Seminararbeit

von
Moritz Steiner
aus
Mannheim

vorgelegt am

Institut AIFB
Prof. Dr. H. Schmeck
Universität Karlruhe

Lehrstuhl für Praktische Informatik IV
Prof. Dr. W. Effelsberg
Universität Mannheim

Juli 2004

Betreuer:
Dipl.-Wi.-Inf. Jürgen Vogel
Dipl.-Wi.-Ing. Bernd Scheuermann

Gliederung

Einleitung

Im Internet werden ständig neue Web-Dienstleistungen und modulare Anwendungen veröffentlicht. Organisationen beschreiben diese und bieten sie in Verzeichnissen als so genannte Web Services an. Die Beschreibung beinhaltet den Anbieter und den Namen der Dienstleistung mit der Beschreibung des Zugriffsprotokolls zu solchen Dienstleistungen.
Die exakte Definition für Web Services ist nach wie vor stark diskutiert. Als minimale Übereinstimmung verschiedener Definitionen kann man sagen, dass es sich bei Web Services um jegliche Art von Softwarekomponenten handelt, die die Fähigkeit besitzen, ihre Funktionalität als so genannte Dienste oder Services im Internet dynamisch zu erzeugen, ausfindig zu machen und über standardisierte Internet-Protokolle (HTTP, XML) bereitzustellen.
Um den Web Services eine robuste und sichere Internet-Anwendung anzubieten, wurden die drei wichtigsten Standards entwickelt: UDDI, WSDL und SOAP [5] [8].

- SOAP (Simple Object Access Protocol)
  Aufrufprotokoll und Nachrichtenaustausch
- WSDL (Web Service Definition Language)
  Schnittstellenbeschreibung
- UDDI (Universal Description, Discovery and Integration)
  universeller Verzeichnisdienst für Dienstleistungen

SOAP ist ein plattformunabhängiges XML-basiertes Protokoll, über das verschiedene Anwendungen per HTTP miteinander kommunizieren. SOAP legt fest, wie ein komplexes XML-Dokument in HTML verpackt und versandt wird. SOAP ist ein Standard, der es ermöglicht XML-Messages zwischen unterschiedlichen Plattformen und aus Programmiersprachen auszutauschen.
WSDL beschreibt netzwerkbasierte XML-Dienste. Sie definiert das Format von Objekten, unabhängig davon, welches Protokoll (SOAP, XML) oder welche Codierung (MIME) diese systemabhängig verwenden.
UDDI ist eine XML-basierte Datenbank mit einem Verzeichnis von E-Business-Unternehmen, ihren Produkten und Dienstleistungen, die außerdem Informationen zu den jeweils bevorzugten Standards für automatisierte Geschäftsabläufe enthält.

Die Aufgabestellung dieser Seminararbeit lautet „WebServices mit Java“, es soll dargestellt werden, wie unter Zuhilfenahme von Java, WebServices bereitgestellt und abgefragt werden können. Das Thema wurde von den Vortragenden Anelia Mircheva und Moritz Steiner in zwei Gebiete aufgeteilt [14]. Frau Mircheva ist im ersten Teil des Vortrages auf das Java Web Services Developer Pack (JWSDP) [18] von Sun eingegangen, das u.a. aus den APIs DOM, SAX, JAXB, JAXP, JAXR, JAX-RPC und SAAJ besteht.
Der zweite Teil des Vortrages, den diese Arbeit vertiefen soll, geht auf AXIS [2][9] ein, erklärt jedoch die Architektur nicht genauer. Es geht vielmehr darum aufzuzeigen, wie einfach die Implementierung von Web Services mit Hilfe dieses Frameworks ist. Zunächst wird AXIS allgemein vorgestellt, im Anschluss wird, anhand eines

durchgängigen Beispiels, vorgeführt wie die Server- und Clientseite zu implementieren und zu installieren sind.
Des Weiteren wird ein Vergleich zwischen .Net [1][15][16] und Java [17] gezogen. Es werden nicht nur die Fähigkeiten im Umgang mit Web Services verglichen, sondern die Plattformen werden auch generell gegenübergestellt.

Axis Allgemein

Axis [2] (Apache eXtensible Interaction System) ist eine SOAP Engine – ein Framework für die Konstruktion von SOAP Prozessoren, Clients, Servern, Gateways usw. Die aktuelle Version ist in Java geschrieben; die C++ Version ist (fast) brauchbar, zumindest zum Testen, für die Client-Seite. AXIS ermöglicht es, sowohl die Server- als auch die Clientseite eines Web Services sehr einfach zu implementieren. Die in der Einleitung benannten APIs müssen für die Anwendung nicht bekannt sein.

