SOA, Webservices und SOAP für Schnelleinsteiger

Inhalt

Einführung

Was ist SOA?

Die **S**ervice **O**riented **A**rchitecture ist ein Konzept zur Bereitstellung unabhängiger, loser und wiederverwendbarer Services. Im Gegensatz zu einem Data Warehouse werden nicht alle Daten, Funktionen und Schnittstellen an einem zentralen Ort gelagert und angeboten, sondern die verschiedenen Prozesse in einem Unternehmen beherbergen die Daten selbst, und stellen wohldefinierte Schnittstellen zum Datenaustausch zur Verfügung. Da SOA ein transzendentes Konzept ist, schreibt es nicht die Benutzung bestimmter Protokolle und Standards voraus.

Webservices, SOAP und WSDL

WebServices sind die konkrete Umsetzung einer SOA. Das Wort "Web" bedeutet in diesem Falle nicht, dass eine Website zur Verfügung gestellt wird, sondern nur dass das HTTP-Protokoll benutzt wird und ein Webserver als Serviceanbieter dient. Dies hat netzwerktechnische Gründe, da in den meisten Unternehmen viele Ports anderer Schnittstellen und Protokolle gesperrt sind.

Das meistverwendete Anwendungsprotokoll, mit dessen Hilfe die Daten zwischen den verschiedenen Systemen ausgetauscht werden, heißt SOAP. SOAP ist ein XML-Derivat, dass eine Weiterentwicklung von XML-RPC darstellt. Alle Daten einer SOAP-Nachricht werden in einen Umschlag (Envelope) als Wurzelelement verpackt. Innerhalb eines Umschlages ist die Nachricht in einen optionale Header und einem Nachrichten-Body aufgeteilt. Der Header enthält ausschließlich Meta-Informationen, der Body die eigentlichen Nutzdaten. Im Gegensatz zu XML-RPC muss der Nachrichtenaustausch mit SOAP nicht synchron erfolgen.

Um WebServices zu beschreiben wird WSDL (**W**eb **S**ervices **D**escription **L**anguange) verwendet, was wiederum ein XML-Derivat darstellt. Diese Datei beinhaltet die Lage der Services, die verfügbaren Services, die benötigten Parameter, die Rückgabewerte, die benutzten Protokolle etc...

SOAP mit PHP5

Benötigte Komponenten

Benötigt wird ein Webserver mit PHP5 und aktivierter SOAP Extension.
Dazu muss PHP mit "**--enable-soap**" compiliert werden oder bei der Binary-Windows-Distribution in der **php.ini** die Extension aktiviert werden.

Client ohne WSDL

Nehmen wir an es existiert eine Funktion "**getServerTime**", die als WebService angeboten wird. Die Funktion hat keine Parameter und liefert als Rückgabewert einen Integer, der eine UNIX-Timestamp enthält. Eine simple Implementierung könnte so aussehen:

```php
<?php
try
{
    $client = new SoapClient(NULL,
        array("location"  => "http://[Ihr Server]/[Ihr Service]",
              "uri"       => "urn:getServerTime",
              "style"     => SOAP_RPC,
              "use"       => SOAP_ENCODED));

    print($client->__soapCall("getServerTime",
            array(),
            array("uri" => "urn:getServerTime",
                  "soapaction" => "urn:getServerTime")));
}
catch (SoapFault $e)
{
        print($e);
}
?>
```

Wie man sieht muss man eine Menge schreiben um eine simple Funktion aufzurufen. Zuerst wird der SoapClient durch Angabe der WSDL, in unserem Falle "**null**" und zusätzlichen Parameter erzeugt. Ist keine WSDL-Datei vorhanden, sind die Parameter zwingend erforderlich, ansonsten optional, da diese automatisch aus der WSDL-Datei ausgelesen werden können. Danach wird die Funktion durch einen Soap-Call aufgerufen. Der erste Parameter ist der Name der aufzurufenden Funktion, danach kommen die zu übergebende Parameter, und zuletzt noch zusätzliche Optionen.

