Toolgestützte Entwicklung von Web Services

Erstellung von Web Services mit Werkzeugen von IBM und BEA

Florian Hawlitzek
Hawlitzek IT-Consulting GmbH

Florian Hawlitzek ist Geschäftsführer der Firma Hawlitzek IT-Consulting GmbH in München. Der Java-Spezialist ist als Consultant und Trainer tätig. Seine Schwerpunkte liegen im Bereich Java Enterprise Connectivity und Application Server, im besonderen EJBs und WebSphere.

Zusätzlich hat er mehrere Bücher und Fachartikel veröffentlicht und ist als Referent auf zahlreichen Fachkonferenzen aktiv. Bereits 1997 hat er seinen ersten Java-Kurs entwickelt. Aufgrund des großen Erfolges entstanden zahlreiche Schulungen und Beratungsaufträge, bei denen Herr Hawlitzek gewinnbringende Synergien zwischen vielen Unternehmen aufbauen konnte.
Unter www.hawlitzek.de finden Sie u.a. Infos zu seinen Büchern, die von ihm veröffentlichten Fachartikel und weitere Konferenzvorträge.

Web Services sind der neue Hype am Softwarehimmel. Endlich kann man verschiedenartigste Umgebungen standardisiert koppeln. Leider ist es aber nicht ganz einfach, die Protokolle zu implementieren und die Konfigurationsdateien richtig bereitzustellen. Wohl dem, der ein Tool besitzt, das all dies erledigt.

Einsatz und Bewertung

Web Services sind als Integrationsplattform vielseitig einsetzbar: Zur Kopplung verschiedener Webanwendungen im Inter- und Intranet, zur Einbindung von Applikationsservern, ERP- und Legacy-Systemen sowie zur Verbindung von J2EE- und .NET-Architekturen. Besonders ASPs und Anbieter virtueller Marktplätze freuen sich über standardisierte Schnittstellen a la SOAP, WSDL, UDDI & Co., da hierüber eine Vielzahl von Kunden ohne großen Anpassungsaufwand angesprochen werden können.

Die Beschreibung der Web Services ist sowohl technisch als auch fachlich, auch an Suchmechanismen wurde gedacht. Damit kann jedes interessierte Unternehmen einen Dienst in einer UDDI-Registry suchen und dessen Beschreibung als WSDL-Dokument abrufen. Auf Basis dieser XML-Beschreibung kann nun ein neuer Client oder ein konkurrierender Service realisiert werden. Die Implementierung kann in einer beliebigen Programmiersprache erfolgen, zum Beispiel Java oder Visual Basic .NET.

Die Vorteile der Web Services liegen also klar auf der Hand und das erklärt auch den augenblicklichen Boom. Aber es gibt auch einige Nachteile: So sind die entsprechenden Schnittstellen noch recht jung und daraus ergeben sich die üblichen zwei Probleme: es gibt noch kaum etablierte Standarddienste und die APIs ändern sich noch häufig. Zum Beispiel ist in den Registries noch kaum Metainformation über die Dienste abgelegt wie z. B. zu deren Preisen oder Abrechnung. Auch in den Bereichen Security und Abläufe/Prozesse tut sich noch viel.

Mit Web Services beschäftigen sich mehrere Vorträge dieser Konferenz, so dass hier auf eine tiefergehende Einführung verzichtet werden kann. Deshalb nur ganz wenige Worte zu Technik: Ein Web Service ist ein Dienst, dessen Schnittstelle standardisiert beschrieben und in der Regel in einem öffentlichen oder markt-/firmeninternen Verzeichnis registriert ist (z. B. in den UDDI-Registries von IBM oder Microsoft). Der Dienst selbst ähnelt einem Java-Servlet, das unter einer URL ansprechbar, in der Regel mit Parametern aufrufbar und Bestandteil einer größeren Anwendung ist. Im Unterschied zum Servlet liefert der Web Service jedoch keine Ausgabe für einen Browser zurück (z. B. eine HTML-Seite), sondern eine XML-Datenstuktur. Der Aufrufer ist also kein Mensch, sondern ein Programm.

Zum Aufruf eines Web Services und zur Rückübertragung der Ergebnisse wird das SOAP-Protokoll benutzt (Simple Object Access Protocol), das alle Daten als XML-Dokument kodiert. Die Beschreibung des Dienstes erfolgt ebenfalls als XML-Dokument, das in der Web Services Description Language (WSDL) abgefasst ist und in einer UDDI-Registry (Universal Description Discovery & Integration) abgelegt wird.