AXIS ist mehr als SOAP, es umfasst auch:

- einen standalone Server
- einen Server, der in Servlet Engines eingebaut werden kann, beispielsweise in Tomcat (in den folgenden Beispielen wird eine Installation mit Tomcat benutzt)
- Unterstützung für die *Web Service Description Language* (WSDL) [5][6][8]
- Werkzeuge, mit deren Hilfe Java Klassen aus WSDL generiert werden können
- Beispielprogramme
- Verschiedene Werkzeuge – beispielsweise den TCP Monitor

AXIS gilt als Dritt-Generation SOAP (bei IBM hieß Axis SOAP4J). Gegen Ende 2000 wurde in den SOAP v2 Gremien diskutiert, wie die SOAP Engines [3] flexibler, leichter zu konfigurieren und einfacher zu handhaben gemacht werden könnten. Zudem sollte die Architektur so gewählt werden, dass neben SOAP auch weitere XML Protokolle von W3C unterstützt werden könnten.
Da die ersten Versionen von SOAP Engines nur mit großem Aufwand zu warten waren, sollte von Grund auf neu entworfen und entwickelt werden.

Nach vielen Diskussionen in verschiedenen Ausschüssen einigte man sich auf folgende Kerneigenschaften:

- **Geschwindigkeit**:

AXIS verwendet SAX, einen ereignisgesteuerten XML Parser. Dieser ist schneller als der, in älteren Versionen verwendete, DOM Parser.

- **Flexibilität**:

Die AXIS Architektur gestattet es dem Entwickler Erweiterungen in den SOAP Engine einzubauen (Header Verarbeitung, System-Management, u.v.m.).

- **Stabilität**:

AXIS definiert publizierte Interfaces, welche sich nicht verändern, so dass alle Erweiterungen, die diese Interfaces beachten, langlebig sein sollten.

- **Komponenten-orientiertes Deployment:**

AXIS gestattet die Definition von wieder verwendbaren Handlern, mit deren Hilfe unterschiedliche Kommunikations-Szenarien („Patterns“) zwischen Geschäftspartnern oder Anwendungen realisiert werden können.

- **Transport Framework:**

Der Kern von AXIS ist völlig transportunabhängig, nicht zuletzt weil SOAP Sender und Listener über unterschiedliche Protokolle unterstützt (SMTP, FTP, messageorientierte Middleware usw.).

- **WSDL Unterstützung:**

AXIS verwendet die Web Service Description Language (WSDL). Mit deren Hilfe können einfach Stubs für remote Services gebaut und automatisch maschinenlesbare Beschreibungen (WSDL) der bereits vorhandenen Services generiert werden.

Die aktuelle Version von SOAP ist im Vergleich zu Version 1.0 wesentlich weiterentwickelt worden, vor allem in Sachen Stabilität.
In AXIS sind enthalten:

- Eine SOAP 1.1/1.2 Engine
- Flexible Konfigurations- und Ausbringungs- Möglichkeiten
- „drop-in“ Veröffentlichung der SOAP Services (JWS): .Java statt .Class Dateien
- Unterstützung aller Basisdatentypen und ein Type Mapping System für die Definition neuer Serialisierer bzw. Deserialisierer.
- Automatische Serialisierung / Deserialisierung der Java Beans, inklusive dem Mapping der Felder auf XML Elemente und Attribute.
- Automatische Konversion zwischen Java Collections und SOAP Arrays und umgekehrt.
- Providers für RPC und Message basierte SOAP Services
- Automatische WSDL Generierung von bereits verfügbar gemachten Services
- WSDL2Java Tool, um Java Proxies und Skeletons aus WSDL Dokumenten zu generieren.
- Java2WSDL Tool, um WSDL aus Java Klassen zu generieren.
- Einfache Sicherheitserweiterungen, welche mit den Servlet 2.2 Security / Roles zusammenarbeiten.
- Unterstützung von Sessions-orientierten Services, via http Cookies oder Transportunabhängigen SOAP Headern.
- Unterstützung einfacher „SOAP with Attachment“ Features.
- Ein EJB Provider, um auf EJB’s als Web Services zugreifen zu können.
- http Servlet-basierter Transport
- JMS basierter Transport
- Einen einfachen eigenständigen Server

AXIS ist keine starre Software aus einem Guss. Module gliedern AXIS in Subsysteme, die verschiedene Aufgaben übernehmen. Die Verarbeitung von Anfragen erfolgt über Filterketten, die sich aus Handlern zusammensetzen.

Die Verkettung der Handler ist flexibel und kann über Konfigurationsdateien angepasst werden. Über eigene Handler kann in die Verarbeitung von SOAP Nachrichten eingegriffen werden. Typische Aufgaben für die Handler sind Logging oder Sicherheitsüberprüfungen.
Bei der Anwendungsentwicklung ist es nicht notwendig selbst in die Filterketten einzugreifen. Mit dem Konzept der Ketten ist AXIS jedoch offen für Erweiterungen.
Damit Web Services bereitgestellt werden können, wird ein Server benötigt, der Web Anfragen verarbeiten kann. AXIS kann in einen Web Container wie den Tomcat integriert werden. Es gibt dafür eine Web Anwendung, die AXIS in Form von Servlets beinhaltet.
Das AXIS Servlet kann Web Services aufnehmen und Anfragen an diese zur Weiterverarbeitung routen. Aufgrund der Zustellungsfunktion spricht man von einem SOAP Router oder Dispatcher.

