Aufbau von Web Services

1 Einführung

1.1 Das Prinzip von Web Services

Web Services sind eine Technologie, die mittlerweile eine weite Verbreitung gefunden hat. In erster Annäherung könnte man eine Metapher beanspruchen, um die Anwendungen von Web Services zu umschreiben: Web Services sind für Computer das, was Webseiten für Menschen sind. Präziser ausgedrückt ist mit Hilfe von Web Services eine plattformunabhängige Kommunikation von Computern über das Internet möglich. Ein Anbieter stellt bestimmte Funktionen zur Verfügung, die von einem oder mehreren Nutzern in Anspruch genommen werden. Die Kommunikation zwischen dem Anbieter der Services und deren Nutzern erfolgt über Nachrichten, die auf XML basieren und zumeist mittels HTTP übertragen werden.

Für den praktischen Einsatz von Web Services gibt es zwei grundlegende Szenarien:

- Kommunikation zwischen zwei Computern, speziell wenn die beiden Systeme auf unterschiedlichen Technologien und Plattformen basieren. Dieses Szenario ist vor allem im B2B-Bereich oft zu finden. Zwei Unternehmen möchten Informationen austauschen, zum Beispiel um Bestellvorgänge zu automatisieren. Als Lösung werden Webservices implementiert, die eine plattformunabhängige Rechner-Rechner Kommunikation ermöglichen.

- Ein Informationsanbieter stellt einer größeren Zahl von Nutzern ganz bestimmte Web Services bereit, die im Grunde jedem Rechner den Zugriff auf sein Informationsangebot ermöglichen. Als Beispiel dafür können die Webservices von *amazon* gelten, mit denen etwa Interpreten oder Titel einer CD abgefragt werden können.

Realisiert werden Web Services auf Basis der so genannten service-orientierten Architektur (SOA).

1.2 Die service-orientierte Architektur (SOA)

In den vergangenen Jahren und Jahrzehnten war die Softwareindustrie auf der ständigen Suche nach neuen Techniken und Architekturen, um den steigenden Anforderungen an Softwaresysteme gerecht werden zu können. Dabei kam es

gelegentlich zu einem Paradigmenwechsel. So hat sich etwa die Objektorientierung weitgehend gegenüber prozeduraler Programmierung durchgesetzt. Im Bereich der Applikationsintegration erwies sie sich allerdings als nicht ausreichend. So wurde als neuer Lösungsansatz die service-orientierte Architektur ins Leben gerufen.

Anders als eine klassische Softwarearchitektur, die ein komplettes System beschreibt, beschränkt sich die service-orientierte Architektur auf die Applikationsintegration. Dabei wird ein Service-Layer über die eigentliche Softwarebasis gelegt, der bestimmte Funktionalitäten der vorhandenen Software zur Verfügung stellt, so dass die Funktionalität über das Netzwerk aufgerufen werden kann. Es handelt sich also nicht um eine Entwicklungs- sondern um eine Integrationstechnologie. Das Ziel ist, vorhandene Funktionalitäten weder neu zu entwickeln noch aufwändig integrieren zu müssen, sondern diese über wohldefinierte Schnittstellen einfach ansprechen zu können.

Der Begriff SOA wurde erstmals Ende der 90er Jahre im Zusammenhang mit der Einführung von Jini durch Sun Microsystems verwendet, da Jini nicht nur den Servicebegriff betont, sondern auch ein Konzept für Servicediscovery und Serviceleasing mitbringt. Das ist es auch, was SOA von anderen verteilten Plattformen wie EJB, DCOM oder CORBA unterscheidet, die ja auch in einer gewissen Form Services zur Verfügung stellen.

Jini ermöglicht ein dynamisches Auffinden und Benutzen von Netzwerkdiensten zum Aufbau eines "Network Plug & Play".

So richtig populär wurde der Begriff SOA aber erst, als sich dann Web Services ziemlich rasch in der IT-Welt verbreitet und durchgesetzt haben. Die schnelle Verbreitung der Web Services liegt darin begründet, dass diese erstmals einen vollständigen Satz von Technologien liefern, die eine Implementierung einer service-orientierten Architektur erlauben.

