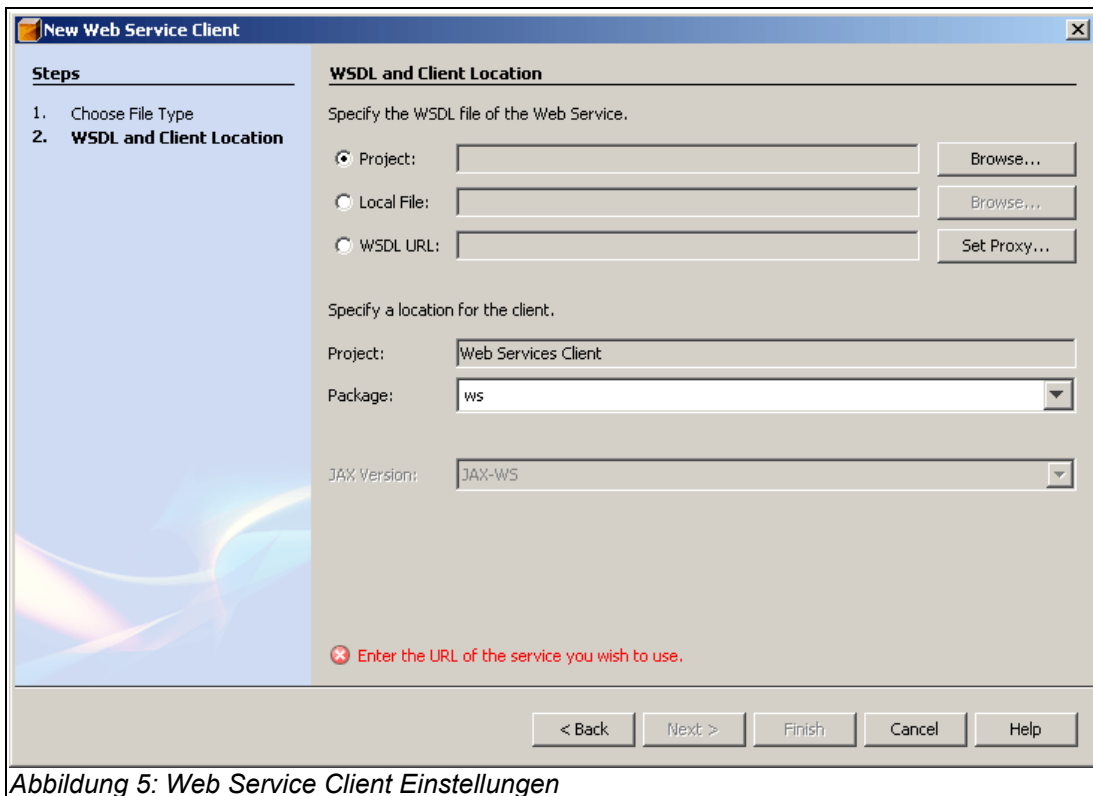


Abbildung 4: Neues Java Application Projekt

Diesmal wird eine einfache Java Application erzeugt, die sich in der Kategorie „General“ befindet. Danach erhält das Projekt einen Namen „Web Services Client“, und das Projekt wird erstellt. Anschließend erfolgt wieder ein Rechtsklick auf den Projektnamen, um den Punkt „New“ und „Web Service Client“ auszuwählen. Es erscheint der Dialog, um alle notwendigen Einstellungen für die Nutzung eines Web Services vorzunehmen.



Für die Nutzung muss eine WSDL Datei angegeben werden. Die kann entweder in einem anderen Projekt vorhanden sein und über das Projekt referenziert werden, als lokale Datei vorliegen oder über eine URL erreichbar sein. In diesem Fall wählt man das eben erstellte Projekt aus und gibt nur noch ein Package an. Danach werden automatisch alle nötigen Klassen für die Nutzung des Web Services aus der WSDL-Datei erstellt. Dies geschieht mit Hilfe des `wsimport`-Tools von JAX-WS.

Die Nutzung anderer Web Services läuft genau nach dem gleichen Muster ab. Man kann eine beliebige WSDL-Datei angeben, und die benötigten Klassen werden erstellt, so daß man anschließend direkt den Web Service nutzen kann.

