Vorlesungsreihe
Entwicklung webbasierter Anwendungen

Entwicklung von Web Services

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung

- Optionen zur Entwicklung von Web Services
  - PHP-Bibliotheken (NuSOAP und neue Bibl. in PHP5)
  - Java- Webservices & Frameworks
  - Quellen :
    - www.php.net (Handbuch)
    - http://ws.apache.org/axis2/ (Axis-Framework)

SOAP-Tools und Entwicklungswerkzeuge

- **Programmierung mit PHP**
- **Standardimplementierungen für Java und C :**
- Angepasste Bibliotheken für Apache oder Tomcat-Webserver
- XML-Parser Xerces (ebenfalls von Apache-Group)
- SOAP-Engine Axis :
  - Java-Implementierung der W3C Empfehlung zu SOAP
  - basierend auf Apache SOAP 3.0 (komplett überarbeitet mit höherer Perf.)
  - unterstützt die Web Service Description Language
  - automat. Generierung von Java-Clients durch Verarbeitung der WSDL-Datei
  - nicht nur Versendung über HTTP, sondern auch FTP etc

Weitere Tools von bekannten Herstellern, insbesondere

- Software AG mit Tamino XML-Datenbank
- IBM Websphere-Entwicklungsumgebung
- Microsoft - .NET-Framework mit SOAP-Modulen

Entwicklung von Web Services mit PHP

- vor PHP Version 5.0 waren keine integrierten Webservice-Funktionalitäten verfügbar
- PHP Version < 5.0 -> Ausweg - NuSOAP-Bibliothek
  - Download : http://sourceforge.net/projects/nusoap/
  - Einführung unter http://www.scottnichol.com/nusoapintro.htm
  - objektorientierte SOAP-Bibliothek
  - **Achtung : letzte Version von 2005, anscheinend keine Pflege mehr**

- ab PHP 5 - integrierte Webservice – Funktionen
  - die integrierte SOAP-Erweiterung von PHP erlaubt das Erstellen von SOAP Servern und Clients
  - Es werden die Standards SOAP 1.1, SOAP 1.2 und WSDL 1.1 unterstützt.
  - Installationsvoraussetzungen :
    - PHP mit der Option --enable-soap konfigurieren (php.ini)
    - zusätzliche GNOME xml Bibliothek (Version libxml-2.5.4) wird vorausgesetzt.

Web Services mit PHP - Server

- Das Objekt **SoapServer** kapselt einen SOAP-Server unter PHP
- Aufruf entweder mit Angabe einer WSDL-Datei oder ohne WSDL
- optionale Angaben z.B. zum SOAP-Standard (vgl. Zeile 2) oder Zeichensatz als Array übergeben

```
$server = new SoapServer("some.wsdl");   // mit WSDL-Datei-Angabe

$server = new SoapServer("some.wsdl", array('soap_version' => SOAP_1_2));

$server = new SoapServer("some.wsdl", array('encoding'=>'ISO-8859-1'));

$server = new SoapServer(null, array('uri' => "http://test-uri/")); // ohne WSDL
```

Web Services mit PHP - Server-Methoden - addfunction

- Dem Objekt **SoapServer** werden mit

  SoapServer->addFunction()

  entsprechende Funktionen zur Bearbeitung von SOAP-Requests bekannt gemacht. (Der Name der Funktion wird als SOAP-Service exportiert und muss in der WSDL-Datei als Service auftauchen !)

```
function echoString($inputString)
{      return $inputString;}
$server->addFunction("echoString");
// auch mehrere Funktionen gleichzeitig registrieren ....
function echoTwoStrings($inputString1, $inputString2)
{    return array("outputString1" => $inputString1,
                  "outputString2" => $inputString2); }
$server->addFunction(array("echoString", "echoTwoStrings"));
```

Alternativ können mit

```
$server->addFunction(SOAP_FUNCTIONS_ALL);
```

ALLE Funktionen verfügbar gemacht werden !

Web Services mit PHP - Server-Methoden - addfunction

- Alternativ können auch Klassen incl. Methoden als Bearbeitungsfunktionen definiert werden :

```
class demo1 {
    function func1()   // Funktion ohne Parameter
                { ... }
}
$server->setClass("demo1");

class demo2 {      function func2($x, $y)
                            { .... }
}
$server->setClass("demo2", $arg1, $arg2);  // mit Parametern
```

Web Services mit PHP - Server- weitere Methoden

- **Weitere Hilfsfunktionen in SoapServer**

SoapServer->fault() - zur Anzeige und Behandlung von SoapServer-Fehlern

void **fault** (string $code, string $string[, string $actor [, mixed $details [, string $name]]])

(auch unter php.net keine weiteren Dokus zu den Fehlern selbst)

SoapServer->getFunctions() - gibt eine Liste aller definierten Funktionen zurück

SoapServer->handle() - Abarbeiten von SOAP-Anfragen

SoapServer->setPersistence() - aktiviert Persistenz-Modus (speichert Objekte ab – per Session)