1.3 Grundprinzipien der SOA

Die interne Struktur eines Services ist gekapselt und nach außen nicht sichtbar. Von außen ist nur die Schnittstelle sichtbar. Wichtig bei der service-orientierten Architektur ist, dass Servicebeschreibung und Serviceimplementierung strikt getrennt sind.

- **Servicebeschreibung:** Die Servicebeschreibung besteht im Wesentlichen aus der Beschreibung der Schnittstelle in einer bestimmten Schnittstellen-Beschreibungssprache. Diese Beschreibung definiert, welche Operationen ein Service anbietet und wie sie aufgerufen werden können. Für den Aufrufer sind Details der Implementierung völlig irrelevant. Er muss weder die verwendete Architektur noch die zugrunde liegende Plattform kennen. Für ihn ist nur die definierte Schnittstelle von Bedeutung.

- **Serviceimplementierung:** Die Serviceimplementierung setzt die Schnittstellenbeschreibung konkret um. Die Umsetzung kann in jedweder Technologie erfolgen.

Die Operationen eines Service sind im Vergleich mit den oft vielen kleinen Funktionen eines Objektes meist grob granular.

Eine wichtige Eigenschaft service-orientierter Architektur ist die Möglichkeit, Services zu publizieren und dynamisch zu lokalisieren. Die Beschreibung verwendbarer Services wird in einem zentralen Verzeichnis oder Repository verwaltet. Über definierte Schnittstellen können Services in diesem Verzeichnis publiziert und aufgefunden werden. Ein Service ist im Netzwerk aufrufbar und wird nicht verteilt und lokal installiert. Ein Service ist dabei in ein eigenständiges Modul gekapselt und wird über dieses bereitgestellt. Dabei wird besonderer Wert auf Interoperabilität zwischen den verschiedensten Plattformen und Programmiersprachen gelegt.

Die service-orientierte Architektur basiert auf der Interaktion zwischen drei Rollen: Service Provider, Service Consumer und Service Broker. Es handelt sich dabei um eine lose Kopplung zwischen Provider und Consumer. Die beiden sind nicht fix miteinander verdrahtet, sondern können über den Broker dynamisch zusammengeführt werden.

- **Service Provider:** implementiert ein Service und stellt es im Netzwerk zur Verfügung. Das Service besitzt eine klar definierte Schnittstelle. Optional kann der Service Provider die Beschreibung des Service bei einem oder mehreren Service Brokern anmelden, damit das Service dynamisch von Service Consumern gefunden werden kann.

- **Service Consumer:** der Verwender eines Services. Kann eine Person sein, die mit Browser ein Web Service aufruft.

- **Service Broker:** fungiert als Vermittler zwischen Service Provider und Service Consumer.

1.4 Web Services und SOA

Wenn heute von service-orientierter Architektur gesprochen wird, bezieht sich das meistens auf Web Services. Web Services stützen sich auf drei Technologien: SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) und UDDI (Universal Description, Discovery and Integration). Alle drei Technologien basieren auf XML. Da XML heute auf praktisch allen Plattformen breite Unterstützung findet, ist damit größtmögliche Interoperabilität gewährleistet.
Als Transportprotokoll wird meistens HTTP eingesetzt. Auch dies ist ein weitverbreitetes Protokoll. Außerdem werden dadurch Probleme mit Firewalls vermieden.

2 SOAP

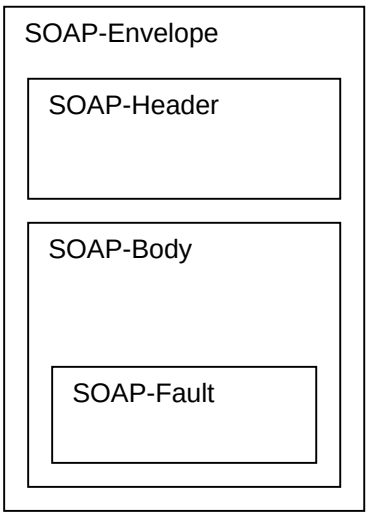
SOAP war ursprünglich eine Abkürzung für *Simple Object Access Protocol*. Da diese Bezeichnung zu Missverständnissen führte, ist SOAP ab 1.2 keine Abkürzung mehr. SOAP ist das Kommunikationsprotokoll für Web Services. Es beschreibt den Aufbau und das Format der Nachrichten, die zwischen Service Provider und Service Consumer ausgetauscht werden. Diese Nachrichten sind XML-Nachrichten.