Auf die Installation und Konfiguration AXIS und Tomcat wird hier nicht näher eingegangen und auf `http://ws.apache.org/axis/java/install.html` bzw. `http://jakarta.apache.org/tomcat/tomcat-5.0-doc/setup.html` verwiesen.

Der folgende Abschnitt zeigt, wie mit Hilfe von AXIS ein einfacher Web Service implementiert und verfügbar gemacht werden kann.

Beispiel der Serverseite mit AXIS

Es folgt ein kleines Beispiel für einen Webservice, der komplexe Zahlen addieren kann. Alle Beispiele, sowohl auf der Server- als auch auf der Client-Seite, wurden im Rahmen des Teleseminars für den Vortrag eigenständig implementiert.

```
1     package calculatorwebservice;
2     public class Calculator {
3       Complex a = new Complex(5,4);
4       public Complex getA() {
5        return a;
6       }
7       public void setA(Complex a) {
8         this.a = a;
9       }
10      public void add(int a, int b){
11        this.a.add(new Complex(a,b));
12      }
13    }
```

```
1     package calculatorwebservice;
2     public class Complex {
3       double real;
4       double imag;
5       public Complex(double a, double b){
6         real=a;
7         imag=b;
8       }
9       public void add(Complex b){
10        this.real+=b.real;
11        this.imag+=b.imag;
12      }
13      public double getImag() {
14        return imag;
15      }
16      public double getReal() {
17        return real;
18      }
19      public void setImag(double imag) {
20        this.imag = imag;
21      }
22      public void setReal(double real) {
23        this.real = real;
24      }
25    }
```

Einen einfachen Web Service, der nur die Standarddatentypen benutzt und keine komplexeren Klassen, wie hier die Klasse `complex.java`, kann sehr einfach verfügbar gemacht werden: Die Endung der Datei von `.java` in `.jws` verändern und

die Datei dann in das Verzeichnis `<webapp-root>/axis/` kopieren. Der Service ist nun über `http://localhost:8080/axis/<servicename>.jws` ansprechbar.
JWS sind lediglich für einfache Web Services sinnvoll und machbar. Sie dürfen keine Pakete enthalten. Fehler sind erst nach dem Installieren, beim Methodenaufruf, erkennbar und der Quellcode des Services wird benötigt. Professionelle Web Services verwenden daher immer Web Service Deployment Descriptoren (WSDD), denn nur mit diesen ist es möglich einen komplexeren Service zu beschreiben.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <deployment
3    xmlns="http://xml.apache.org/axis/wsdd/"
4    xmlns:ns="http://calculatorwebservice"
5    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
6    <service name="Calculator" provider="java:RPC">
7     <parameter name="className" value="calculatorwebservice.Calculator"/>
8     <parameter name="allowedMethods" value="*"/>
9     <parameter name="scope" value="Application"/>
10    <typeMapping
11      xmlns:ns="http://calculatorwebservice"
12      qname="ns:Complex"
13      type="java:calculatorwebservice.Complex"
14      serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
15      deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
16      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
17    />
18  </deployment>
```

Ausgesuchte Zeilen und ihre Bedeutung:

3) Zuerst wird der Namensraum für WSDD definiert,

5) dann folgt der Namensraum für Java, die Implementierungssprache.

6) Der Service wurde als Calculator beim Server angemeldet; es handelt sich um einen RPC (keinen Messaging) Web Service in Java, in der AXIS Architektur um einen Provider, „java:RPC“.

7) Der Web Service wird mithilfe der Klasse `calculatorwebservice.Calculator` implementiert

8) Alle Methoden des Dienstes („*“) können genutzt werden.
   Alternativ könnte auch eine Liste der erlaubten Methoden (durch Leerzeichen getrennte Methodennamen) angegeben werden.

9) Bei den Scoped Services geht es darum festzulegen, wie ein Service Objekt gehandhabt wird: [7][11]
   a. „Request“ Scope (Standardeinstellung):
      für jeden SOAP Request im Service wird ein neues Objekt kreiert
   b. „Application“ Scope:
      In diesem Fall wird das Singleton Pattern benutzt d.h. ein einzelnes Objekt bedient alle Requests.

In diesem Beispiel ist Application ausgewählt, das bedeutet, dass der Wert der Variablen `Complex` a nie auf den Initialwert zurückgesetzt wird. Völlig gleich, wie viele Benutzer den Service benutzen und welche Zahlen sie addieren, der veränderte Wert wird immer mitgeführt.

c. „Session“ Scope:
kreiert für jeden Session fähigen Client ein Objekt.

Diese Methode basiert auf herkömmlichen Cookies. Es gibt auch eine weitere Form des Sessionmanagements mit AXIS. Es besteht die Möglichkeit im SOAP Header eine Session ID mitzuführen. Dies würde dann so aussehen:

```
<soapenv:Header>
    <ns1:sessionID soapenv:mustUnderstand="0" xsi:type="xsd:long"
                   xmlns:ns1="http://xml.apache.org/axis/session">
       836102490310038623
    </ns1:sessionID>
</soapenv:Header>
```

10) Eine der wichtigsten Punkte der JAX-RPC Spezifikation ist die Definition von Standard Mappings für WSDL Dokumente (also die Service Beschreibung) auf Java Konstrukte und umgekehrt. Vor JAX-RPC definierte jeder Tool Anbieter sein eigenes Mapping. Die Standardisierung basiert auf WSDL 1.1 und SOAP 1.1, sie kann also bei neuen Versionen unter Umständen überarbeitet werden.
Das `TypeMapping`:
Alle Standarddatentypen werden von AXIS wie folgt übersetzt: [7][11]

xsd:base64Binary	`byte[]`
xsd:boolean	`boolean`
xsd:byte	`byte`
xsd:dateTime	`java.util.Calendar`
xsd:decimal	`java.math.BigDecimal`
xsd:double	`double`
xsd:float	`float`
xsd:hexBinary	`byte[]`
xsd:int	`int`
xsd:integer	`java.math.BigInteger`
xsd:long	`long`
xsd:short	`short`
xsd:string	`java.lang.String`

XML Arrays werden auf Java Arrays abgebildet.

Bei der Klasse `Complex` handelt es sich nicht um einen solchen Typ. Damit ein Java Objekt mit SOAP übertragen werden kann, müssen Client und Server dieses analysieren können. AXIS kennt das Konzept des Serializers / Deserializers. Es besteht die Möglichkeit eigene Serializer / Deserializer zu schreiben und in AXIS zu registrieren. Der AXIS Bean Serializer, der hier im Beispiel benutzt wird, gestattet das (de-)serialisieren von Java Bean Klassen. In der WSDL, die als nächstes besprochen wird, taucht die Deklaration des Typs noch einmal auf.

Der Service kann nun mit Hilfe der WSDD deployed werden:

```
java org.apache.axis.client.AdminClient deploy.wsdd
```

Die Erstellung dieses Dokuments übernimmt auch AXIS, dies erfolgt in zwei Schritten. Zuerst wird eine WSDL Datei erzeugt. Diese beschreibt den Service in XML, so können Clients zu diesem Service auch in einer anderen Sprache als Java implementiert werden.

```
java org.apache.axis.wsdl.Java2WSDL -o Calculator.wsdl
-l"http://localhost:8080/CalculatorWebService/services/Calculator"
-n"CalculatorService"
-p"calculatorwebservice" "CalculatorService"
calculatorwebservice.Calculator
```

Aus der generierten WSDL werden hier nur die Zeilen angeführt, die das TypeMapping der Klasse Complex betreffen, die vollständige Datei befindet sich im Anhang. Der Typ `complex`, für den in der WSDD die Serialisierung festgelegt wurde, wird hier durch seine öffentlichen Attribute, sowie deren Typen beschrieben.

```
1     <complexType name="Complex">
2       <sequence>
3         <element name="imag" type="xsd:double" />
4         <element name="real" type="xsd:double" />
5       </sequence>
```

Aus der WSDL Datei kann nun mit einem weiteren Aufruf die WSDD Datei erzeugt werden. Hiermit wird auch eine Datei zum undeployen des Service generiert: `undeploy.wsdd`.