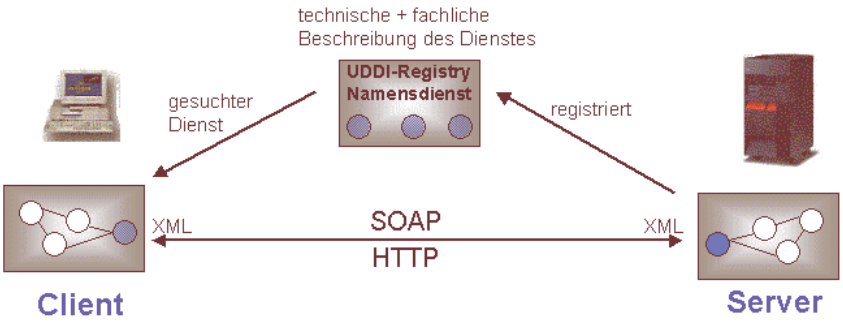


Abbildung 1: Registrierung und Nutzung eines Web Services

Plattform

Die erste grundsätzliche Entscheidung ist die für die technische Basis. Hier gibt zwei große Konkurrenten, denen die Analysten etwa dieselben Marktanteile vorhersagen: *Microsoft .NET* und *J2EE*. In der Microsoft-Welt ist man weitgehend auf Windows festgelegt, hat aber die Auswahl verschiedener Programmiersprachen wie *Visual Basic .NET, C#, C++* oder *Delphi*. Das andere Lager bilden die Java-Verfechter rund um *IBM, Sun, BEA, Apache* und Co. Hier ist schon eine Vielzahl von Produkten auf dem Markt und die Integration von Web Services in die *Java 2 Enterprise Edition* schreitet mit großen Schritten voran. Für welche Plattform man sich nun entscheidet ist für die Entwicklung und Toolauswahl von ausschlaggebender Bedeutung, für die Kopplung der Dienste spielt sie dank der standardisierten Protokolle aber keine Rolle mehr. Wir beschränken uns im Folgenden auf Java-Produkte.

Produkte

Mittlerweile unterstützen fast alle J2EE-Server den Einsatz von Web Services, die ja auch fester Bestandteil der nächsten J2EE-Version 1.4 sein werden. In der Art der Unterstützung der eingesetzten API-Versionen gibt es aber noch große Unterschiede. Beispiele für Java-Applikationsserver mit Web-Service-Support sind *IONA E2A Application Server J2EE Technology Edition, IBM WebSphere, BEA WebLogic, Borland Enterprise Server, SilverStream eXtend Application Server, Oracle 9i* oder *SonicXQ*.

Zur Entwicklung wird eine Vielzahl an Tools angeboten. Viele davon basieren auf einfachen Kommandozeilentools und Library-Sammlung, die auch oft die Basis der Laufzeitumgebungen kommerzieller Server darstellen. Als erstes ist das Open-Source-Projekt *Apache SOAP* und dessen Nachfolger *Axis* zu nennen, der bereits die neusten API-Versionen implementiert. IBMs *Web Services Toolkit* (*WSTK*) enthält *Axis* und bietet weitere Tools rund um WSDL und UDDI sowie ausführliche Dokumentationen, Tutorials und Beispiele. Das *Java Web Services Dev. Pack* (*JWSDP*) von *Sun Microsystems* beinhaltet eine Sammlung der verschiedener XML-APIs sowie Betaversionen von *Tomcat* und der JSP Tag Library.

Eine kurze Historie: 1998/99 gab die ersten Ideen und Vorläufer zu Web Services, die von Microsoft aufgegriffen und etwas später gemeinsam mit IBM weiterentwickelt wurden. Das erste Toolkit, das SOAP 1.1 unterstützte, brachte *IBM* Anfang 2000 als *SOAP4J* heraus. Dieses wurde dann zur Weiterentwicklung an die *Apache Foundation* übergeben. Im Gegenzug wurden die neueren Versionen der *Apache*-Projekte *SOAP* (1.1) und *Axis* (SOAP 1.2) in *IBMs WSTK* übernommen. Microsoft brachte auch mehrere Versionen ihres SOAP-Toolkits heraus und arbeitete intensiv an der Interoperabilität mit der *IBM/Apache*-Umgebung. Die MS-SOAP-Implementierung ist eine wichtige Basis für Microsofts .NET-Architektur. Schließlich kam auch die Java-Community mit ins Boot und erstellte im Rahmen der Java XML-APIs *JAX-RPX* (JSR101), *JAXM* (JSR-067) und *JAXR* Schnittstellen zur *Java 2 Enterprise Edition* (J2EE).

