



Hochschule für Angewandte
Wissenschaften Hamburg
Hamburg University of Applied Sciences

Amine El Ayadi

Webservices: REST vs. SOAP

Ausarbeitung eingereicht im Rahmen der Veranstaltung Anwendung1
im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft

Abgegeben am 25. Juli 2008

Inhaltsverzeichnis

Inhaltsverzeichnis	2
1. Einführung und Motivation.....	3
1.1 SOA – Service Oriented Architecture	3
1.2 Webservices.....	3
1.3 Motivation.....	4
2. SOAP-Webservices	4
2.1 SOAP (Simple Object Access Protocol)	5
2.2 WSDL (Web Service Description Language)	6
2.3 UDDI (Universal Description, Discovery and Integration)	6
2.4 Tool-Unterstützung	7
2.5 Beispielanwendung	7
3. REST (Representational State Transfer).....	8
3.1 Definition	8
3.2 Merkmale einer REST-Anwendung.....	9
3.3 Rest- Dokumentwechsel mittels XLink	10
3.4 Rest-Kommunikationsprotokoll.....	11
3.5 Vorteile von REST	12
3.6 Wie realisiere ich einen REST-Webservice?.....	12
3.7 Tool-Unterstützung	13
3.8 Beispielanwendung	13
4. REST vs. SOAP.....	14
5. Fazit.....	15
Quellen.....	16

1. Einführung und Motivation

Service, Funktionalitäten oder Methoden, die im Web angeboten werden, können als Dienste bzw. Webservices betrachtet werden. Solche Dienste lassen sich auf verschiedene Arten integrieren. Heute existieren mit REST und SOAP zwei Ansätze zur Integration mit Webservices, die sehr unterschiedliche Ansätze verfolgen. Im Folgenden werden beide Technologien beschrieben und verglichen. Dabei wird zuerst die Notwendigkeit für den Ansatz von Webservices durch eine Motivation erläutert, danach werden SOAP-Webservices vorgestellt und durch eine Beispielanwendung beschrieben, dann werden REST-basierte Webservices und deren Vorteile definiert. Zuletzt findet ein Vergleich statt. Ziel dieser Arbeit ist es, den Entwicklern eine Unterstützung bei der Entscheidung zwischen beiden Technologien zu geben.

1.1 SOA – Service Oriented Architecture

Die Integration verteilter Applikationen und Softwaresysteme in Unternehmen, zwischen Unternehmen und im Internet gewinnt zunehmend Bedeutung.

Die SOA (Service Oriented Architecture/ service-orientierte Architektur) gilt seit einigen Jahren als Best Practice für diese Integration. Dabei stellt sich immer die Frage, in welcher Form der Service Provider die Nachricht (Service Request) erhalten soll – oder, genauer ausgedrückt, wie die Kommunikation zwischen den einzelnen Services ablaufen soll.

Passende Kopplungstechnologien müssen entsprechend plattform- und programmiersprachen-unabhängig sein und dürfen nur sehr geringe Annahmen über die integrierten Systeme machen, vom daher ist eine lose Kopplung notwendig.

Webservices kann man als Umsetzung dieses Ansatzes mit Internet-Technologien sehen.

1.2 Webservices

Webservices sind in den letzten Jahren sehr populär geworden. Durch ein XML-gekapseltes Protokoll gewährleisten Webservices eine zuverlässige Kommunikation und ermöglichen, heterogene Systeme miteinander zu kombinieren.

Im Rahmen einer Studie [HiWS], bei der 610 Unternehmen befragt wurden, wurde ermittelt, wo bereits Webservices eingesetzt werden, wo dies noch nicht der Fall ist und wo dies geplant ist (s. Abbildung 1).

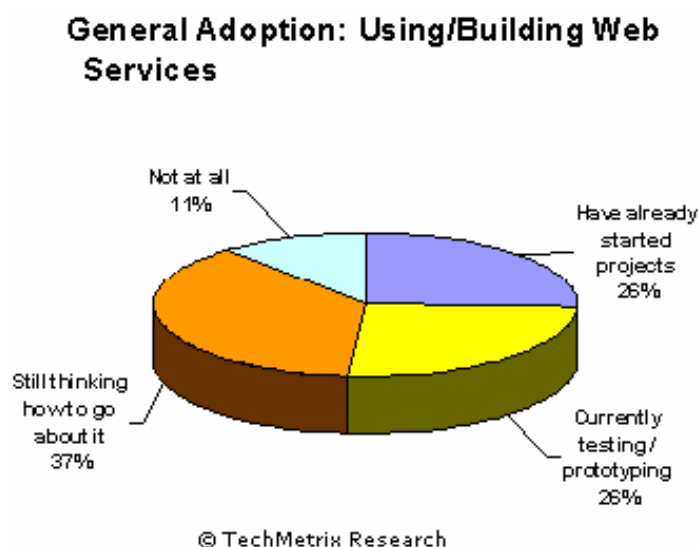


Abbildung 1: Verwendung/Aufbau-Webservices [HiWS]

Das Ergebnis der Befragung:

In 26 Prozent der befragten Unternehmen werden Webservices bereits eingesetzt, bei weiteren 26 Prozent befinden sich diese in der Projektphase. 37 Prozent der Studienteilnehmer gaben an, Webservices zumindest in Erwägung zu ziehen und nur 11 Prozent haben sich bislang noch überhaupt nicht mit diesem Thema auseinander gesetzt.

1.3 Motivation

Webservices werden meist mit SOAP in Verbindung gebracht. Thomas Roy Fielding, einer der Hauptautoren, die die Spezifikation Hypertext Transfer Protocol (HTTP) eingeführt haben, beschreibt in seiner Dissertation einen Architekturstil, den er REpresentational State Transfer oder kurz REST nennt. Dabei handelt es sich um eine weitere Alternative für die Realisierung von Webservices. REST baut auf Prinzipien auf, die in der größten verteilten Anwendung überhaupt eingesetzt werden – dem World Wide Web.