```
java org.apache.axis.wsdl.WSDL2Java --server-side <wsdl-file-URL>
```

Im folgenden Abschnitt wird nun ein Client implementiert, mit dem der erstellte Web Service abgefragt werden kann.

Beispiel der Clientseite mit AXIS

Für den nun implementierten und aktivierten Web Service kann mit Hilfe von AXIS sehr einfach ein Client erstellt werden. Dafür muss die Adresse des Services bekannt sein, an diese wird `?wsdl` angehängt:

`http://localhost:8080/CalculatorWebService/services/Calculator?wsdl`

An dieser Stelle kann nun ein Befehl benutzt werden, der von der Serverseite zum erstellen der WSDD schon bekannt ist.

`java org.apache.axis.wsdl.WSDL2Java <wsdl-file-URL>`

Für jedes `<type>` aus der WSDL wird eine Java-Klasse, für jedes `<portType>` ein Java-Interface, für jedes `<binding>` eine Stub-Klasse, für jedes `<service>` ein Service Locator-Interface und die zugehörige Implementierung generiert.
Unter Verwendung dieser Klassen können die Methoden des entfernten Servers wie lokal implementierte Methoden angesprochen werden. Zuvor muss, mit einer einzigen Zeile, das entfernte Objekt, das die Methoden bereitstellt, gefunden und instanziiert werden:

`Calculator service = new CalculatorServiceLocator().getCalculator();`

Der vollständige Client kann so aussehen.

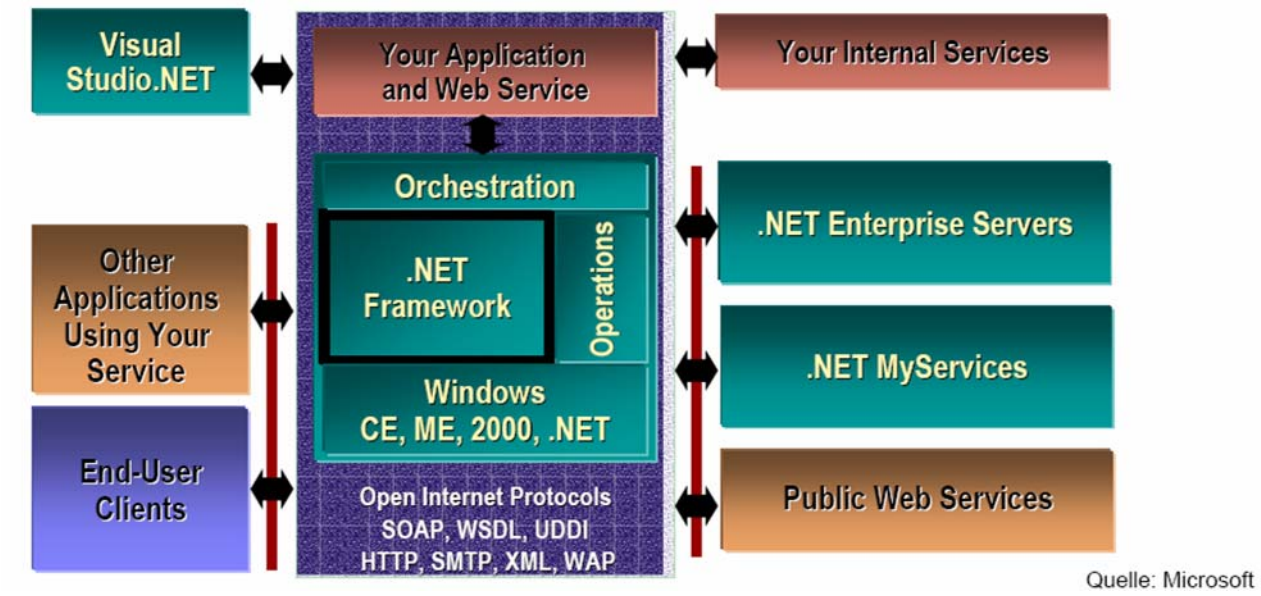
```
1  public class CalculatorClient {
2    public static void main(String[] args) {
3      try {
4        Calculator service = new
                                 CalculatorServiceLocator().getCalculator();
5        Complex temp = null;
6        System.out.print("service.getA()=");
7        temp=service.getA();
8        System.out.println(temp.getReal()+" , "+temp.getImag());
9        System.out.println("service.add(new Complex(99,23))");
10        service.add(99, 23);
11        System.out.print("service.getA()=");
12        System.out.println(service.getA().getReal()+" , "
                                              +service.getA().getImag());
13     }
14     catch (RemoteException ex) {
15       System.out.println("ex=" + ex);
16     }
17     catch (ServiceException ex) {
18       System.out.println("ex=" + ex);
19     }
20   }
21 }
```

Es wird deutlich, dass es nicht notwendig ist, besondere Java Kenntnisse zu haben, um einen Web Service anzusprechen. Zu bemerken ist hier noch, dass Dank des TypeMappings die Klasse `Complex` ganz normal benutzt werden kann.

Vergleich Java - .Net

Zu Beginn eine kurze Gegenüberstellung der beiden Plattformen, ohne speziell auf Web Services einzugehen.

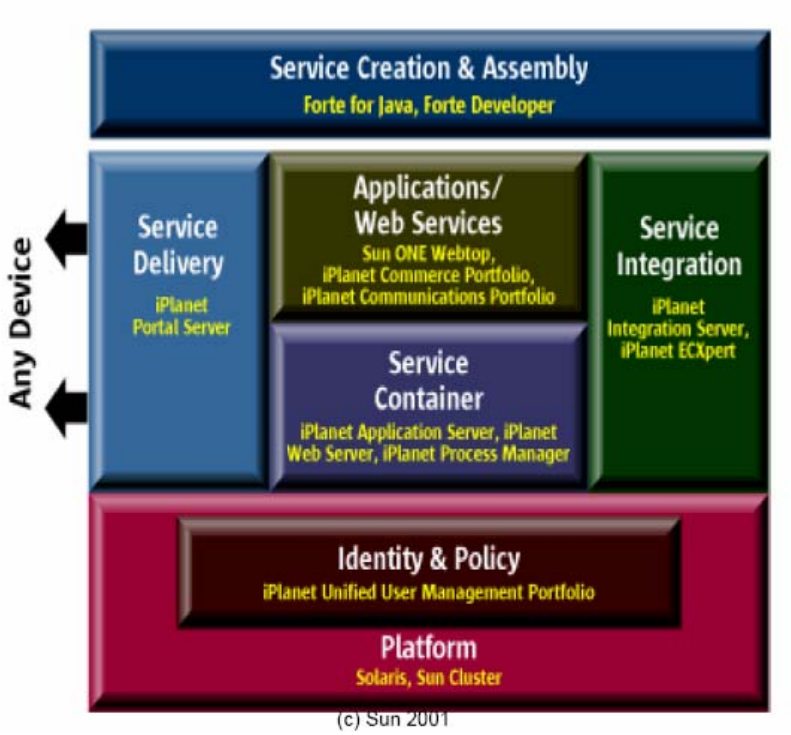
Microsoft .Net



Quelle: Microsoft

Microsoft .Net [15][16] ist ein vollständiges Framework aus einem Guss. Es integriert Betriebssystem, Serverprodukte, Anwendungen und Dienste. Es baut auf den bekannten Standards auf und zielt auch auf Kleingeräte wie PDAs. Es ist plattformabhängig, aber weitgehend programmiersprachenunabhängig. Es ist sowohl für Windows- und Web-Clients als auch für komponentenorientierte Systeme geeignet.

Sun Open Network Environment [17][13]



Die Einzelkomponenten sind sehr ausgereift, sie ergeben jedoch kein vollkommenes Framework, dessen Komponenten einträglich kooperieren. Im Zusammenspiel mit Frameworks wie z.B. AXIS verschwindet diese Schwäche jedoch. Das Sun Framework ist anderen Betriebsystemen gegenüber offen, dafür ist es an eine Programmiersprache gebunden: Java.

Vergleich