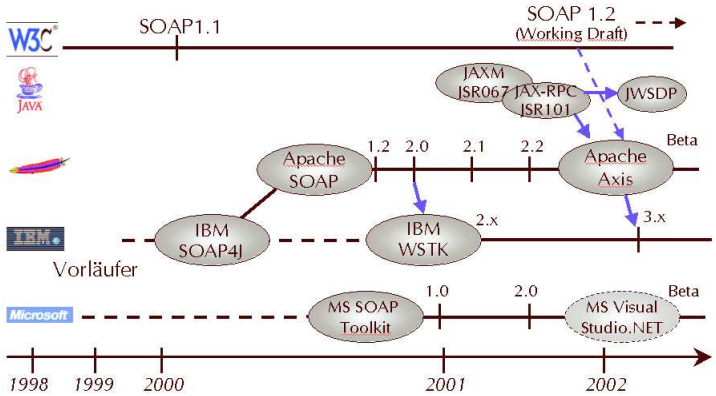


Abbildung 2: Entwicklung der SOAP-APIs und Toolkits

Bei den integrierten Entwicklungsumgebung ist die Unterstützung von Web Services ein wichtiges Verkaufsargument. So reihen sich z. B. *IBM WebSphere Studio, Borland Delphi, Kylix, JBuilder, BEA Workshop (Cajun), XML Spy, Microsoft Visual Studio .NET, Oracle 9i Developer, SilverStream eXtend Workbench* und *Together ControlCenter* in Phalanx der IDEs ein. Wir werden uns hier exemplarisch etwas näher mit den recht unterschiedlichen Tools der beiden Marktführer bei den Application Servern, *BEA* und *IBM* beschäftigen.

IBM WebSphere Studio Application Developer

Der *Application Developer* von IBM kam Ende 2001 auf den Markt und tritt in vielen Bereichen die Nachfolge *von VisualAge for Java* und dem „alten" *WebSphere Studio* an. Er umfasst Werkzeuge rund um die Entwicklung von J2EE-Anwendungen (EJB, JSP, Servlets, ...), Weboberflächen, XML und WebServices. Die wizard-gestützte Entwicklung von Fatclients mit dem Visual Composition Editor fehlt allerdings gegenüber dem bisherigen VisualAge und die Anbindung von Host- und ERP-Systemen mittels Connectoren (J2EE Connector Architecture) ist nur in der teuren Integration Edition enthalten.

Der *Application Developer* setzt auf einer offenen Workbench auf, die *IBM* im Rahmen des *Eclipse*-Projektes als Open Source freigegeben hat. Die einzelnen Werkzeuge sind darin als Plug-Ins realisiert. *IBM* hofft mit dieser Öffnung neben *Rational, Instantiations* oder *Merant* noch viele weitere Partner zu finden, die *Eclipse* eine ähnliche Verbreitung wie *Microsofts Visual Studio* bereiten soll. Im Unterschied zu *VisualAge* setzt *WSAD* nicht mehr auf einem zentralen Code-Repository auf, sondern arbeitet datei-basiert.

Entwicklung von Web Services in WSAD

Voraussetzung zur Entwicklung eines Web Services ist ein sogenanntes Web Project. Darunter gibt es die Verzeichnisse „source" und „webApplication". In letzterem befindet sich dann später der zu deployende Code und die Konfigurationsdateien (siehe Abbildung 3).