Das Web stellt selbst eine gigantische REST-Anwendung dar, wobei viele Shops, Suchmaschinen oder Buchungssysteme ohne Absicht bereits als REST-basierter Webservice implementiert sind. Das Architekturmodell REST beschreibt, wie das Web funktionieren sollte und ist dazu gedacht, als Anleitung und als Referenz für zukünftige Erweiterungen zu dienen.

REST ist kein Produkt oder Standard. REST beschreibt, wie Webstandards in einer webgerechten Weise eingesetzt werden können.

2. SOAP-Webservices

SOAP ist ein XML-Kommunikationsprotokoll, welches auf einer einfachen Idee beruht und im Laufe der Jahre um immer mehr Funktionen erweitert wurde. Dadurch ist SOAP heute sehr mächtig, aber auch entsprechend komplex. Das Verlockende an SOAP war ursprünglich die Einfachheit und Eleganz, aber durch die neuen Standards und APIs, die hinzugekommen sind, hat SOAP seine Einfachheit verloren und seine Komplexität entsprechend erhöht.

SOAP stand früher für *Simple Object Access Protocol*. Inzwischen wird SOAP nicht mehr als Akronym, sondern als Eigennamen verwendet.

Die Grundidee von SOAP-Webservices ist, einen Request in Form einer XML-Datenstruktur () an einen definierten Server zu schicken, wobei der Server einen Router enthält, der die XML-Datenstruktur interpretiert und die aufzurufende Operation am Server erkennt. Die benötigten Parameter für die Operation werden aus den XML-Daten ausgelesen und dem Operationsaufruf mitgegeben.

SOAP, WSDL und UDDI bilden die drei Kern-Standards für SOAP Webservices (s. Abbildung 2), Dabei arbeitet das Simple Object Access Protocol (SOAP) mit XML-Syntax und stellt die Verbindung zwischen Client und Web Service her.

Die *Web Services Description Language* (WSDL) beschreibt die Schnittstellen-Definitionen eines Webservices.

Universal Description, Discovery and Integration (UDDI) ist ein Industrievorschlag für verteilte Web-basierende „Dienstekataloge“, die als Webservices bereitgestellt werden. Es basiert auf XML und SOAP und stellt ein Verzeichnis von Adress- und Produktdaten sowie Anwendungsschnittstellen der verschiedenen Webservices-Anbieter zur Verfügung.

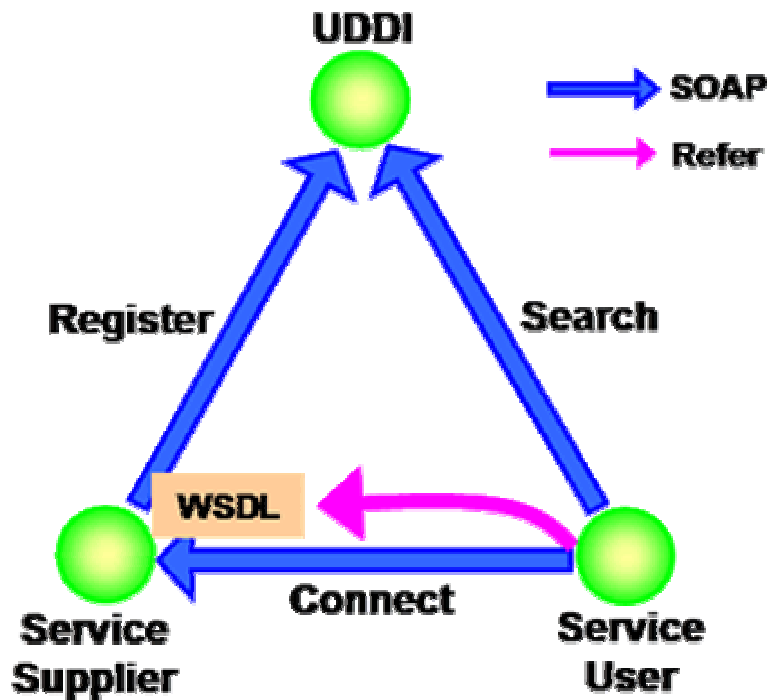


Abbildung 2: Kern-Standards für SOAP Webservices [15]

2.1 SOAP (Simple Object Access Protocol)

2.1.1 Definition

SOAP ist ein Netzwerkprotokoll zum Austausch von XML-basierten Nachrichten über ein Computernetzwerk. Es kodiert die Nachricht in XML und überträgt Aufrufe über standardisierte Internet-Technologien an die Methode des Webservices sowie das Resultat der Anfrage.

Als Transportmedium wird meistens HTTP verwendet, das unabhängig von der verwendeten Programmiersprache, dem Objektmodell sowie dem jeweiligen Betriebssystem ist.

2.1.2 Aufbau einer SOAP-Nachricht

SOAP-Messages enthalten einen **Envelope**, einen **Header** und einen **Body** (s. Abbildung 3):

- **SOAP Envelope:**
Die Hülle, die den Header und den Body umfasst. ...
- **SOAP Header:**
Optionales Element, das Attribute enthält, welche für die Verarbeitung der Nachricht wichtig sind, z.B. Informationen über das Routing der Nachricht; dazu enthält ein SOAP-Header Kontrolldaten und Direktiven, z.B. bzgl. Transaktionssemantik, Sicherheit, Zuverlässigkeit, Kodierung, etc.