Web Services mit PHP - Client - Objekt

- analog zum Server kapselt auch ein SoapClient die Funktionalität.
- Aufruf wieder mit WSDL oder OHNE WSDL und optionalen Parametern

```
$client = new SoapClient("some.wsdl");  // mit WSDL
$options = array('soap_version'   => SOAP_1_2));
$client = new SoapClient("some.wsdl", options ); // mit zus. Optionen
$client = new SoapClient(null, array('location' =>
"http://localhost/soap.php",'uri'      => "http://test-uri/"));  // ohne WSDL
```

Als Optionen können u.a. definiert werden :

- soap_version - > SOAP 1.1 oder SOAP 1.2
- login / password zur optionalen HTTP-Authentifizierung
- Bei Verbindung über Proxies Angabe der Parameter mit proxy_host, proxy_port, proxy_login und proxy_password
- compression – zur optionalen Kompression der Daten (muss Server unterstützen)
- encoding – (interne) Zeichenkodierung, SOAP-Anfrage selbst immer utf-8
- exceptions (boolean) steuert Fehler-Exceptions vom Typ SoapFault
- connection_timeout - Timeout der connection , trace zum Nachverfolgen

```
Bsp: opt=  array('compression' => SOAP_COMPRESSION_ACCEPT |
                                  SOAP_COMPRESSION_GZIP));
```

Web Services mit PHP - Client - Methoden

- Falls eine WSDL verwendet wird, erzeugt PHP entsprechende PHP-Funktionen :

```
$result =  $client->meinefunktion();
$result =  $client->meinefunktion($a, $b);
```

- Rückgabe generell über Funktionswert, mehrere Werte per Array !

- alternativ (oder immer bei Anlage OHNE WSDL) Aufruf mit soapCall
- Syntax: mixed **__soapCall** (string $function_name , array $arguments [, array $options [, mixed $input_headers [, array &$output_headers]]])
- Bsp.:

```
$client->__soapCall("meinefunktion", array($a, $b, $c));
$client->__soapCall(" meinefunktion", array($a, $b, $c), NULL,

        new SoapHeader(), $output_headers);
```

Web Services mit PHP - Client - Weitere Methoden

- Weitere Hilfsmethoden des SoapClient-Objekts

- $SoapClient->__getFunctions() - Liste der verfügbaren SOAP Funktionen
- SoapClient->__getTypes() - Liste der verfügbaren SOAP-Typen
  - Beide Funktionen setzen WSDL-Modus voraus

Bei gesetzter Option Trace können Daten zum letzten Aufruf abgefragt werden :

- $SoapClient->__getLastRequest() - gibt letzte SOAP-Anfrage zurück
- $SoapClient->__getLastRequestHeaders() - letzter SOAP-Anfrage-Header
- $SoapClient->__getLastResponse() - gibt letzte SOAP-Antwort zurück
- SoapClient->__getLastResponseHeaders() - SOAP-Antwort-Header zurück

Web Services mit PHP -Anwendungsbeispiel -

Abgleich von Adressen auf einen zentralen Adressenserver

```
$q1base= "select * from adressen ";  $result = mysql_query($q1 ,$connect); // Daten
echo "<BR>Starte Adressen -Abgleich für Stammdaten<BR> ";
 $client = new SoapClient('http://www.wwlservice.de/webservice/wwl_fv.php?wsdl' );

while ( ($row = mysql_fetch_array ($result) )  && $rowcount<$rowmax )
{  $rowcount++;  $parameter['username'] = $uname; $parameter['password'] =  $pw1;
   $parameter['passwordcheck'] = md5( $pw1 .$fvkey);
   $parameter['firma1']  = utf8_encode( $row['Firmenname'] ); //   Stammdaten setzen
   $parameter['strasse']  = utf8_encode( $row['Strasse'] ) ;
   $ergebnis = $client->editStammdaten( $parameter );

    if (strcmp($ergebnis->status,"USER_NOT_FOUND") ==0 )
        {   echo  "<br>Insert NEW user ... <BR><hr>" ;
             $ergebnis = $client->addUser( $parameter );
            if (strcmp($ergebnis->status,"OK") ==0 )
                        {     $statusinfo="OKnew";                    }
       else   {   $statusinfo="Error-newUser: " . $ergebnis->status ;        }   }
```

Web Services unter Java

Optionen

- Mit **AXIS-Bibliothek** (Apache eXtensible Interaction System) der Apache-Foundation (war erste Option vor der Verfügbarkeit von eigenen Java-Packages)
  - http://ws.apache.org/axis/ bzw. http://ws.apache.org/axis2/
  - Aktuelle Version Axis2 mit deutlich verbesserter Geschwindigkeit, weniger Speicherbedarf, **Hot Deployment** (neue WS ohne Start/Stop des Servers), etc.
- mit Java Web Services Developer Pack (JWSDP)
- **oder den ab Java SE6 nun direkt verfügbaren javax.jws.WebService - Standardlibraries**
- **Bei allen 3 Optionen müssen die erzeugten Java-Class-Dateien und zusätzliche Konfigurationsskripte auf einem Anwendungsserver deployed werden !**