Client mit WSDL

Nun wäre es doch schön zu wissen, welche Funktionen überhaupt existieren, und wie diese definiert sind. Mit einer WSDL-Datei wird dies ermöglicht. Der Soap-Call vereinfacht sich dadurch sehr:

```php
<?php
try
{
    $client = new SoapClient("http://[Ihr Server]/[Ihre WSDL]");
    print($client->getServerTime());
}
catch (SoapFault $e)
{
    print($e);
}
?>
```

Durch parsen der WSDL-Datei werden alle Services zu direkt aufufbaren Methoden des Clients. Die Handhabung wird dadurch einfacher, allerdings die Last und der Netzwerk-Traffic gesteigert.

Die WSDL, die diesen Service beschreibt kann folgendermaßen aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  targetNamespace="http://[Ihr Server]/[Ihr Service]"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://[Ihr Server]/[Ihr Service]"
  xmlns:intf="http://[Ihr Server]/[Ihr Service]"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <wsdl:message name="getServerTimeRequest">
   </wsdl:message>
   <wsdl:message name="getServerTimeResponse">
      <wsdl:part name="getServerTimeReturn" type="xsd:int" />
   </wsdl:message>

   <wsdl:portType name="server1">
      <wsdl:operation name="getServerTime">
         <wsdl:input message="impl:getServerTimeRequest"
           name="getServerTimeRequest"  />
         <wsdl:output message="impl:getServerTimeResponse"
           name="getServerTimeResponse" />
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="server1SoapBinding" type="impl:server1">
      <wsdlsoap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http" />
      <wsdl:operation name="getServerTime">
         <wsdlsoap:operation soapAction="" />
         <wsdl:input name="getServerTimeRequest">
            <wsdlsoap:body
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="http://DefaultNamespace" use="encoded" />
         </wsdl:input>
         <wsdl:output name="getServerTimeResponse">
            <wsdlsoap:body
              encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="http://[Ihr Server]/[Ihr Service]" use="encoded" />
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="server1Service">
      <wsdl:port binding="impl:server1SoapBinding" name="server1">
         <wsdlsoap:address location="http://[Ihr Server]/[Ihr Service]" />
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

Eine WSDL-Datei gliedert sich in 2 Grundbestandteile: einen abstrakten Teil, der die Datentypen und die Schnittstellen definiert, und einen konkreten Teil, der das Binding und die Adresse des Service definiert. In unserem Beispiel fehlt die Datentypdefinition, die durch die Sektion **<types>** angegeben wird, da wir auf den Einsatz von komplexen Typen verzichten. Der Abschnitt **<message>** beschreibt die Nachrichten, die zwischen Server und Client ausgetauscht werden können. **<portType>** beschreibt, um welche Art des Nachrichtenaustauschs es sich handelt, ob es eine Request-Response-Operation ist, eine Notification, d. h. eine Nachricht des Servers ohne Anfrage, oder ein asynchroner Nachrichtenaustausch. Das **<binding>** gibt die verwendeten

Protokolle und das Nachrichtenformat an. Der letzte Abschnitt **<services>** definiert den Service, und beschreibt aus welchen Ports er sich zusammensetzt.

SOAP-Server

Nachdem wir nun 2 Clients programmiert haben, wollen wir uns natürlich auch die andere Seite, den Server, anschauen:

```php
<?php
/**
 * @desc Gibt die aktuelle Serverzeit zurück
 * @return int
 */
function getServerTime()
{
        return time();
}

ini_set("soap.wsdl_cache_enabled", "0");
$server = new SoapServer("webservice1.wsdl");
$server->addFunction("getServerTime");
$server->handle();
?>
```

Mit dem ini_set Befehl wird das Caching der WSDL-Datei verhindert. Da das Parsen der WSDL-Datei bei jedem Serviceaufruf viel Performance braucht, sollte man die Zeile im Produktivbetrieb wieder entfernen. Standardmäßig wird die Datei einen Tag gepuffert. Die WSDL-Datei muss vorhanden sein und per Hand erzeugt werden, da die SOAP-Extension in PHP zur Zeit noch keine Generierung dieser unterstützt. Andere SOAP-Implementierungen wie NuSOAP oder PEAR::SOAP unterstützen WSDL-Generierung schon, allerdings nur in Verbindung mit PHPDoc, da die Reflection-API in PHP kein vollständiges Type-Hinting beherrscht. Da diese Implementierungen im Gegensatz zur nativen C-Extension in PHP geschrieben sind, ist die Performance aber um einiges schlechter. Außerdem befindet sich Ext/SOAP gerade erst im Anfangsstadium der Entwicklung, so daß wir auf eine zukünftige WSDL-Generierung hoffen dürfen.