Mit SOAP ist es möglich, entfernte Prozeduraufrufe (RPC, Remote Procedure Call) zu implementieren. Dies ist eine der häufigsten Verwendungen von SOAP.

SOAP ist kein Transportprotokoll: zum Transport von SOAP-Nachrichten wird ein separates Transportprotokoll benötigt. Meistens kommt dabei HTTP zum Einsatz, je nach Anwendung ist aber auch SMTP oder FTP möglich.

2.1 Der Aufbau einer SOAP-Nachricht

Eine SOAP-Nachricht besteht aus drei Hauptelementen: dem SOAP-Envelope, dem SOAP-Header und dem SOAP-Body.



SOAP-Envelope: dies ist das Wurzelelement einer SOAP-Nachricht und umschließt alle anderen Elemente wie ein Umschlag.

SOAP-Header: optional. Enthält Verwaltungsinformationen bzw. zusätzliche Daten wie zum Beispiel Daten, die für Zwischenstationen (Intermediaries) bestimmt sind.

SOAP-Body: Enthält die eigentlichen Nutzdaten. Bei einem Request sind dies die erforderlichen Parameter, und bei der Antwort die Rückgabewerte.

SOAP-Fault: Fehler. Dieses Element ist nur dann in der Nachricht im Body enthalten, wenn ein Fehler aufgetreten ist.

Eine SOAP-Nachricht könnte also so aussehen:

```
<?xml version=”1.0” encoding=”UTF-8”>
<env:Envelope xmlns:env=”http://schemas.xmlsoap.org/soap/envelope”
   xmlns:xsd=”http://xxx.w3.org/2001/XMLSchema”
   xmlns:xsi=”http://xxx.w3.org/2001/XMLSchema-instance”>

   <env:Header>
      <xy:sessionid xmlns:xy=”http://bei.spiel.at/mein-namespace”>
         4711
      </xy:sessionid>
   </xy:env>

   <env:Body>
      <getBuchTitel xmlns=“”>
         <arg0 xsi:type=“xsd:string“>4235345345</arg0>
      </getBuchTitel>
   </env:Body>

</env:Envelope>
```

2.2 SOAP-Envelope

Der SOAP-Envelope wird durch das Element <*Envelope*> repräsentiert. Der Envelope umschließt die ganze Nachricht. Er ist daher in jeder SOAP-Nachricht genau einmal vorhanden. Im Envelope können Namensräume definiert sein.

2.2.1 encodingStyle

Das Attribut *encodingStyle* definiert die Art der Serialisierung, die für die Nachricht verwendet werden soll. Dieses Attribut kann bei jedem Element verwendet werden. Es bezieht sich auf das Element und alle Kindelemente, die selbst kein *encodingStyle*-Attribut besitzen.

2.3 SOAP-Header

Der SOAP-Header wird durch das <*Header*>-Element repräsentiert. Wenn es vorhanden ist, muss es direkt auf den Envelope folgen.

SOAP-Header werden im Transaktionsmanagement dazu verwendet, Session-Informationen zu speichern. Dies ist insbesondere dann von Bedeutung, wenn der Transport nicht über HTTP, sondern beispielsweise über SMTP erfolgt. In diesem Fall kommt die Antwort nicht im gleichen Aufruf zurück, daher ist eine Session-ID nötig, um die Antwort dem ursprünglichen Request zuordnen zu können.

Wenn eine Authentifizierung verwendet wird, dann ist der Header der geeignete Platz, um die dazu nötigen Daten zu speichern.

2.3.1 mustUnderstand

Wenn der Empfänger den Body eines Requests nicht versteht, muss er die Nachricht zurückweisen und einen entsprechenden SOAP-Fault generieren. Beim Header ist das nicht so: wenn der Empfänger den Header oder Teile davon nicht versteht, könnte er den Request trotzdem verarbeiten und beantworten. Wünscht der Requestor jedoch, dass der gesamte Header verstanden werden muss, so setzt er das Attribut *mustUnderstand* des Headers auf *"1"* (bzw. *"true"*, je nach SOAP-Version). Der Server wird die Nachricht dann zurückweisen, wenn er auch nur einen Teil des Headers nicht versteht.