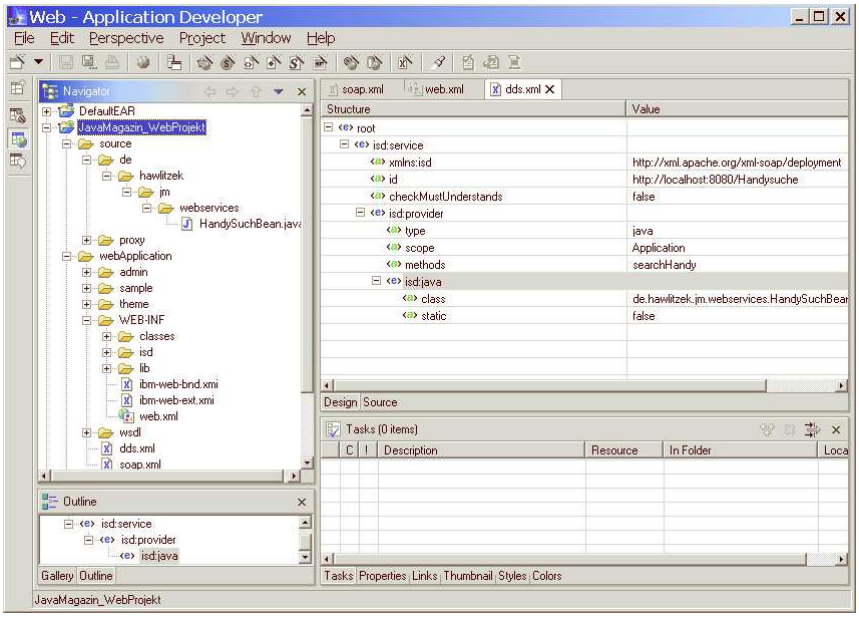


Abbildung 3: Web-Perspektive des Beispielprojektes

Häufig entwickelt man den Web Service nicht vollkommen neu, sondern setzt auf vorhandenem Code auf. *WSAD* erlaubt neben der Neuimplementierung das Wrappen eines Web Services um eine JavaBean, eine EJB oder Datenbankanweisungen (in der

Integration Edition auch um einen Connector-Aufruf). In unserem Beispiel möchten wir eine Art Shopsuche für Handys anbieten. Der Nutzer, z.B. ein Händler oder ein Portal gibt ein bestimmtes Handy per Marke und Modell an und unserer Service liefert die URL mit der Handy-Detailinfo bzw. Bestellseite zurück.

Wir bauen hier auf einer JavaBean auf, die in einer Datenbank mit Handydaten die passende URL heraussucht (*de.hawlitzek.jm.webservices.HandySuchBean.java*). Im Wizard geben wir nun als Web Service Type „Java bean Web service“ an und wählen unsere Bean aus, die wir zuvor im source-Verzeichnis des Projektes abgelegt hatten. Auf der nächsten Seite des Wizards können wir einige Angaben zu den Konfigrationsdateien machen. Hier gibt man die Namen und Pfade der WSDL-Files an: das Binding-Dokument beinhaltet später die Schnittstellen- und Protokollbeschreibung unseres Dienstes, das Service-Dokument die Adressen und Namensräume. Ebenfalls auf dieser Seite spezifizieren wir die Ziel-URL unseres Dienstes und im Falle von JavaBean den Scope, in dem die JavaBean erzeugt werden soll, z.B. pro Anfrage, nur einmal usw.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HandySuchBeanRemoteInterface"
 targetNamespace="http://www.hawlitzek.de/definitions/HandySuchBeanRemoteInterface"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:tns="http://www.hawlitzek.de/definitions/HandySuchBeanRemoteInterface"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <message name="searchHandyRequest">
    <part name="marke" type="xsd:string"/>
    <part name="modell" type="xsd:string"/>
  </message>
  <message name="searchHandyResponse">
    <part name="result" type="xsd:string"/>
  </message>
  <portType name="HandySuchBeanJavaPortType">
    <operation name="searchHandy">
      <input name="searchHandyRequest" message="tns:searchHandyRequest"/>
      <output name="searchHandyResponse" message="tns:searchHandyResponse"/>
    </operation>
  </portType>
  <binding name="HandySuchBeanBinding" type="tns:HandySuchBeanJavaPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="searchHandy">
      <soap:operation soapAction="" style="rpc"/>
      <input name="searchHandyRequest">
        <soap:body use="encoded"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="http://www.hawlitzek.de:8080/Handysuche"/>
      </input>
      <output name="searchHandyResponse">
        <soap:body use="encoded"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
         namespace="http://www.hawlitzek.de:8080/Handysuche"/>
      </output>
    </operation>
  </binding>
</definitions>
```

Listing 1: WSDL-Binding-Datei

Im nächsten Schritt können wir nun die einzelnen Methoden unserer JavaBean (oder EJB) selektieren, die wir als Dienste mappen möchten. Wir wählen die Methode *searchHandy(String marke, String modell)*. Hierbei gibt es verschiedene Möglichkeiten, die Parameter und Rückgabewerte in die SOAP-XML-Form zu bringen, so dass einfache Datentypen ohne viel Aufwand und XML-Parsing eingesetzt werden können, aber auch komplexe, benutzerdefinierte Mappings möglich sind.

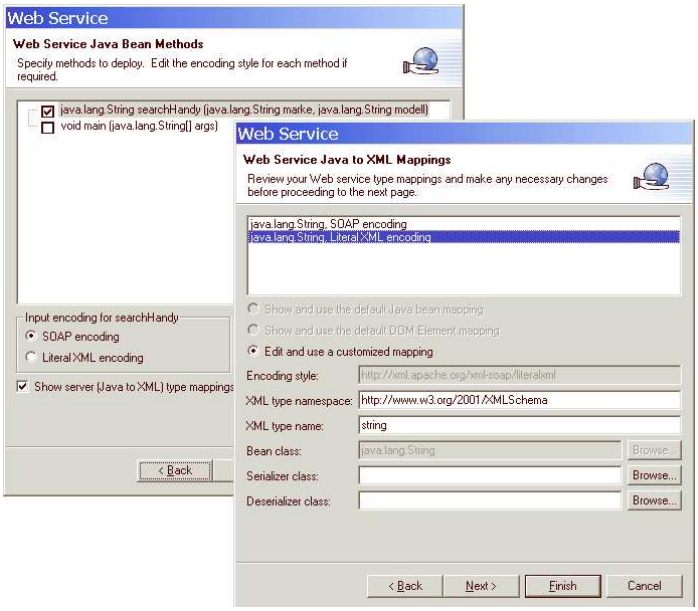


Abbildung 4: Mapping der Java-Datentypen nach XML

Schließlich können wir uns noch einen Proxy und einen Testclient generieren, mit dem wir unseren Web Service auch ohne Client-Anwendung und noch vor der UDDI-Registrierung als Servlet/JSP-Anwendung ausprobieren können.

Was wurde nun erzeugt:

- zwei WSDL-Dateien (Binding und Service)
- der SOAP Deployment Deskriptor (dds.xml)
- eine Webanwendung zum Testen des Dienstes
- eine Administrationsanwendung zum Starten und Stoppen des Web Services
- ein Client-Proxy, der den programmatischen Zugriff auf den Dienst vereinfacht

Abbildung 5: Generierter Testclient