Nachdem wir nun den Soapserver mit Hilfe der WSDL-Datei initialisiert haben, fügen wir die Funktion mit "**addFunction**" durch Angabe des Funktionsnames als Webservice hinzu. Als Letztes teilen wir dem Serverobjekt durch "**handle**" mit, dass er SOAP-Requests verarbeiten soll.

SOAP mit Java und Axis

Benötigte Komponenten

Ein J2EE-WebServer (Tomcat) mit installiertem AXIS Framework.

Java client

Da SOAP von einer Programmiersprache unabhängig ist, da der Datenaustausch über XML erfolgt, können wir den vorher implementierten PHP Webservice natürlich mit einem in Java implementierten Client abfragen.

```
 1  import org.apache.axis.client.Call;
 2  import org.apache.axis.client.Service;
 3  import javax.xml.namespace.QName;
 4
 5  public class ClientTest
 6  {
 7      public static void main(String[] args)
 8      {
 9          try
10          {
11              String endpoint = "http://[Ihr Server]/[Ihr Service]";
12
13              Service service = new Service();
14              Call call = (Call) service.createCall();
15
16              call.setTargetEndpointAddress( new java.net.URL(endpoint) );
17              call.setOperationName("getServerTime");
18              Integer ret = (Integer) call.invoke( new Object[] {} );
19
20              System.out.println("ServerTime: " + ret);
21          }
22          catch (Exception e)
23          {
24             System.err.println(e.toString());
25             e.printStackTrace();
26          }
27          System.exit(0);
28      }
29  }
```

Damit dieses Programm compiliert müssen sich natürlich die benötigten AXIS-Libaries im classpath befinden.

Der String endpoint gibt die Adresse des Webservice an, den wir vorher in PHP implementiert haben. Danach wird ein call erzeugt, der als Attribute die Serviceadresse und den Funktionsnamen bekommt. Zuletzt führen wir durch "**invoke**" den Service-Call aus. Da die Funktion keine Parameter benötigt, übergeben wir nur einen leeren Objekt-Array. Den Rückgabewert des Aufrufs können wir sofort auf einen Integer casten, da wir den Typ des Wertes schon kennen.

Java Webservice mit AXIS

Mit dem AXIS-Framework ist es kinderleicht einen WebService zu erstellen. Alles was man dazu benötigt ist eine normale Java-Klasse, die als Endung nicht ".java" sondern ".jws" erhält.

```
 1  import java.util.Date;
 2
 3  public class MyWebService
 4  {
 5      public long getServerTime()
 6      {
 7              Date now = new Date();
 8              return now.getTime();
 9      }
10  }
```

AXIS stellt automatisch alle Methoden, die public sind, als Webservice bereit. Außerdem liefert es automatisch die WSDL-Datei, wenn man den Service mit "http://[Ihr Server]/axis/MyWebService.jws?wsdl" aufruft. Ein weiterer Vorteil von AXIS ist, dass man den Webservice auch anstelle eines SOAP-Request mit einem normalen GET-Request aufrufen kann. In unserem Falle wäre das "http://[Ihr Server]/axis/MyWebService.jws?method=getServerTime". Zu beachten ist, dass die Rückgabenachricht allerdings eine SOAP-Nachricht ist. Man spart sich hier den SOAP-Overhead zumindest in einer Richtung.

Literaturhinweise

SOA

- Definition SOA, Wikipedia
  http://de.wikipedia.org/wiki/Service_Oriented_Architecture
- What is Service-Oriented Architecture, webservices.xml.com
  http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html
- Discover the role of information management in SOA, IBM
  http://www-128.ibm.com/developerworks/webservices/library/ws-soa-ims/

SOAP

- SOAP, W3C
  http://www.w3.org/TR/soap/
- PHP SOAP Extension, Dmitry Stogov
  http://www.zend.com/php5/articles/php5-SOAP.php
- Access an enterprise application from a PHP script, IBM
  http://www-106.ibm.com/developerworks/opensource/library/os-phpws/
- PHP SOAP extension manual, php.net
  http://de2.php.net/manual/en/ref.soap.php

WSDL

- Web Services Description Language (WSDL) 1.1, W3C
  http://www.w3.org/TR/wsdl
- Understanding WSDL, Microsoft
  http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/library/en-us/dnwebsrv/html/understandwsdl.asp