Das Grundkonzept der Laufzeitsysteme ist sehr ähnlich, denn die .NET Designer haben sich an Java orientiert [1]. Java ist im Gegensatz zu .NET prinzipiell auf Interpretation ausgelegt und erlaubt das Einbinden eigener Class-Loader und Security-Manager. Bei .Net wird die Mächtigkeit der Programmiersprache durch die Common Language Runtime (CLR) bestimmt, die auch prozedurale Sprachen erlaubt. Auch das Objektmodell hat viele Gemeinsamkeiten. Schnittstellen sind in beiden Fällen rein abstrakte Klassen. Nur einfache Implementierungsvererbung ist erlaubt, dafür Mehrfachvererbung über Schnittstellen. Die Namensräume aus .NET werden in Java zu Paketen und die Zugriffsbeschränkungen (public, private, …) sind ebenfalls auf beiden Seiten zu finden. Es gibt jedoch auch Unterschiede. In .NET sind alle Datentypen Klassen, wobei in Java zwischen primitiven Typen und Klassen unterschieden wird. In .Net haben Referenzen auf Funktionen und Methoden spezielle Typen: events und delegates.

Beim Komponentenmodell stehen den Enterprise Java Beans (EJBs) [4] [12] auf der einen, die Assemblies auf der anderen Seite gegenüber. Beide ermöglichen die

Interoperabilität mit existierenden Komponenten (COM+ in .NET und CORBA in Java). Das Modell der EJBs ist wesentlich reifer und die Container sind leistungsfähiger. In .NET existiert ein explizites Versionierungskonzept, das den gleichzeitigen Zugriff auf verschiedene Versionen erlaubt.
Beim Datenbankzugriff benutzen beide Modelle abstrakte APIs und entkoppeln jeweils Datenquelle und Nutzer. ADO.NET ist auf XML ausgerichtet und bevorzugt einen asynchronen Zugriff, dagegen ist JDBC binär und greift eher synchron zu.

Vergleich .NET – JAVA im Bereich Web Services

Web Services sind ein Kernbestandteil von .NET, es kann auf sie über verschiedene Wege zugegriffen werden. Entweder über .aspm Seiten als ASP .NET Web Service oder als .NET Remoting Web Services. Die Entwicklung ist sehr einfach, da in Visual Studio .NET eine umfangreiche Web Service Unterstützung enthalten ist. Die Kommandozeilentools, beispielsweise zur Bearbeitung der WSDL Dateien, sind bereits in der kostenlosen Version enthalten. Die .NET Web Services sind an den Microsoft Internet Information Server (IIS) und die Windows Plattform gebunden.
Sun stellt zur Entwicklung von Web Services das Java Web Services Developer Pack (JWSDP) [18] [10] bereit. Im Gegensatz zu .NET ist es ebXML konform. Es existieren verschiedene Toolkits, wie z.B. Apache AXIS, IBM Web Services Tollkit und GLUE. Web Services werden oft als Servlets oder EJBs implementiert, gerade das Zusammenspiel mit EJBs macht Web Services überhaupt erst sehr mächtig. [4] [12]
.NET bietet auch Unterstützung für mobile Web Services. Die Smart Device Extension (SDE) für Windows CE Geräte enthält ein abgespecktes .NET Framework. Dieses arbeitet komplementär zum Betriebssystem, welches das Scheduling, die Anzeige der Benutzeroberfläche, die Eingabeverarbeitung und die Ressourcenverwaltung übernimmt. Es unterstützt auch ADO.NET und SQL Server Datenhaltung, selbst mit abbrechenden Verbindungen. Das große Manko ist, dass die SDE Web Services nur als Client unterstützt, auch mittelfristig ist eine Unterstützung mobiler Geräte als Web Service Provider nicht vorgesehen.
Auf der Java Seite sind zahlreiche Toolkits für J2ME verfügbar: kSOAP, GLUE, eSOAP, Wingfoot SOAP. Diese ermöglichen das Erstellen von Web Service Servern und Clients.

Zusammenfassung und Ausblick

Die Konzepte von .NET und Java bezüglich Web Services sind sehr ähnlich, sie benutzen beide SOAP, WSDL und UDDI, dadurch ist eine Interoperabilität weitgehend gewährleistet. Beide erreichen eine effiziente Verarbeitung durch eine Vorkompilierung der Services. .NET Web Services können in allen Sprachen geschrieben werden, sie sind aber nicht ebXML konform. Java Web Services sind nicht an einen Web Server oder einen Servlet- bzw. EJB Container gebunden. Eine Möglichkeit ist AXIS zu benutzen, darauf wird im nächsten Abschnitt noch näher eingegangen. In Java ist die Web Service Unterstützung für mobile Geräte besser.