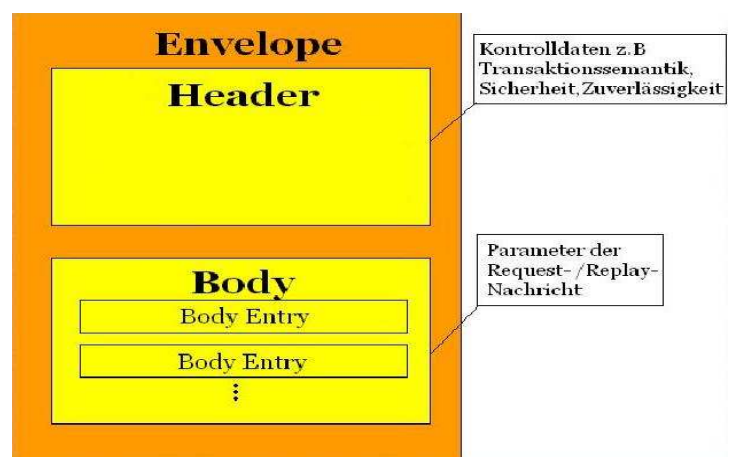


Abbildung 3: Aufbau einer SOAP-Nachricht [15]

- **SOAP Body**

Pflichtelement, das die Nachricht enthält, also die Informationen, welche an den Empfänger gesendet werden sollen (Parameter der Request- / Reply-Nachricht). Ein SOAP-Body muss als wohlgeformtes XML-Dokument vorliegen

2.2 WSDL (Web Service Description Language)

2.2.1 Definition

Die Web Service Description Language (WSDL) ist ein XML-Derivat zur Beschreibung der Schnittstelle eines Webservices und definiert die Nachrichtenstrom-Formate und Funktionsaufrufe. Das WSDL-Dokument beschreibt also im Wesentlichen, welche Methoden der Webservice anbietet, mit welchen Parametern sie aufzurufen sind, was sie zurückliefern und wie Kontakt zu meinem Webservices aufgenommen werden kann (s. Abbildung 4).



Abbildung 4: Arbeitsweise WSDL

Mit WSDL können Benutzer Proxys sprach- und plattformunabhängig für Webdienste erstellen. Im Gegensatz zu vergleichbaren Architekturen wie CORBA ist bei Webservices eine Schnittstellendefinition nicht zwingend erforderlich. Trotzdem ist diese sehr zu empfehlen, da das WSDL-Dokument alle wichtigen Informationen zum Aufruf eines Dienstes enthält. Aber wenn man die Struktur des Dienstes kennt, wird die Schnittstellendefinition aus Performanzgründen oft weggelassen.

2.3 UDDI (Universal Description, Discovery and Integration)

Um einen Service benutzen zu können, muss die Anwendung (als Service Consumer) dem Serviceanbieter bekannt sein. Dies wird durch die *Universal Description, Discovery and Integration* (UDDI) ermöglicht. Dabei kommunizieren drei Akteure miteinander. Der Service Consumer ist eine Anwendung, die einen Service benutzen möchte. Der Service Provider dagegen bietet einen Service an. Als Vermittler zwischen beiden steht die Service Discovery (UDDI), die ein Verzeichnis verfügbarer Services enthält.

UDDI ist eine weltweite Registrierungsstelle von Webservices und bezeichnet einen standardisierten Verzeichnisdienst (Veröffentlichung und Entdeckung) von Adress- und Produktdaten sowie Anwendungs-Schnittstellen der verschiedenen Webservices-Anbieter. Dabei soll dieser Verzeichnisdienst die zentrale Rolle in einem Umfeld von dynamischen Webservices spielen.

Der Verzeichnisdienst besitzt eine SOAP-Schnittstelle. Er enthält Unternehmen, ihre Daten und ihre Services.

2.4 Tool-Unterstützung

Die umfangreiche Tool-Unterstützung für SOAP-Webservices macht diese interessant. Dazu gehören Apache AXIS, XSUL, WebSphere Integration Developer, der WebSphere Studio Application Developer sowie der Rational Application Developer.

Apache AXIS ist eine Open-Source-Implementierung der SOAP-Webservices unter der Lizenz der Apache Software Foundation. AXIS kümmert sich um die Codierung und Decodierung von XML-Nachrichten. Dabei können automatisch WSDL aus Java-Code und Client-Klassen aus WSDL generiert werden. Der Entwickler kann sich ganz auf die Implementierung der Funktionen konzentrieren.

WS/XSUL v2 (auch bekannt als XSOAP4) ist eine modulare Java-Bibliothek, um Webservices zu erstellen, die XML verwenden. Dabei unterstützt XSUL XML-Schemata und WSDL 1.1 doc/literal-Bindings. XSUL bietet dadurch eine einfache Möglichkeit, SOAP-Webservices zu erstellen und unter der Voraussetzung, dass die Webservices in WSDL beschrieben sind und das XML-Infoset verwenden, anzusprechen.

Der WebSphere Integration Developer (WID) ist ein kommerzielles Produkt von IBM, der auf der Entwicklungsumgebung Eclipse basiert und die Source-Implementierung Apache AXIS sowie das eigene WebSphere Framework integriert.

Die WebSphere Studio Application Developer Integration Edition (WSADIE) ist der Nachfolger von „Visual Age for Java“ und basiert auf der freien Entwicklungsumgebung Eclipse mit vielen zusätzlichen Komponenten – unter anderem dem Java Visual Editor (JVE), der das Erstellen von AWT- und Swing-basierten Oberflächen gestattet, sowie Datenbankwerkzeugen.

Der „Rational Application Developer“ (RAD) ist eine umfassende, auf Eclipse basierende Entwicklungsumgebung, die Entwickler unterstützen soll, Web-, SOAP-, Java-, J2EE- und Portalanwendungen schneller zu entwerfen, entwickeln, analysieren, testen, anzupassen und einzurichten.

2.5 Beispielanwendung

Als Beispielanwendung soll ein Webservice-Client implementiert werden, der nach Packstation-Adressen und deren Map-Darstellung in YellowMap sucht.

Für diese Beispielanwendung wird Eclipse als Plattform verwendet, dazu das AWT Eclipse Plugin und der Apache AXIS. Als Webservicebeschreibung wurde ein WSDL-Dokument zur Verfügung gestellt (s. Anhang A1-1).

Mit Hilfe des AWT Eclipse Plugins und Apache Axis können automatisch aus dem WSDL-Dokument Java-Client-Klassen erzeugt werden:

Als erstes soll der Dialog Menu „New Web Service Client“ ausgewählt werden (s. Abbildung 5).