Damit nun Operationen ausgeführt werden können, muss die `Main.java`, die vorhin automatisch erzeugt wurde, per Doppelklick geöffnet werden. Diese enthält bereits eine `main()`-Methode und soll dort einen Web Service nutzen. Hierzu kann man einfach den Cursor in die `main()`-Methode setzen und per Rechtsklick „Web Service Client Resources“ -> „Call Web Service Operation“ eine der beiden Operationen auswählen, die zur Verfügung stehen. Der automatisch erzeugte Code sieht dann folgendermaßen aus:

```
public static void main(String[] args) {
    try {
        // Call Web Service Operation
        ws.NumberFunctionsService service =
            new ws.NumberFunctionsService();
        ws.NumberFunctions port = service.getNumberFunctionsPort();

        // TODO initialize WS operation arguments here
        double arg0 = 3.0;

        // TODO process result here
        double result = port.getQuadrat(arg0);

        System.out.println("Result = "+result);
    } catch (Exception ex) { }
}
```

Sofern der Web Service im Tomcat noch läuft, kann man diesen jetzt nutzen. Ein Rechtsklick auf den Projektnamen und „Run Project“ führt die Datei aus. Die Netbeans Ausgabe sieht dann so aus:

```
Compiling 1 source file to C:\Web Services Client\build\classes
compile:
run:
Result = 9.0
BUILD SUCCESSFUL (total time: 3 seconds)
```

Diese beiden Beispiele verdeutlichen wie SUN probiert die Entwickler bei der Web Service Programmierung zu unterstützen. In der Vergangenheit gabe es vor allem Beschwerden über die umständliche XML-Konfiguration von JAX-RPC-basierten Web Services. Mit der neuen JAX-WS API bekommt der Entwickler eigentlich nichts mehr davon mit, da er nur noch einige Annotations ein seinen Code einfügen muss. Daher ist die Einführung von JAX-WS ein wichtiger Schritt für die Entwicklung von Web Services mit Java.

## 5. Java Web Services im Vergleich zu .NET

Bei der Entwicklung von Web Services gibt es prinzipiell vier Bereiche, die beachtet werden müssen: Beschreibung, Implementation, Publizieren und Ausführen. Für jeden Aufgabenbereich soll ein kurzer Vergleich der Möglichkeiten von Java und .NET verglichen werden.

Die Beschreibung von Web Services erfolgt in beiden Sprachen grundsätzlich per WSDL. Bei Java Web Services fragt man typischerweise den Endpunkt per UDDI ab, wohingegen bei .NET Web Services ein XML-Namespace in der WSDL-Datei verwendet wird, um den Endpunkt zu definieren. Für die Serverseite stellt .NET eine Komponente zur Verfügung, die ein Mapping zwischen den Operationen aus der WSDL-Datei und COM-Object Methoden erzeugt. Zusätzlich wird ein Web Services Meta Language (WSML)-Dokument erstellt, das für Microsofts SOAP Implementation benötigt wird.

Wie die Implementation von Web Services mit Java funktioniert, wurde bereits ausführlich beschrieben. Bei .NET wird im Vergleich dazu kein nativer Maschinencode erstellt, sondern sogenannter Microsoft Intermediate Language (MSIL)-Code. Dieser Code wird von einem Just in Time (JIT)-Compiler zur Laufzeit in einer virtuellen Maschine (Common Language Runtime (CLR)) ausgeführt. Dies ist eigentlich der gleiche Ablauf, wie bei Java Anwendungen, aber .NET bietet den Vorteil, dass jede .NET Programmiersprache diese Art von Web Services bereitstellen kann.

Das Publizieren von Web Services kann in Java mit Hilfe von UDDI4J getan werden. Dies ist eine Open Source API, die es erlaubt mit einer UDDI-Registry zu kommunizieren. Microsoft hat zunächst versucht Web Services mittels DISCO-Dateien zu veröffentlichen. DISCO-Dateien sind XML-Dateien, die links zu allen nötigen Ressourcen für einen Web Service enthalten. Nachdem UDDI sich verbreitet hat, ist Microsoft dazu übergegangen UDDI zu unterstützen. Dafür gibt es eine UDDI SDK, die in den .NET Umgebungen genutzt werden kann.

Das Verwenden von SOAP-basierten Web Services in Java wurde bereits mittels JAX-RPC und JAX-WS vorgestellt. Bei beiden Möglichkeiten werden Operationen, die in der WSDL-Datei definiert sind per RPC ausgeführt. In .NET muss für den Zugang zu einem Web Service ein Web Service Listener implementiert werden. Die einfachste Methode ist dafür die Verwendung des Microsoft SOAP Toolkits Version 2. Für den Zugriff wird hierfür ein SOAPClient Objekt erzeugt, das u.a. die WSDL-Datei als Parameter enthält. Wenn nun ein SOAP-Aufruf erfolgt, wandelt der SOAPClient automatisch die Anfrage in einen SOAP-Request um und gibt die schon geparsete SOAP-Response zurück.