AXIS ist ein SOAP-Server (auch für reines SOAP nutzbar), der semi-automatisches Typemapping zwischen Java und XSD unterstützt. AXIS generiert automatisch WSDL Dateien zur Beschreibung des Services aus der Java Implementierung und andersherum auch Stubs aus einer WDSL zur Benutzung des Services durch den Client. AXIS hat flexible Architektur und ist gut anpassbar und erweiterbar.
Der Sicherheitsaspekt ist ein zentrales noch ungelöstes Thema, das allerdings Web Services im Allgemeinen betrifft und nicht nur AXIS.
SOAP 1.2 wird noch nicht vollständig unterstützt, in der nächsten Version wird es soweit sein. AXIS wird sich voraussichtlich in der Java Welt durchsetzen, momentan wird auch eine C++ Version entwickelt, die allerdings noch im Beta Stadium ist.

.NET bietet alles aus einer Hand, dem gegenüber wird Java von unterschiedlichen Herstellern angeboten. .NET ist eine Produktfamilie, Java (J2xE) eine Spezifikation. Web Services sind sowohl in .NET als auch in Java zu realisieren, daher ist es nicht ratsam die Plattform zu wechseln. Wenn bisher Windows verwandt wurde ist .NET die bessere Alternative, wenn bisher Unix / Linux und Java benutzt wurde Java. Die Probleme liegen nicht in der Wahl der Plattform, sondern bei der Standardisierung der XML Formate, bei der Frage ob sich UDDI als universeller Suchdienst durchsetzen wird, sowie in der Sicherheits- und Transaktionsfähigkeit.

Literaturverzeichnis

[1] .Net Framework Community Website, **Compare Microsoft .NET to J2EE Technology**, http://www.gotdotnet.com/team/compare/default.aspx, 2003

[2] Apache AXIS, **Documentation, Installation, User's Guide, Developer's Guide, Integration Guide, Architecture Guide, Reference Guide, Reading Guide, Requirements**, http://ws.apache.org/axis/

[3] Apache SOAP, **About, Features, Documentation**, http://ws.apache.org/soap/
Armstrong, Ball, Bodoff, Carson, Fisher, Fordin, Green, Haase und Jendrock, **The Java™ Web Services Tutorial**, Sun Microsystems, Inc., 2003

[4] Backschat, Gardon, **Enterprise JavaBeans: Grundlagen – Konzepte – Praxis**, Spektrum Akademischer Verlag GmbH, 2002

[5] Cerami, **Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL**, O'Reilly, 2002

[6] Foster, Porter, Wear und Hablutzel, **Developing Web Services with Java APIs for XML Using WSDP**, Syngress Publishing, Inc., 2002

[7] Frotscher, **Advanced Web Services mit Apache Axis**, http://www.w-jax.de/konferenzen/w-jax03/powerworkshops/pw02_frotscher_1.pdf, 2003

[8] Graham, Simeonov, Boubez, Davis, Daniels, Nakamura und Neyama, **Building Web Services with Java™: Making Sense of XML, SOAP, WSDL, and UDDI**, Sams Publishing, 2001

[9] Hunt, **Web Services in Java**, www.jaydeetechnology.co.uk/planetjava/tutorials/server/Web-Services-in-Java.pdf, 2003

[10] Lamprecht, **Web Services Technologien im J2EE Standard**, www.isys.uni-klu.ac.at/ISYS/ Courses/03SS/S_DKE/lamprecht.ppt, 2003

[11] Langner, **Web services mit Java: Neuentwicklung und Refactoring in der Praxis**, Markt-und-Technik-Verl., 2003

[12] Matena, Stearns, **Applying Enterprise JavaBeans: Component-Based Development for the J2EE Platform**, Addison-Wesley, 2001

[13] McGovern, Tyagi, Stevens und Matthew, **Java Web Services Architecture**, Morgan Kaufmann Publishers, 2003

[14] Micherva und Steiner, **Web Services mit Java**, http://www.informatik.uni-mannheim.de/informatik/pi4/stud/veranstaltungen/ss2004/seminar/vortraege/net.pdf, 2003

[15] Microsoft, **Defining the Basic Elements of .NET**, http://www.microsoft.com/net/basics/whatis.asp, 2003