2.4 Intermediaries

Eine SOAP-Nachricht muss nicht unbedingt direkt zum Empfänger gelangen. Sie kann auch über mehrere zwischengeschaltete Stationen geführt werden. Solche zwischengeschaltete Stationen werden *Intermediaries* genannt. Alle beteiligten Stationen, der Absender, die Intermediaries und der Empfänger, werden als *actor* bezeichnet.
Im SOAP-Header kann definiert werden, dass einzelne Informationen für einen bestimmten Actor (auch Intermediary) bestimmt sind. Als Wert für das Attribut *actor* wird ein URI, der den Knoten spezifiziert, angegeben. Mit einem speziellen URI kann angegeben werden, dass die Daten für den nächsten Intermediary bestimmt sind. Enthält ein Header-Element kein *actor*-Attribut, so bedeutet dies, dass es für den endgültigen Empfänger bestimmt ist.

2.5 SOAP-Body

Der SOAP-Body enthält die eigentlichen Nutzdaten. Er wird durch das Element <*Body*> repräsentiert. Der Body ist ein Kindelement von Envelope und steht direkt nach dem Header-Element, sofern dieses vorhanden ist. Die tatsächlichen Nutzdaten sind Kindelemente des Body, sogenannte Body-Entries. Es dürfen mehrere solche Body-Entries vorhanden sein.

2.6 SOAP-Faults

Kann der Empfänger einen Request nicht verarbeiten, so wird er abgewiesen. Als Antwort wird eine Nachricht mit einem <*Fault*>-Element zurückgeschickt. Es handelt sich um ein Kindelement von Body, ein spezieller Body-Entry also, der nur einmal vorkommen darf.

```
<env:Body>
   <env:Fault>
      <faultcode>Client.Authentication</faultcode>
      <faultstring>Identifizierung fehlgeschlagen</faultstring>
      <faultactor>http://www.infrasoft.at</faultactor>
      <details>
         <!—Spezielle Einzelheiten -->
      </details>
   </env:Fault>
</env:Body>
```

Ein Fault-Element kann folgende Kindelemente enthalten:

- **faultcode:** Der Fehlercode in Form eines durch eine vordefinierte Fehlerkategorie qualifizierten Namens
- **faultstring:** Beschreibung des Fehlers
- **faultactor:** URI des Actors, bei dem der Fehler aufgetreten ist, wenn die Nachricht mehrere Actors passiert hat.
- **detail:** Kann Detailinformationen enthalten.

3 WSDL

Wie bereits erwähnt, stellt Interoperabilität ein wesentliches Designziel von service-orientierter Architektur im Allgemeinen und von Web Services im Speziellen dar. Web Services und deren Clients können auf den unterschiedlichsten Systemen und Plattformen laufen, wie etwa Java, Microsoft .Net, Linux oder sogar Host-Plattformen. Ebenso können die unterschiedlichsten Programmiersprachen verwendet werden, wie Java, C#, Perl oder sogar Cobol.
Damit nun all diese Systeme einander verstehen können, ist es nötig, die Schnittstelle in einer standardisierten Form zu beschreiben. Dieser Standard muss die verfügbaren Methoden eines Service sowie deren Parameter und Rückgabewerte beschreiben. Für diesen Zweck wurde die Web Service Description Language (WSDL) eingeführt, eine auf XML basierende Beschreibungssprache. Eine in WSDL erfolgte Beschreibung der Schnittstelle eines Web Service nennt man WSDL-Dokument.