Web Services mit AXIS – Java - WS-Server

```
import javax.xml.stream.XMLStreamException;
import org.apache.axiom.om.*;

public class StockQuoteService {
    public OMElement getPrice(OMElement element) throws XMLStreamException {
        // Berechnung des Preises ausgehend vom Firmennamen
        Double price = (Double) get_price_form_db( element) ;
      String returnText = "-1";
        if(price != null){ returnText  = "" + price.doubleValue(); }

        OMFactory fac = OMAbstractFactory.getOMFactory(); // Axis-Objekte
        OMNamespace omNs =
            fac.createOMNamespace("http://axiom.service.quickstart.samples/xsd", "tns");
        OMElement method = fac.createOMElement("getPriceResponse", omNs);
        OMElement value = fac.createOMElement("price", omNs);
        value.addChild(fac.createOMText(value, returnText));
        method.addChild(value);
        return method;
    }     }      }
```

Web Services mit AXIS – Java - WS-Client

```
import org.apache.axiom.om.*;
public class AXIOMClient {
    private static EndpointReference targetEPR =
        new EndpointReference("http://localhost:8080/axis2/services/StockQuoteService");

   public static OMElement getPricePayload(String symbol) {
        OMFactory fac = OMAbstractFactory.getOMFactory();
        OMNamespace omNs =
     fac.createOMNamespace("http://axiom.service.quickstart.samples/xsd", "tns");

        OMElement method = fac.createOMElement("getPrice", omNs);
        OMElement value = fac.createOMElement("symbol", omNs);
        value.addChild(fac.createOMText(value, symbol));
        method.addChild(value);
        return method;
    }

    public static OMElement updatePayload(String symbol, double price) {
         // analog zum Ändern des Preises ....
      }
```

Web Services mit AXIS – WS-Client - Main- Funktion

```
public static void main(String[] args) {
    try {     OMElement getPricePayload = getPricePayload("GE");

        Options options = new Options();
        options.setTo(targetEPR);
        options.setTransportInProtocol(Constants.TRANSPORT_HTTP);

        ServiceClient sender = new ServiceClient();
        sender.setOptions(options);

        sender.fireAndForget(updatePayload);
        System.err.println("price updated");
        OMElement result = sender.sendReceive(getPricePayload);

        String response = result.getFirstElement().getText();
        System.err.println("Current price of WSO: " + response);

    } catch (Exception e) {  e.printStackTrace();
    }    }    }
```

Web Services mit JWS – WS-Server

```
package demo1.ws1;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@WebService
@SOAPBinding(style=Style.RPC)            // Java – Annotations

public class Calculator {   // Nutzklasse
public long addValues(int val1, int val2) {  return val1 + val2;     }      }
----------
package demo1.ws1service;
import javax.xml.ws.Endpoint;
import demo1.ws1;
        public class CalculatorServer {
         public static void main (String args[]) {
        Calculator server = new Calculator();
        Endpoint endpoint = Endpoint.publish("http://localhost:8080/calculator", server);
} }
```

Web Services mit JWS – WS-Client

```
import java.net.*;  import javax.xml.*;
/**     This class was generated by the JAX-WS RI.   From the WSDL-file  */
@WebServiceClient(name = "CalculatorService", targetNamespace = "http://...", wsdlLocation
     = "http://localhost:8080/calculator?wsdl")

public class CalculatorService  extends Service
{    public CalculatorService(URL wsdlLocation, QName serviceName) {
       super(wsdlLocation, serviceName);     }
@WebEndpoint(name = "CalculatorPort")
   public Calculator getCalculatorPort() {
      return (Calculator)super.getPort(new QName("http://../", "CalculatorPort"),
     Calculator.class);
   } }; //
---------------------------
public class CalculatorClient {  // Aufruf des WS
public static void main(String args[]) {
            CalculatorService service = new CalculatorService();
            Calculator calculator = service.getCalculatorPort();
            System.out.println("Summe: " + calculator.addValues(17, 13));
}}
```

Web Services – Entwicklung – Zusammenfassung

Allgemein

- Grundprinzipien des Aufbaus von SoapServer und SoapClient immer gleich
- programmtechnischer Aufwand je nach Sprache und Bibliothek sehr stark differierend
- zum Teil automatische Generierung von WSDL-Dateien aus vorhandenem Code und/oder einer Codegenerierung aus WSDL-Dateien
- In den letzten 2 Jahren Integration von WS-Funktionalität in die Basisbibliotheken, damit bessere Unterstützung

Sprachspezifisch

- Deployment-Aufwand bei Java deutlich höher als bei PHP
- dafür mehr Optionen bzgl. Verfügbarkeit / Sicherheit durch Applikationsserver