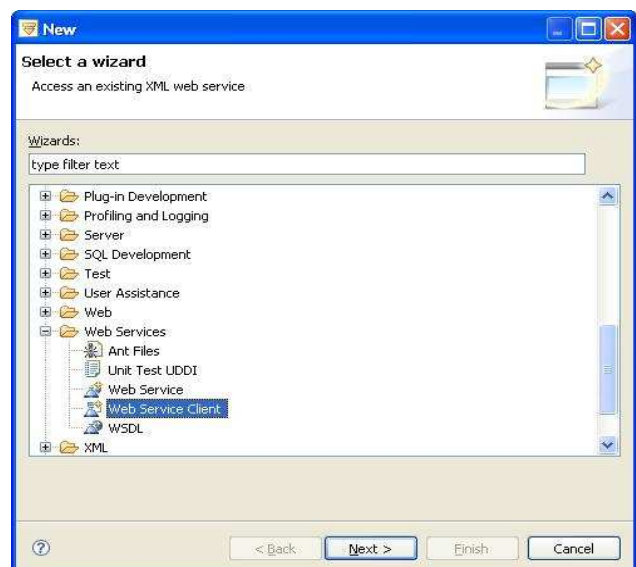


Abbildung 5: Erstellen eines Web Service Clients (1)

Im zweiten Schritt wird die Methode WSDLtoJava aus der Axis-Bibliothek verwendet, um aus der WSDL-Dokumentation die Javaklassen zu erzeugen. Dafür muss einfach der Pfad der WSDL-Datei bei Eclipse abgegeben werden (s. Abbildung 6).

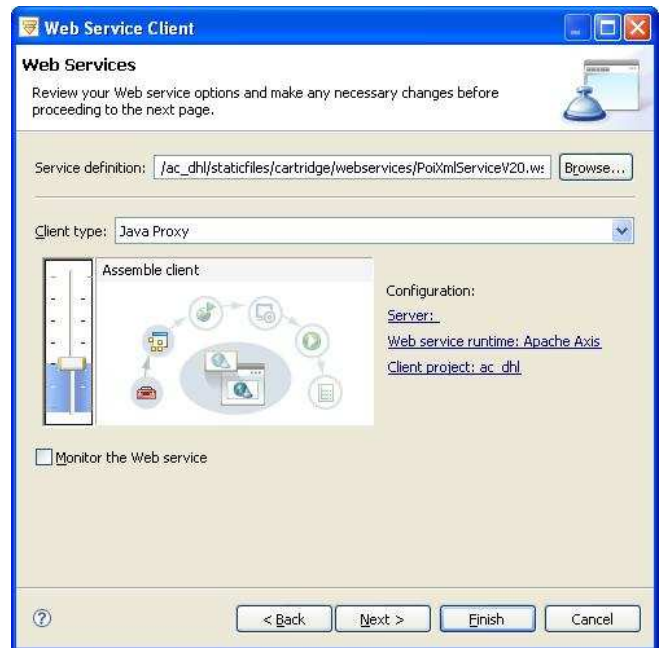


Abbildung 6: Erstellen eines Webservice Client (2)

Nach der automatischen Erzeugung der Javaklassen können die Methoden des Webservices einfach aufgerufen werden (s. Anhang A1-2):

```
CPoiXmlServiceSoap service = (new CPoiXmlServiceLocator()).getCPoiXmlServiceSoap();
CXmlDataPoiV20 serviceResult = service.poi(query);
```

Die Methode poi der SOAP-Schnittstelle erwartet einen einzelnen Wert („query“), der allerdings in Form eines CSV (mit '&' als Trennzeichen zwischen den Schlüssel-Wert-Paaren) beliebig viele Parameter enthalten kann. Der übergebene Wert gleicht also dem Querystring eines HTTP-GET und soll folgende Parameter enthalten:

- Country: Landeskennzeichen, z.B. D, CH, A
- Zip: Postleitzahl oder Teile der Postleitzahl
- Town: Stadt
- Town2: Stadtteil
- Street: Straße
- HouseNo: Hausnummer
- ...

serviceResult ist ein automatisch generiertes Objekt vom Type CPoiXmlServiceSoap und ist eine Liste von Packstation-Adressen und deren Map-Darstellung(Links)

Im Anhang befindet sich der komplette Code für diese Beispielanwendung (Anhang A1).

3. REST (Representational State Transfer)

3.1 Definition

REST (auch RESTful Webservices), ein Akronym für REpresentational State Transfer, stellt eine andere Herangehensweise zur Erstellung von Webservices im Vergleich zu SOAP und WSDL dar. Der Schwerpunkt liegt auf der Interaktion von zustandsbehafteten Ressourcen und gilt als einfacher zu handhaben als SOAP.

REST ist kein Standard, sondern ein Architekturstil, der die Idee des Webs darstellt; das Web ist demnach eine Ausprägung des REST-Architekturstils. Für die Umsetzung werden nur bereits seit langem verfügbare Internet-Technologien verwendet. Dabei beschreibt REST, wie Webstandards in einer webgerechten Weise eingesetzt werden können.

3.2 Merkmale einer REST-Anwendung

- **Namentliche Ressourcen:**

Das Grundkonzept von REST ist, dass jede Ressource mit einem URI (Uniform Resource Identifier) eindeutig identifizierbar ist. Eine Ressource kann jede Art von Information sein, die man eindeutig identifizieren kann. Das kann zum Beispiel ein Textdokument, ein Bild, eine Video- oder Audiodatei, eine Kollektion von mehreren anderen Ressourcen, eine Person etc. sein. Dabei wird diese Ressource über ihre URI adressiert und angesprochen [3].

- **Zugriff:**

Mit HTTP als Kommunikationsprotokoll können Nachrichten an die Ressource gesendet bzw. von dieser empfangen werden. REST hat die Semantik des HTTP-Protokolls übernommen [3]. Dabei wird die Ressource durch die Methoden GET, PUT, POST und DELETE bearbeitet. Ein HTTP GET wird empfohlen, wenn die Ressource durch Parameter identifiziert werden können, aber wenn dies kompliziert ist, dann wird ein HTTP POST empfohlen.