3.1 Der Aufbau eines WSDL-Dokuments

Ein WSDL-Dokument besteht aus dem Wurzelelement (definitions) und 5 möglichen darin vorkommenden Elementen: *types*, *message*, *portType*, *binding* und *service*. Die 5 Elemente beschreiben die verschiedenen Aspekte eines Web Service.

```
<wsdl:definitions>

   <wsdl:types> ... </wsdl:types>
   <wsdl:message> ... </wsdl:message>
   <wsdl:portType> ...  </wsdl:portType>
   <wsdl:binding> ...   </wsdl:binding>
   <wsdl:service> ...   </wsdl:service>

</wsdl:definitions>
```

Während die Elemente *types*, *message* und *portType* den abstrakten Teil bilden, besteht der konkrete Teil aus den Elementen *binding* und *service*. Der abstrakte Teil beschreibt die Schnittstelle des Service (Operationen, Datentypen) unabhängig vom Transportprotokoll und der konkreten Implementierung. Der konkrete Teil definiert, wie das Web Service erreichbar ist (URI, Protokoll) und wie die Daten serialisiert und codiert werden.

3.2 types

Dieses Element ist optional. Es beschreibt jene Datentypen, die von den Nachrichten verwendet werden. Allerdings müssen nur komplexe, zusammengesetzte Datentypen beschrieben werden, einfache wie *integer*, *float*, *double*, *date* und *string* sind bereits im XML-Schema vordefiniert.

```
<wsdl:types>
   <schema targetNamespace=http://bei.spiel.at>
      <complexType name="Buch">
         <sequence>
            <element name="Titel" type=xsd:string />
            <element name="Preis" type=xsd:float />
         </sequence>
      </complexType>
   </schema>
</wsdl:types>
```

Das Unterelement *schema* kann mehrfach vorkommen.

3.3 message

Mit diesem Element wird eine Nachricht beschrieben, das zwischen Web Service und Client ausgetauscht wird. Das *message*-Element kann mehrfach vorkommen, einmal für jede Nachricht.

```
<wsdl:message name="getBuchTitelAnfrage">
   <wsdl:part name="isbn" type="xsd:string" />
</wsdl:message>

<wsdl:message name="getBuchTitelAntwort">
   <wsdl:part name="Titel" type="xsd:string />
</wsdl:message>
```

Die Beschreibung der Nachrichten beinhaltet auch die verwendeten Datentypen. Dabei kann es sich entweder um einfache Datentypen aus dem XML-Schema handeln, oder um definierte Datentypen aus dem *types*-Element.

Eine Nachricht kann mehrere *parts* enthalten. Bei Requests entspricht ein *part* einem Parameter. Bei Response-Nachrichten entspricht ein *part* dem Rückgabewert oder einem Durchgangsparameter. Die her definierten Nachrichten werden später im *portType*-Element zum Aufbau eines kompletten Requests verwendet.

3.4 portType

Hier werden die Operationen des Web Service definiert, wobei jedes Element *operation* einer aufrufbaren Funktion entspricht.

```
<wsdl:portType name="Buchhandlung">
   <wsdl:operation name="getBuchTitel">
      <wsdl:input message="getBuchTitelAnfrage"
         name="getBuchTitelAnfrage" />
      <wsdl:output message="getBuchTitelAntwort"
         name="getBuchTitelAntwort" />
   </wsdl:operation>
</wsdl:portType>
```

In unserem Beispiel besteht die Operation *getBuchTitel* aus zwei Nachrichten: die Eingabenachricht (input) *getBuchTitelAnfrage* und die Ausgabenachricht (output) *getBuchTitelAntwort*. Das Attribut *message* verweist auf zuvor definierte Einzelnachrichten.

Mit WSDL können vier Aufrufverfahren beschrieben werden, die aus einer oder zwei Einzelnachrichten (input, output) bestehen:

- Request-Response (input, output)
- Solicit-Response (output, input)
- One-Way (input)
- Notification (output)

Das Request-Response-Verfahren (Anfrage-Antwort) implementiert einen entfernten Prozeduraufruf, bei der der Client solange blockiert ist, bis die Antwort zurückgesendet wird. Dies ist der häufigste Anwendungsfall.

Das Solicit-Response-Verfahren beschreitet den umgekehrten Weg: das Web Service beginnt die Kommunikation, der Client antwortet.

Beim One-Way-Verfahren schickt der Client lediglich eine Nachricht, er bekommt keine Nachricht zurück. Für diese Art Kommunikation eignet sich auch SMTP als Transportprotokoll.

Das Notification-Verfahren ist das Gegenteil davon: das Web Service schickt eine Nachricht.