```
<root>
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
id="http://www.hawlitzek.de:8080/Handysuche" checkMustUnderstands="false">
  <isd:provider type="java" scope="Application" methods="searchHandy">
    <isd:java class="de.hawlitzek.jm.webservices.HandySuchBean" static="false"/>
  </isd:provider>
</isd:service>
</root>
```

Listing 2: SOAP Deployment Deskriptor

Test

WSAD bietet für den Test eine Serverumgebung auf Basis des Applikationsservers *WebSphere Advanced Single Server 4.0*. Für den Test können wir die Standardkonfiguration in der *WSAD*-Server-Perspektive weitgehend übernehmen. Nur auf der Seite „Data source“ tragen wir noch unsere JDBC-Datenquelle mit der Handy-Datenbank ein, auf die die HandySuchBean zugreift.

Zunächst starten wir den Web Service. Dazu wechseln wir wieder in die Web-Perspektive, wählen in unserem Projekt, die Datei *webApplication\admin\index.html* und rufen „Run on Server“ im Kontextmenü auf. Dies startet automatisch die Testumgebung. Mit der Adminoberfläche können wir nun unseren Dienst auswählen und starten.

Abbildung 6: Generierte Adminoberfläche

Analog starten wir die generierte JSP-Beispielanwendung unter *webApplication\sample\TestClient.jsp.*

Und so sieht ein Paar von SOAP-Nachrichten aus, die wir über den Testclient an unseren Server geschickt haben:

Request:

```
<?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:Body>
<ns1:searchHandy xmlns:ns1="http://www.hawlitzek.de:8080/Handysuche"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<marke xsi:type="xsd:string">Nokia</marke>
<modell xsi:type="xsd:string">6310</modell>
</ns1:searchHandy>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response:

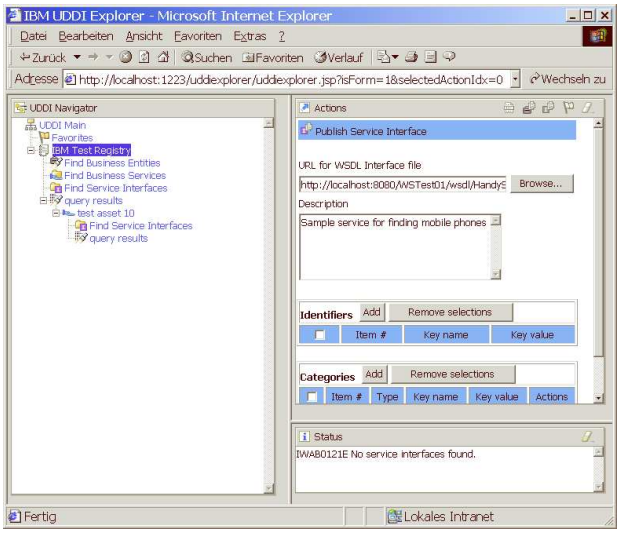
```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<SOAP-ENV:Body>
<ns1:searchHandyResponse xmlns:ns1="http://www.hawlitzek.de:8080/Handysuche"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<return xsi:type="xsd:string">http://www.hawlitzek.de:8080/servlet/
 de.hawlitzek.schulung.servlet.HandyAuswahlServlet?marke=Nokia&amp;modell=6310</return>
</ns1:searchHandyResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 3: SOAP Request und Reponse

Deployment und Registrierung

Nachdem der lokale Test erfolgreich verlaufen ist, können wir den Web Service nun für den Betrieb fertig stellen. Dazu müssen die WSDL-Dateien so überarbeiten, dass die Pfade auf unseren Produktionsserver zeigen. Hierbei hilft der eingebaute XML-Editor.

Für das Deployment der Webanwendung als J2EE-EAR-Datei wird auch noch das Konfigurationsfile *web.xml* benötigt, das *WSAD* ebenfalls generiert hat. Wichtig für unseren Server ist das darin registrierte Servlet namens rpcrouter. Dabei handelt es sich um die *IBM/Apache*-SOAP-Implementierung, die eingehende Requests an die Web Services weiterleitet. Danach exportieren wir die gesamte Anwendung als EAR-Datei und spielen sie im Server ein.



Für die Registrierung des Dienstes in eine UDDI-Registry rufen wir „Export | UDDI“ und starten damit den *IBM UDDI Explorer*. In diesem Tool können Sie beliebige Registry kontaktieren, z. B. eine private, die Business Registries von *IBM* und *Microsoft* oder eine der Test Registries. Voreingestellt ist die IBM Test Registry, in der jede seine Dienste beliebiger Art kostenlos anmelden kann. Wir wählen unsere WSDL-Datei aus und können nun noch eine Beschreibung und eine Kategorisierung hinzufügen.

Abbildung 7: Registrierung in einer UDDI-Registry

BEA WebLogic Workshop

Einen ganz anderen Ansatz stellt *BEAs WebLogic Workshop* dar (frührer Codename: *Cajun*). Diese Entwicklungsumgebung will keine universelle J2EE-Entwicklungsumgebung für Profis sein, sondern ist ein Tool, mit dem auch Entwickler mit wenig Java-Knowhow Web Services aus Komponenten zusammensetzen, deployen und testen können. Mit einen grafischen Editor stellt der Entwickler die Schnittstellen zum Client zusammen und bindet fertige Komponenten („Controls“) wie z. B. andere Web Services, EJB-, Datenbank- oder JMS-Zugriffe in den Workflow ein.