- **Zustandslos (Stateless):**

Nachrichten sind in REST selbstbeschreibend. Da der Server keinen Clientstatus verfolgt, muss in einer Nachricht alles enthalten sein, um die Anfrage zu interpretieren. Für die Interpretation einer Nachricht wird kein Wissen über vorherige oder spätere Nachrichten benötigt. Dabei ist es notwendig, dass die Nachricht den URI der Ressource enthält. Der statuslose Zustand eines Dienstes ermöglicht das parallele Laufen von Interaktionen und macht das Beheben von Fehlerzuständen leichter [7].

Um die Zuverlässigkeit und Skalierbarkeit der Dienste zu erhöhen, muss jede Anfrage alle Informationen beinhalten, weil Dienste in REST den Status einer Interaktion mit dem Client nicht speichern.

- **Präsentation:**

Die Präsentation einer Ressource kann auf unterschiedliche Weise geschehen. Diese Repräsentationen werden als Nachricht zwischen Client und Service ausgetauscht und können zum Beispiel XML-Dateien, Webseiten in unterschiedlichen Sprachen oder ein PDF sein [7].

- **Verlinkung von Ressourcen:**

Jede Repräsentation versetzt den Client in einen Zustand. Mögliche Operationen / Dienste werden als Links angeboten. Folgt der Client einem solchen Link, gibt eine weitere Repräsentation eine andere Ressource zurück, und so verändert er seinen Zustand [3]. Repräsentationen können auf weitere Ressourcen verweisen, die ihrerseits wieder Repräsentationen liefern, die wiederum auf Ressourcen verweisen können.

- **Cache:**

Caching wird unterstützt, um die Netzwerkkommunikation gering zu halten. Dabei kann der Server seine Antwort als cachefähig oder nicht-cachefähig kennzeichnen [7].

- „pull-based“-Interaktionen:

Die Kommunikation erfolgt auf Abruf. Konsumenten „ziehen“ vom Server Informationen. Dabei ist der Client aktiv und fordert vom passiven Server eine Repräsentation an bzw. modifiziert eine Ressource.

3.3 Rest- Dokumentwechsel mittels XLink

Mit jedem Dokument kann der Client tiefer in die Anwendung einsteigen. Als Einstiegspunkt reicht eine einzige URL aus. Die Idee der aus dem Web bekannten Hypertext-Dokumente wird so auf Webservices übertragen [3].

Folgende Abbildung beschreibt die Rest- Navigation über Links u. HTTP GET.

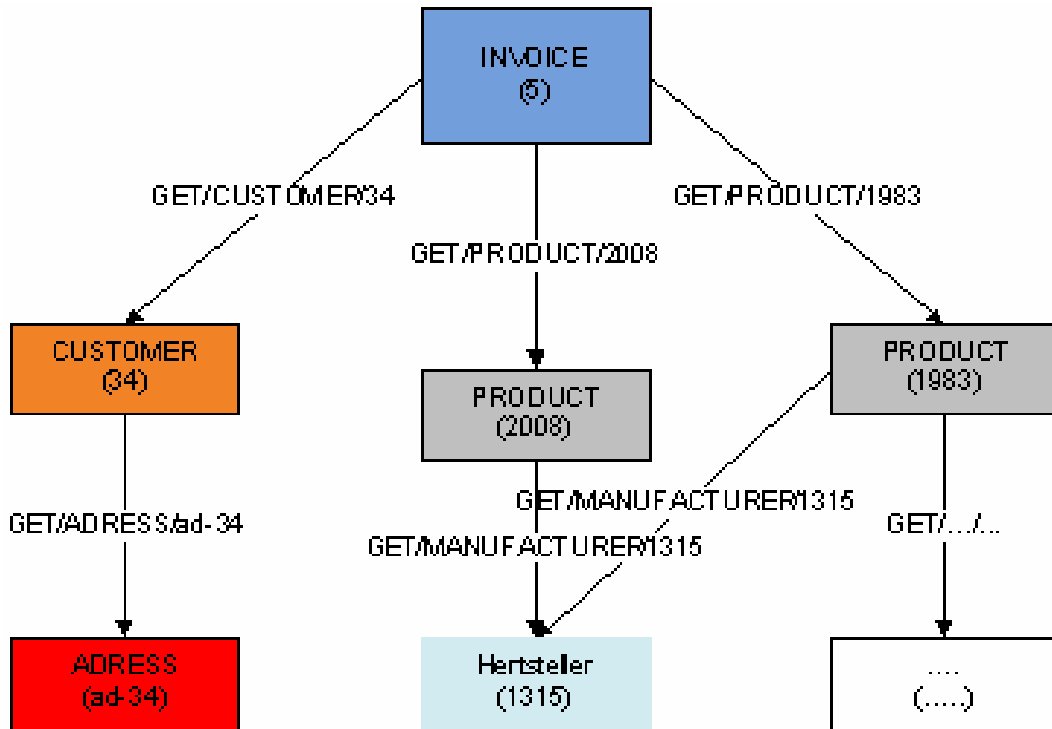


Abbildung 5: die Rest- Navigation über Links u. HTTP GET

Mit dem **Request** `GET/rest/INVOICE/5` (<http://localhost:8080/rest/INVOICE/5>) werden die Details von der Rechnung mit der id 5 angefordert, als **Response** kommt folgender XML-Code zurück:

```

<?xml version="1.0" encoding="UTF-8" ?>
<INVOICE xmlns:xlink="http://www.w3.org/1999/xlink">
  <ID>5</ID>
  <CUSTOMERID xlink: href="http://localhost:8080/rest/CUSTOMER/34/">34</CUSTOMERID>
  <POSITION nr="1" quantity="5">
    <PRODUCT xlink: href="http://localhost:8080/rest/PRODUCT/1983" nr="4501">
      <DESCRIPTION>Beschreibung Artikel 1</DESCRIPTION>
    </PRODUCT>
  </POSITION>
  <POSITION nr="2" quantity="2"> ..... </POSITION>
  <TOTAL>20.80</TOTAL>
</INVOICE>
  
```

Beispiel 1: HTTP GET & XLink

Mit **XLink** kann der Client z.B. die Kundendaten „`xlink: href="http://localhost:8080/rest/CUSTOMER/34/"`“ oder Produktdaten „`xlink: href="http://localhost:8080/rest/PRODUCT/1983"`“ anfordern.