3.5 binding

Dieses Element ist dem konkreten Teil zuzuordnen. Die abstrakte Beschreibung des Web Service wird an das tatsächliche Nachrichtenformat (der Aufbau des SOAP-Body), die Kodierung (Encoding) und das Transportprotokolle gebunden.

```
<wsdl:binding name="BuchhandlungBinding">
   <wsdlsoap:bindng style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
   <wsdl:operation name="getBuchtitel">
      <wsdlsoap:operation soapAction="" />

      <wsdl:input name="getBuchhandlungAnfrage">
         <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.og/soap/encoding/"
            namespace="http://bei.spiel.at" use="encoded" />
      <wsdl:input>

      <wsdl:output name="getBuchhandlungAntwort">
         <wsdlsoap:body
            encodingStyle="http://schemas.xmlsoap.og/soap/encoding/"
            namespace="http://bei.spiel.at" use="encoded" />
      <wsdl:output>

   </wsdl:operation>
</wsdl:binding>
```

3.6 service

Dieses ebenfalls dem konkreten Teil zuzurechnende Element definierte die tatsächlichen Netzwerkadressen, an die sich der Client wenden muss.

```
<wsdl:service name=”BuchhandlungService“>
   <wsdl:port binding=“BuchhandlungSoapBinding“ name=“Buchhandlung“>
      <wsdlsoap:address location=”http://bei.spiel.at:8080/Buchh“ />
   </wsdl:port>
</wsdl:service>
```

Es können mehrere *port*-Element angegeben werden, wobei jedes eine Netzwerkadresse spezifiziert. Dadurch können mehrere Netzwerkadressen, unter denen das Web Service erreichbar ist, angegeben werden.

4 UDDI

Universal Description, Discovery and Integration ist ein Verzeichnisdienst, der, vor allem im Business-To-Business (B2B) Bereich, dazu dienen soll, Unternehmen und die von ihnen angebotenen Dienste zu finden. UDDI stellt neben SOAP und WSDL die dritte Säule der Web Services dar.

UDDI ist selbst eine Web Service Anwendung und benutzt SOAP als Protokoll.

Ein UDDI-Registry bietet folgende Funktionalität:

- **White Pages:** Namensregister, sortiert nach Namen, enthält alle Anbieter von Services sowie sämtliche Kontaktinformationen (Telefon, Fax, ...)
- **Yellow Pages:** Branchenverzeichnis. Kategorisiert Unternehmen und die von ihnen publizierten Services, sodass diese nach ihren Kategorien gesucht werden können (wie die gelben Seiten im Telefonbuch)
- **Green Pages:** Informationen über die technischen Details (WSDL) der angebotenen Services

4.1 Die Datenstrukturen

UDDI verwendet fünf Datenstrukturen zur Beschreibung: *businessEntity*, *businessService*, *bindingTemplate*, *publisherAssertion* und *tModel*.

- **businessEntity:** repräsentiert eine Organisation. Enthält eine Liste von *businessServices*, das sind die von dieser Organisation publizierten Services
- **businessService:** stellt ein Service dar. Ist ein Kindelement von *businessEntity*.
- **bindingTemplate:** beschreibt die technischen Details zum Aufruf des Service. Ist ein Kindelement von *businessService*.
- **publisherAssertion:** da große Unternehmen schlecht durch einzelne *businessEntities* dargestellt werden können, werden mit *publisherAssertions* Beziehungen zwischen Organisationen dargestellt (z.B. Mutter- und Tochtergesellschaft). Beziehungen werden erst sichtbar, wenn beide die gleiche Information publiziert haben. Dies verhindert die einseitige Registrierung von falschen Beziehungen.
- **tModel:** kann einerseits eine technische Spezifikation darstellen (Ablegen eines WSDL-Dokuments), andererseits dient es zur Kategorisierung von Datenstrukturen.

4.2 UDDI API

Das UDDI API lässt sich in zwei Gruppen einteilen: das Inquiry-API und das Publisher-API.