Dieser kurze Vergleich zeigt einige Unterschiede bei der Programmierung von Web Services auf, die aber nicht ausschlaggebend sind, sich auf eine Sprache festzulegen. Microsoft hat mit der Einführung von .NET bewußt auf Web Service Entwicklung gesetzt, was ein klarer Vorteil ist, da man bequem in jeder .NET Sprache die Programmierung vornehmen kann. SUN hat nun mit JAX-WS in Java 1.6 nachgezogen und einer verbesserte Web Service API zur Verfügung gestellt. Ein weiterer Vorteil von .NET ist die durchgängige Unterstützung in den Microsoft IDEs, da man sich durch den einheitlichen Workflow schnell zurechtfindet. Bei Java ist der Vorteil hingegen, dass es eine Vielzahl von frei erhältlichen Tools gibt, die einem bei der Entwicklung helfen, und es gibt viele frei erhältliche Server, mit denen Web Services bereitgestellt werden können.

Letztendlich werden diese Faktoren aber keine Entscheidung bringen, da meist schon Anwendungen bestehen, die weiterentwickelt werden sollen. In diesem Fall bleibt man bei der verwendeten Sprache, egal ob es .NET oder Java ist.

## 6. Interessante Java-WS Projekte

An dieser Stelle wird kurz auf interessante Java Projekte verwiesen, die die Web Service Entwicklung mit Java weiter voranbringen können.

### 6.1. Project Glassfish

Die Neuentwicklung der Web Service API, die im Kern aus JAX-WS 2.0, JAXB 2.0 und SAAJ 1.3 besteht, stellt einen sehr wichtigen Bestandteil des neuen open-source Application Server von SUN dar. Natürlich ist für einfache und kleine Web Services nicht gleich ein Application Server die richtige Wahl, aber für den Enterprise Bereich ist dieses Projekt eine sehr interessante Alternative zu bekannten Server wie z.B. JBoss.

### 6.2. XINS (XML Interface for Network Services)

XINS ist ein open-source Web Service Framework, dass als Ziel eine einfache Web Service Entwicklung und gute Code-Qualität hat. Im Vergleich zu AXIS wird hier jedoch nicht auf SOAP als RPC Protokoll gesetzt, sondern es wurde ein POX-RPC Protokoll eingesetzt, dass sich am REST Architekturstil orientiert. Anfragen sind nur per HTTP möglich und die Antworten sind in XML. Das Framework setzt auf ein einfaches XML-Format für die Funktionsdefinition auf. Dabei läuft die Entwicklung immer nach dem selben Schema ab: Planung, XML-Definition, Code-Generierung und Programmierung. Aus den erstellten Funktionsdefinitionen wird per Kommandozeile oder auch aus der IDE heraus die Code-Generierung gestartet. Hierbei wird für jede Funktion genau eine Methode erzeugt, für alle Methoden eine HTML und/oder OpenOffice Dokumentation, Unit Tests und Web-Formulare zum Testen der Services erzeugt. Dieses Framework stellt eine gute Alternative zur Web Service Programmierung mit SOAP dar.

### **6.3. Xfire**

Xfire ist eine SOAP-Framework, dass in direkter Konkurrenz zu AXIS steht. Beide Frameworks bieten eine einfache API und unterstützen den Programmier durch Code-Generierung. Xfire bietet aber weitere interessante Features, die es erwähnenswert machen. Es setzt auf einen StAX Parser auf, der eine deutlich bessere Performance bietet, als der DOM Parser von AXIS. Dadurch wird die Latenzzeit und Bearbeitungszeit der Web Service Aufrufe deutlich reduziert. Sehr interessant ist auch die direkte Unterstützung von Spring, so dass bestehende Spring Beans als Service bereitgestellt werden können. Auch dieses Framework stellt eine gute Alternative zur Web Service Programmierung dar.

## 7. Literaturverzeichnis

<http://code.google.com>

<http://java.sun.com>

[http://rmh.blogs.com/weblog/2005/06/jaxrpc\\_is\\_bad\\_b.html](http://rmh.blogs.com/weblog/2005/06/jaxrpc_is_bad_b.html)

[http://weblogs.java.net/blog/kohlert/archive/2006/01/publishing\\_a\\_re.html](http://weblogs.java.net/blog/kohlert/archive/2006/01/publishing_a_re.html)

<http://webservices.xml.com>

<http://ws.apache.org/axis>

<http://www-128.ibm.com/developerworks/edu/ws-dw-ws-soacert1.html>

<http://www-128.ibm.com/developerworks/edu/x-dw-x-ultimashup1.html>

<http://www.deitel.com/WebServices>

<http://www.hpl.hp.com/techreports/2005/HPL-2005-83.pdf>

<http://www.javaworld.com/javaworld/jw-09-2003/jw-0912-webservices.html>

<http://www.jeckle.de>

<http://www.oio.de>

<http://www.service-architecture.com/web-services/articles/index.html>

<http://www.xfront.com/REST-Web-Services.html>

<http://www.xml.com/pub/a/2005/02/16/rest-report.html>

<http://xfire.codehaus.org>

<http://xins.sourceforge.net>

<https://glassfish.dev.java.net>

Nakhimovsky, Alexander; Myers, Tom: Google, Amazon and Beyond: Creating and Consuming Web Services; APress