3.4 Rest-Kommunikationsprotokoll

REST verwendet HTTP als Kommunikationsprotokoll. Dabei ist das Interface von REST generisch. Damit sich Client und Server verständigen können, müssen keine Protokoll-Konventionen bekannt sein. HTTP unterstützt einen festen Satz von vier Operationen, die von REST verwendet werden.

- **GET:**

Mit GET fordert der Client Daten oder Repräsentation vom Server an. Requests sollten frei von Seiteneffekten sein [7]. Da GET-Requests nur lesend auf dem Server zugreifen, können sie beliebig oft abgeschickt werden (s. Beispiel 1).

- **POST:**

Mit POST schickt der Client neu anzulegende oder anzupassende Daten an den Server. Beispielsweise könnte eine Adresse zu einem Profil hinzugefügt oder Preise in einem Warenkorb angepasst werden (s. Beispiel 2)[5]. POST ist nicht frei von Seiteneffekten. Beispielsweise können durch einen POST-Aufruf Felder in einer Datenbank verändert oder Prozesse auf dem Server gestartet werden.

Mit dem **Request POST/rest/PRODUCT/5** und die Ressource:

```
<resource>
  <price>17.50</price>
</resource>
```

Wird der Preis des Produktes mit der id 5 angepasst.

Beispiel 2

- **PUT:**

Mit PUT aktualisiert oder ersetzt der Client bereits existierende Daten oder Ressourcen auf dem Server (s. Beispiel 3).

Mit dem **Request PUT/rest/PRODUCT/** und die Ressource:

```
<resource>
  <id>6</id>
  <Name> test </test>
  <price>17.50</price>
</resource>
```

Wird ein Produkt mit der id 6 angelegt.

Beispiel 3

- **DELETE:**

Mit DELETE löscht der Client Daten oder Ressourcen auf dem Server (s. Beispiel 4).

Mit dem **Request DELETE/rest/PRODUCT/5/** Wird ein Produkt mit der id 5 gelöscht.

Beispiel 4

Jede REST-Ressource besitzt über die http-Methoden GET, POST, PUT und DELETE eine generische Schnittstelle. Mit diesen vier Methoden können die meisten Anwendungsfälle abgedeckt werden. Viele Anwendungen, die SQL verwenden, benutzen auch nur die generischen Befehle SELECT, INSERT, UPDATE und DELETE.

3.5 Vorteile von REST

3.5.1 Skalierbarkeit

Da jede Anfrage alle Informationen beinhaltet, müssen Dienste in REST den Status einer Interaktion mit dem Client nicht speichern, auch bei einem Wechsel von einem Server zu einem anderen muss kein Status ausgetauscht werden, und somit können Dienste Speicherplatz sparen. Dies hat positive Auswirkungen auf die Skalierbarkeit einer Anwendung.

REST basiert auf Standard-Webtechnologien und das enorme Wachstum des World Wide Web hat gezeigt, in welchem Ausmaß Webtechnologien skalieren [7].

Durch den Ansatz von Proxys und Caches wird die Performance erhöht. Dabei können Komponenten wie Server, Proxys und Web Anwendungen einzeln installiert und gewartet werden.

3.5.2 Zuverlässigkeit

Das Fehlen eines Zustandes erhöht ebenso die Zuverlässigkeit eines Dienstes, da dadurch der Wiederherstellungsprozess erleichtert wird [7]. Ein fehlgeschlagener Dienst kann schneller gestartet werden, weil keine Rücksicht auf einen Zustand genommen werden muss.

3.5.3 Anbindung von Fremdsystemen

In keiner anderen Anwendung sind so viele Legacy-Systeme wie im Web integriert. Über verschiedene Gateways kann auf eine Fülle von Systemen zugegriffen werden. Die Details von Fremdsystemen werden hinter Schnittstellen versteckt.

3.5.4 Unabhängig installierbare Komponenten

Die einzelnen Komponenten in einer REST-Anwendung sind unabhängig voneinander, das führt zur Stabilität der Kommunikation zwischen diesen Komponenten, dazu kann ein Deployment unabhängig durchgeführt werden, ähnlich wie im Web, wo auch der Inhalt einzelner Seiten ausgetauscht werden kann, ohne weitere Seiten anzupassen. Für sehr große Systeme, wie z.B. das World Wide Web oder den E-Mail-Dienst im Internet, ist ein unabhängiges Deployment eine Grundvoraussetzung [7].

3.5.5 Komposition von Diensten

Über den globalen und universellen Adressraum der URIs können die Grenzen einer Anwendung leicht überschritten werden. Mit HTTP gibt es keine Grenzen zwischen den Anwendungen. Ein Dokument verweist einfach auf eine Ressource, die sich in einer anderen Organisation befindet. Durch die Verfolgung dieses Links kann man, ohne es zu beabsichtigen, zu einer völlig anderen Anwendung gelangen. Genau genommen gibt es gar keine REST-Services. Es gibt nur Ressourcen, die zusammen genutzt werden können.

3.6 Wie realisiere ich einen REST-Webservice?

Das Ziel von einer REST-Anwendung ist, einem Client den Zugriff auf Funktionen oder Methoden entfernter Anwendungen, die auf Objekten arbeiten, zu ermöglichen. Dabei soll der Client über entfernte Methoden auf die Objekte des Servers zugreifen. Eine REST-Anwendung soll die Ressourcen identifizieren und ihre Objekte über URLs nach außen sichtbar machen. Schemas sollen für jede Art der Repräsentation angeboten werden [5].

Geschäftsobjekte müssen über eine URL erreichbar sein und mit einem Dokument repräsentiert werden können. Für dieses Dokument soll ein Format (d.h. welche Repräsentation die Ressource hat, z.B. HTML, HTTP, XML), vorzugsweise XML, ausgewählt werden. Für Verweise oder Dokumentenwechsel sollen Hyperlinks, vorzugsweise XLinks, verwendet werden.