- **Inquiry-API:** dieses API dient zum Suchen und Abfragen von Informationen. Es enthält Requests wie:
  - find_business
  - find_service
  - find_binding
  - find_tModel
  - get_businessDetail
  - get_serviceDetail
  - get_bindingDetail
  - get_tModelDetail
- **Publisher-API:** dient zum Publizieren, d.h. zum Anlegen, Ändern und Löschen von Informationen. Das Publisher-API steht nur Service Providern zur Verfügung. Es enthält Requests wie:
  - save_business
  - save_service
  - save_binding
  - save_tModel
  - delete_business
  - delete_service
  - delete_binding
  - delete_tModel

5 Werkzeuge für die Entwicklung

5.1 Web Services mit Java

Es gibt eine Reihe von Entwicklungswerkzeuge zum Entwickeln von Web Services für Java wie etwa das J2EE Java Web Services Developer Pack (Java WSDP).

Das zurzeit wohl am weitesten verbreitete Werkzeug ist Apache Axis. Es ist dies eine Open Source Implementierung des Web Service Standards SOAP unter der Lizenz der Apache Software Foundation.

Da das Schreiben von Web Services von Hand nicht ganz einfach ist – man muss eine Menge Code sowie komplizierte Deployment Descriptoren schreiben – bietet Axis hierfür zwei Generatoren an:

- **Java2WSDL:** dient zur Generierung einer WSDL aus Java-Code

- **WSDL2Java:** erzeugt Java-Code sowie einen WSDD (Web Service Deployment Descriptor) aus einer WSDL

Um möglichst wenig Code zu schreiben, sei folgende Vorgangsweise empfohlen:

- Man schreibe leere Funktionsrümpfe
- Mit Java2WSDL ein WSDL-Dokument generieren
- Mit WSDL2Java aus dem WSDL-Dokument den Java-Code erzeugen. Es entstehen folgende Dateien:
    - XX.java – die Interfacedefinition
    - XXService.java – repräsentiert das Service (am Client)
    - XXServiceLocator.java – zum Auffinden des Service am Client
    - XXSoapBindingImpl.java – die Implementation des Web Service; hier Code einfügen
    - XXSoapBindingStub.java – der Client Stub
    - deploy.wsdd – der Deployment Descriptor zum Deployen am Server
    - undeploy.wsdd – zum Undeployen

Mit dieser Vorgangsweise ist es nur mehr nötig, den Code des Web Service selbst zu schreiben, d.h. man kann sich auf die Anwendung konzentrieren.

5.2 Web Services mit Microsoft .NET

Mit dem .NET Framework ist es sehr einfach, ein Web Service zu schreiben. Man muss nicht einmal die grundlegenden Technologien wie SOAP, WSDL usw. verstehen, um ein einfaches Web Service zu implementieren.

Mit Microsoft Visual Studio .NET geht es sogar noch einfacher: Man öffnet ein C# oder Visual Basic ASP.NET Web Service Projekt und bekommt eine Schablone, in der man nur mehr seinen Code eintragen muss.

5.3 Weitere Entwicklungswerkzeuge

Es gibt eine Vielzahl weiterer Entwicklungswerkzeuge für alle möglichen Plattformen und Programmiersprachen. Diese sind in einer ständigen Entwicklung begriffen. Dem interessierten Leser wird empfohlen, sich gezielt bei den verschiedenen Anbietern von Entwicklungswerkzeugen über den aktuellen Stand ihres Angebots zu informieren.

Ergänzend zu diesem Text steht Ihnen auf der InfraSoft Website ein Glossar zur Verfügung, in dem Sie die meisten der hier verwendeten Begriffe finden. Über das aktuelle Angebot an weiteren, kostenlosen Fachbeiträgen zur Softwareentwicklung informieren Sie sich bitte unter http://www.infrasoft.at/service.

Harald Pichler
Wien, im Februar 2005

Der Autor ist Mitarbeiter der InfraSoft, einem Unternehmen, das auf komplexe Softwareentwicklungen spezialisiert ist. Die Experten der InfraSoft haben langjährige Erfahrungen in der Entwicklung und verfügen über fundierte Kenntnisse in Design, Analyse, Realisierung, Test und Projektmanagement.
Für **individuelle Beratungen** zur Entwicklung von Softwarelösungen und die Bereitstellung von **Realisierungsteams** wenden Sie sich bitte an info@infrasoft.at.