Anders als bei *WSAD* wird hier nicht ein Web Service rund um eine bestehende Code-Einheit generiert, sondern dieser komplett neu als Java-Klasse „komponiert“. Wählt man nicht explizit die WSDL-Datei aus, sieht man praktisch nicht von den unterliegenden Technologien SOAP und WSDL. Um die einzelnen Schritte miteinander zu verknüpfen, ist natürlich doch etwas Java-Wissen notwendig. Insbesondere sollte man die Framework-Klassen des Workshops kennen, die die Basis für den grafischen Editor bilden. Leider basiert der generierte Code nicht direkt auf den Standardklassen von SOAP, so dass der erzeugte Dienst ausschließlich auf *WebLogic 7* läuft.

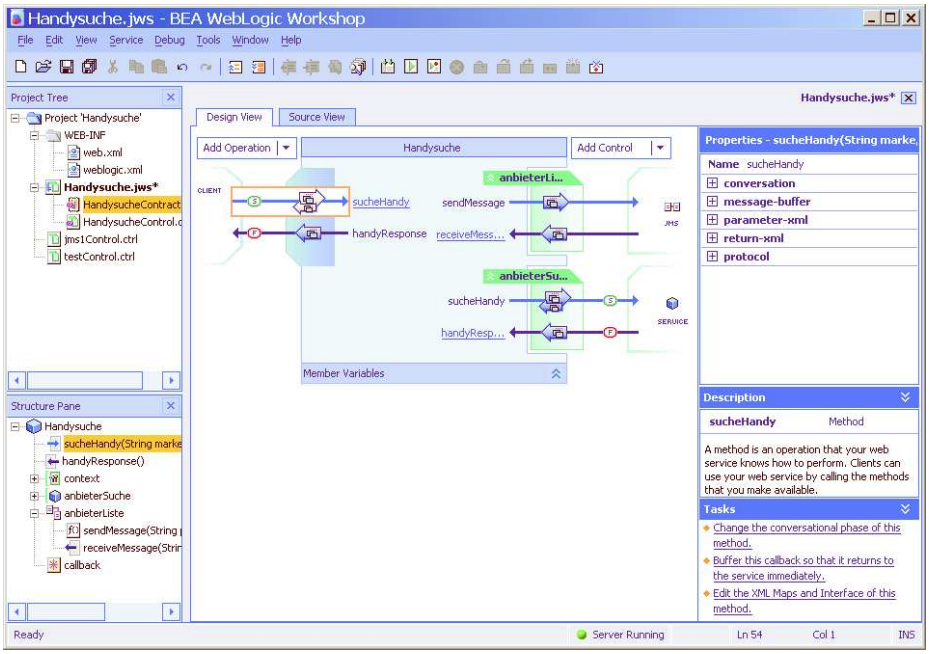


Abbildung 8: Design View von WebLogic Workshop

Das Deployment auf den *WebLogic Server* 7.0 geht per Knopfdruck ohne irgendeine manuelle Konfiguration. In Abbildung 9 sehen Sie den Testclient. Ebenfalls enthalten ist eine Webanwendung für das Suchen und Registrieren von Web Services in einer UDDI-Registry.

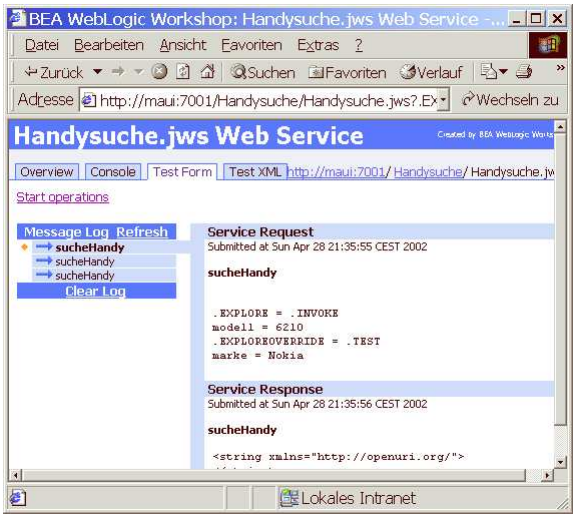


Abbildung 9: Test eines Web Services in WebLogic Workshop

Fazit

Die Entwicklung von Web Services ist kein leichtes Unterfangen. Die vorgestellten Werkzeuge helfen bei der Erstellung der teilweise kryptischen Konfigurationsdateien und generieren praktische Testanwendungen. Das *BEA*-Tool unterstützt die neuesten Standards und ist sehr leicht zu bedienen. Allerdings ist die Erweiterbarkeit und Flexibilität etwas eingeschränkt, zudem laufen die generierten Web Services nur auf dem hauseigenen Server. Das *IBM*-Tool etwas schwieriger zu bedienen und basiert noch auf etwas älteren APIs. *WSAD* ist jedoch weitaus mächtiger und durch die den Open-Source-Framework *eclipse* beinahe unbegrenzt erweiterbar. So fügt beispielsweise die Integration Edition die Möglichkeit hinzu, Backendaufrufe wie z.B. Hostprogramme auf Basis der J2EE Connector Architecture als Web Services zur Verfügung zu stellen.