Der Kern eines REST-Servers unterscheidet sich nicht von einer SOAP-Anwendung. Der Unterschied liegt in der Schnittstelle, die sich bei REST grundlegend von einer SOAP-Anwendung unterscheidet. REST verwendet die HTTP-Methoden, die auf über URL adressierbaren Ressourcen arbeiten, während RPC ein komponentenbasiertes Interface mit spezialisierten Methoden oder Funktionen zur Verfügung stellt. Die Methodensemantik soll dann beschrieben werden und die Response-Codes abgebildet werden (404, 403, 500...), von denen es in HTML schon für eigentlich jeden denkbaren Anwendungsfall einen gibt. (Bspw. "Methode ist nicht mehr unter dieser URI erreichbar" und neue URI ausgeben).

3.7 Tool-Unterstützung

Aufgrund seiner Einfachheit kann REST problemlos auch in allen Programmiersprachen verwendet werden, mit denen auch Internet-Anwendungen erstellt werden können. Für viele Programmiersprachen existieren aber bereits Tools wie z.B. HTTPClient oder RESTlet für Java oder httplib2 für Python (in Frameworks wie Grails für Groovy oder Rails für Ruby ist REST-Unterstützung bereits integriert).

Mit **Java** können Webservice-Clients ohne Zuhilfenahme von Tools programmiert werden. Dabei kann die Entwicklungsumgebung Eclipse benutzt werden. Ein Client muss für einen REST-Webservice-Aufruf einen HTTP-Request an den Server schicken und anschließend in der Lage zu sein, die Antwort zu interpretieren. Um die Anfrage zu senden, muss lediglich eine HTTP-Connection aufgebaut werden, der die abzufragende URL übergeben wird.

Der HTTPClient ist eine Open-Source-Entwicklung von Apache im Rahmen ihres Jakarta-Projekts. Der HTTPClient ist ein kleines Tool, das bei der Absetzung von HTTP-Anfragen mit Java unterstützen soll [7].

Restlet ist eine Java-Bibliothek, um den REST -Architekturstil in Java zu verwenden. Die Restlet-API unterstützt alle REST-Konzepte wie zum Beispiel Ressourcen, Repräsentationen, Daten, Konnektoren, Komponenten, etc. Sie kann sowohl für Client- als auch für Server-Webanwendungen eingesetzt werden.

3.8 Beispielanwendung

Als Beispielanwendung soll eine Client-Anwendung erstellt werden, die Packstation-Adressen anhand von Adressdaten vom Server anfordert.

Für diese Beispielanwendung wird Eclipse als Entwicklungsplattform verwendet, HTTPClient für die HTTP-Anfragen und JAXB, um das zurückgegebene XML-Dokument zu interpretieren.

Zuerst wird eine neue Instance vom HTTPClient erstellt:

```
HttpClient httpClient = new HttpClient();
```

- Danach soll die Request-Methode (POST / GET /..) definiert werden, in diesem Fall POST

```
PostMethod method = new PostMethod(„Webservice_URL“);
```

- Nach der Methodendefinition können Request-Parameter definiert werden:

```
List<NameValuePair> params = new ArrayList<NameValuePair>();
params.add(new NameValuePair(„ADDR_RECV_ZIP“,„20146“));
params.add(new NameValuePair(„ADDR_RECV_STR“,„Sedanstr. 23“));
....
```

- Die definierten Parameter sollen in der „RequestBody“ der „PostMethod“ gesetzt werden:

```
method.setRequestBody(params.toArray(new NameValuePair[0]));
```

- Jetzt kann der Request abgeschickt werden:

```
httpClient.executeMethod(method);
```

- Die Response kann als InputStream ausgelesen werden:

```
InputStream is=method.getResponseBodyAsStream();
```

- An dieser Stelle soll der Client den Response interpretieren:
 - Zuerst wird ein xsd-Schema benötigt, das den Response des Services darstellt (s. Anhang A2-1).
 - Aus dem xsd-Schema sollen Javaklassen mit Hilfe von JAXB automatisch generiert werden (s. Anhang A2-2).
 - Jetzt kann der Response interpretiert und in Java-Objekte umgewandelt werden:

```
ObjectFactory of = new ObjectFactory();
JAXBContext jc = JAXBContext.newInstance(of.getClass().getPackage().getName());
Unmarshaller unmarshaller = jc.createUnmarshaller();
AddressList addressList = (AddressList) unmarshaller.unmarshal(is);
```

Somit hat der Client die angeforderte Liste vom Server bekommen und interpretiert.
Im Anhang befindet sich der komplette Code für diese Beispielanwendung (s. Anhang A2).

Die Entwicklung eines SOAP Webservices ist deutlich einfacher als ein auf REST basierten Webservice. Dabei kann die WSDL-Datei dem Entwickler helfen, den Webservice und seine Funktionalität zu verstehen. Das Axis Werkzeug generiert automatisch Methoden, bei denen die entsprechenden Parameter gefordert werden. Dies grenzt bereits einige Fehlerquellen ein. Deshalb ist hier eine ausführliche Dokumentation nicht so wichtig. Im Gegensatz zu REST, ist eine gute Dokumentation seitens der Webservice Provider unbedingt erforderlich, damit der Entwickler versteht, wie er seine HTTP-Anfrage aufbauen muss.

4. REST vs. SOAP

Die **SOAP-Schnittstelle** hat eine „statische Typisierung“, dabei werden die SOAP-Operationen und ihre Parameter in WSDL beschrieben. Client und Server werden aneinander über WSDL verbunden. Damit hängen die Clients immer von der kompletten Schnittstelle eines Services ab, auch wenn sie bestimmte Operationen oder Parameter nicht benötigen. Das hat aber auch der Nachteil, wenn eine neue Version der WSDL auf dem Server installiert werden soll, dann müssen alle Clients sofort und zeitgleich auf die neue Schnittstellenversion migriert werden. Daher behilft man sich im SOAP-Umfeld damit, bei jeder Schnittstellenänderung die alte Schnittstelle unverändert bestehen zu lassen und die geänderte Version als neue zusätzliche Schnittstelle anzubieten.