[16] Schmi und Mazur, **Web Services mit .Net**, http://www.informatik.uni-mannheim.de/informatik/pi4/stud/veranstaltungen/ss2004/seminar/vortraege/net.pdf, 2003

[17] Sun, **Java Technology and Web Services**, http://java.sun.com/webservices/index.jsp

[18] Sun, **Java Web Services Developer Pack**, java.sun.com/webservices/jwsdp/index.jsp

Anhang

Die vollständige WSDL

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
targetNamespace="http://localhost:8080/CalculatorWebService/services/Calcul
ator"
xmlns:impl="http://localhost:8080/CalculatorWebService/services/Calculator"
xmlns:intf="http://localhost:8080/CalculatorWebService/services/Calculator"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns1="http://calculatorwebservice"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<wsdl:types>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://calculatorwebservice">
<import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<complexType name="Complex">
<sequence>
<element name="imag" type="xsd:double" />
<element name="real" type="xsd:double" />
</sequence>
</complexType>
</schema>
</wsdl:types>
<wsdl:message name="getARequest" />
<wsdl:message name="setARequest">
<wsdl:part name="a" type="tns1:Complex" />
</wsdl:message>
<wsdl:message name="setAResponse" />
<wsdl:message name="addRequest">
<wsdl:part name="a" type="xsd:int" />
<wsdl:part name="b" type="xsd:int" />
</wsdl:message>
<wsdl:message name="addResponse" />
<wsdl:message name="getAResponse">
<wsdl:part name="getAReturn" type="tns1:Complex" />
</wsdl:message>
<wsdl:portType name="Calculator">
<wsdl:operation name="add" parameterOrder="a b">
<wsdl:input name="addRequest" message="impl:addRequest" />
<wsdl:output name="addResponse" message="impl:addResponse" />
</wsdl:operation>
<wsdl:operation name="getA">
<wsdl:input name="getARequest" message="impl:getARequest" />
<wsdl:output name="getAResponse" message="impl:getAResponse" />
</wsdl:operation>
<wsdl:operation name="setA" parameterOrder="a">
<wsdl:input name="setARequest" message="impl:setARequest" />
<wsdl:output name="setAResponse" message="impl:setAResponse" />
</wsdl:operation>
</wsdl:portType>
```

```
<wsdl:binding name="CalculatorSoapBinding" type="impl:Calculator">
<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />
<wsdl:operation name="add">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="addRequest">
<wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/CalculatorWebService/services/Calculator"
/>
</wsdl:input>
<wsdl:output name="addResponse">
<wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/CalculatorWebService/services/Calculator"
/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getA">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="getARequest">
<wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/CalculatorWebService/services/Calculator"
/>
</wsdl:input>
<wsdl:output name="getAResponse">
<wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/CalculatorWebService/services/Calculator"
/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="setA">
<wsdlsoap:operation soapAction="" />
<wsdl:input name="setARequest">
<wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/CalculatorWebService/services/Calculator"
/>
</wsdl:input>
<wsdl:output name="setAResponse">
<wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/CalculatorWebService/services/Calculator"
/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="CalculatorService">
<wsdl:port name="Calculator" binding="impl:CalculatorSoapBinding">
<wsdlsoap:address
location="http://localhost:8080/CalculatorWebService/services/Calculator"
/>
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```