Im Gegensatz dazu ist die **REST-Schnittstelle** eine „dynamische Schnittstelle“ und kann einfach erweitert oder angepasst werden (Hinzufügen weiterer Felder ist z.B. einfach möglich). Die Clients verbinden sich nur mit den Teilen der Schnittstelle, die sie wirklich benötigen. Daher müssen Clients nicht angepasst werden, wenn der Server z.B. zusätzliche Parameter unterstützt oder wenn der Server mehr Felder in den Rückgabe-Nutzdaten liefert.

Damit lassen sich viele Änderungen an REST-Schnittstellen durchführen, ohne dass alle Clients auf einmal angepasst werden müssen.

Bei dem **Zugriff** auf Ressourcen kann **SOAP** durch viele Firewalls nur komplett erlaubt oder verboten werden. Dafür muss der Inhalt der SOAP-Nachricht überprüft werden.

Im Gegensatz dazu ist bei **REST** eine **Zugriffssteuerung** durch Firewalls möglich (über Zugriffsmethode und URL).

Folgende Tabelle zeigt einen detaillierten Vergleich der beiden Technologien:

	SOAP	REST
Schnittstellenbeschreibung	WSDL	keine, generisch
Adressmodell	URI	URI
Schnittstelle	anwendungsspezifisch	generisch (GET, POST...)
Discovery	UDDI	generische Schnittstellen u. Verweise
Status	(Server) / Client	im Client
Akzeptanz in der Industrie	hoch	unbewusst
Werkzeuge	unzählige	wenige spezielle, viele Standardtools
Transport	HTTP, SMTP, JMS...	HTTP
Standards	W3C u.a.	(ist ein Architekturstil)
Performance	< 30 ms	< 3 ms bis beliebig

Tabelle 1: REST vs. SOAP

5. Fazit

REST ist einfach zu erlernen und umzusetzen und funktioniert auch in einem maximal heterogenen Umfeld. Dabei bietet REST eine maximale Performance und Skalierbarkeit für sehr viele parallele Benutzer. Als Protokoll werden ausschließlich Standards wie HTTP oder URIs verwendet. Dazu macht sich REST die Ideen des Webs wie Hypertext und einen unbegrenzten globalen Adressraum zunutze. Damit sind REST-basierte Webservices wirkliche *Webservices* im Gegensatz zu SOAP-Webservices.

Die umfangreiche Tool-Unterstützung für **SOAP**-Webservices macht sie interessant, sehr mächtig, aber auch entsprechend komplex, besonders wenn es um Prozess-Steuerung und -Monitoring geht. Verteilte Transaktionen zwischen den Systemen werden bei SOAP-Webservices unterstützt.

Wenn im Unternehmen verteilte Transaktionen zwischen den gekoppelten Systemen benötigt werden und die Einzelsysteme diese auch unterstützen oder wenn es um Prozess-Steuerung und -Monitoring geht, spricht das für **SOAP**. Aber wenn eine maximale Performance und Skalierbarkeit für sehr viele parallele Benutzer oder in einem maximal heterogenen Umfeld erreicht werden sollen, dann stellt sich **REST** als beste Lösung dar.

Die Entscheidung für **REST** oder **SOAP** ist sicher nicht einfach. Aber es gibt sowohl für SOAP als auch für REST bevorzugte Einsatzmöglichkeiten. Zum einen bieten beide Technologien eine lose, programmiersprachenunabhängige Kopplung von Softwaresystemen. Zum anderen setzen die REST- bzw. SOAP-Schnittstellen auf den Softwaresystemen auf und sind nicht bis tief in deren Kern integriert. Es ist daher verhältnismäßig einfach, beide Schnittstellen parallel zu unterstützen. Letztlich sind REST und SOAP keine Konkurrenten – das eigentliche Problem ist HTTP.

Quellen

- [1] Web Services Platform Architecture, Weerawarana u.a., Prentice Hall PTR (2005)
- [2] http://www.developer.com/java/web/article.php/10935_2207371_1

(letzter Zugriff: 15.06.08)

- [3] <http://www.oio.de/public/xml/rest webservices.htm> (letzter Zugriff: 15.06.08)
- [4] <http://www.extreme.indiana.edu/xgws/xsul/> (letzter Zugriff: 15.06.08)
- [5] <http://www.oio.de/m/soap-xmlrpc-rest/index.htm> (letzter Zugriff: 15.06.08)
- [6] <http://msdn.microsoft.com/vstudio/express/default.aspx> (letzter Zugriff: 15.06.08)
- [7] http://elib.uni-stuttgart.de/opus/volltexte/2006/2618/pdf/FACH_0056.pdf
(SOA vs. REST basierend auf Google, eBay, Amazon, Yahoo!)
(letzter Zugriff: 15.06.08)
- [8] <http://ws.apache.org/axis/java/reference.html> (letzter Zugriff: 15.06.08)
- [9] <http://jakarta.apache.org/commons/httpclient/> (letzter Zugriff: 15.06.08)
- [10] <http://www.restlet.org/> (letzter Zugriff: 15.06.08)
- [11] <http://xins.sourceforge.net/> (letzter Zugriff: 15.06.08)
- [12] <http://crispy.sourceforge.net/> (letzter Zugriff: 15.06.08)
- [13] <http://crispy.sourceforge.net/index.html> (letzter Zugriff: 15.06.08)
- [14] Web Services – Einsatzmöglichkeiten für das Information Retrieval im WWW
Fabio Tosques & Philipp Mayr
- [15] <http://www.nec.co.jp/middle/WebOTX-e/function/webservice.html>
(letzter Zugriff: 15.06.08)
- [HiWS] Hiccup In Web Services Adoption? 04/14/2005 By Jean-Christophe Cimetiere,
TechMetrix Research