9 Webservice als Standard-Nachrichtenformat

Webservices sind Programme, die mit Hilfe von standardisierten Internetprotokollen, z. B. HTTP, SMTP oder SOAP, von anderen Computern über ein Netzwerk zugreifbar sind.

La Grandeur d'un métier est avant tous d'unir les hommes.
Die Größe eines Berufs ist vor allem, die Menschen zu einen.
Antoine de Saint Exupéry, Terre des Hommes (II-2)

Entscheidung für ein Protokoll

In ungezählten Schnittstellenszenarien, zum Beispiel bei EDI-Übertragungen, standen die beteiligten Partner wie der Esel vor dem Heuhaufen und befassten sich mit unerheblichen Details der Übertragung, zum Beispiel, ob etwa ein bestimmtes Feld übertragen werden soll, ob die Kundennummer mit neun oder achtzehn Stellen gesendet wird oder ob Zahlen mit führenden Nullen oder linksbündig gesendet werden.

Universelles Schnittstellenformat

Webservices legen aufbauend auf XML und dem bereits für das WWW als Standard etablierten HTTP-Protokoll ein universelles Schnittstellenformat fest, das sowohl vom Sender als auch vom Empfänger der Schnittstellennachrichten verstanden wird. Daraufhin sendet der Requestor eine Anfrage via Internet an den Service, der wiederum verarbeitet die Nachricht und sendet eine Antwort zurück an den Requestor.

Webservices sind RPC-Aufrufe via HTTP

Das wesentlich Neue an Webservices ist die *Standardisierung*, denn nicht die Technologie ist neu, sondern die Tatsache, dass man sich auf ein einheitliches Übertragungsprotokoll geeinigt hat. Dieses Protokoll ist auch nicht verordnet, sondern entstand durch Übernahme einer nahe liegenden Konvention. In der einfachsten Interpretation sind Webservices RPC-Aufrufe über das HTTP-Protokoll.

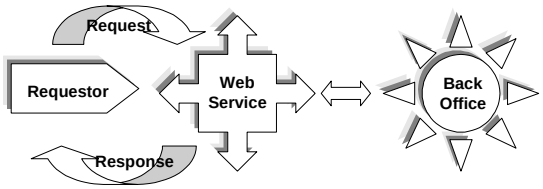

Abbildung 9.1 Interaktion mit Back-Office-Applikation via Webservices

Application Programming Interface

Stellt ein Software-Service eine Schnittstelle zur Verfügung, über die eine andere Software Anfragen an den Service stellen und Antworten entgegennehmen kann, so nennen wir das ein *Application Programming Interface* – API.

»WebAPI«

Im engeren Sinne ist ein Webservice eine Schnittstelle zu einem über TCP/IP aufrufbaren Programm, weshalb auch gelegentlich der Begriff *WebAPI* zu finden ist. In anderen Worten: Ist die API eines Software-Service durch eine Internetverbindung aufrufbar, so sprechen wir von einem Webservice.

9.1 Beispiel für Webservices mit Amazon.com

Seattle, Bundesstaat Washington, USA. Wir schreiben das Jahr 1994. Es ist das Gründungsjahr von Amazon.com, Inc. Amazon.com ist eines der ersten Unternehmen, die ausschließlich über das neue Medium *World Wide Web* einen Handel aufbauen.

Nur neun Jahre später – wir schreiben das Jahr 2003 – ist Amazon.com weltweit der größte Anbieter von Büchern, Musik und Videos im Internet. Die inzwischen mehr als acht Millionen Kunden in 160 Ländern können per Internet ihre Bestellungen aufgeben und sie sich bequem nach Hause bringen lassen.

Wenn heute jemand auf das Thema Internet stößt, dann assoziiert er damit fast automatisch diesen Pionier des eCommerce, Amazon.com. Kaum ein anderer individueller Webshop wurde so zur Legende und damit auch zum Synonym für eCommerce. In vorbildlicher Weise gelang es Amazon.com, Vertrauen und Akzeptanz zu schaffen für das neue Medium World Wide Web.

Amazon.com-Katalog als Webservice

Nun ist Amazon.com vor kurzer Zeit wieder einen Schritt weiter gegangen und bietet seine Dienste in vollautomatisierter Form an. Während man zwar auch schon bisher auf Amazon.com-Seiten durch Links auf der eigenen Homepage verweisen konnte, hat Amazon.com mittlerweile seinen Shop so standardisiert, dass Kataloganfragen, Katalogauskünfte und Einkäufe auch durch einfache, webfähige Computerprogramme möglich werden. Wir nennen so was gemäß unserer Definition einen Webservice.

9.1.1 Einfache Beispiele für Webservices

Bevor wir uns dem detaillierten Beispiel von Webservices mit Amazon.com widmen, wollen wir mit ein paar einfachen Beispielen beginnen, um ein Gefühl für Webservices im Allgemeinen zu bekommen.

Einfachste Form sind statische Webseiten

Die einfachste Form eines Webservices ist das Aufrufen einer Internetseite über einen Browser. Das Prinzip, das dahinter steckt, ist zunächst sehr, sehr einfach:

Der anfragende Requester sendet seinen Wunsch über ein TCP/IP-Netzwerk zu einem namentlich bekannten Server. Der Server sucht die gewünschte Datei und sendet das gefundene Dokument an den Requester zurück. Als Protokolle für diese Kommunikation werden heute fast ausschließlich HTTP und FTP verwendet.

Dynamische Webseiten mit Forms oder CGI-Parametern

Nachdem Sie sich Tausende von Webseiten angesehen haben, kommt die nächste Begehrlichkeit fast zwangsläufig: Die Seiten sollten dynamisch auf die Eingaben des Benutzers reagieren. Dazu muss normalerweise der URL ein Parameter mitgegeben werden, der vom Server dynamisch interpretiert werden kann. Das folgende Beispiel bestimmt die aktuelle UTC-Zeit (*Greenwich Mean Time*):

```
http://www.nanonull.com/TimeService/TimeService.asmx/
getUTCTime?
```

CGI-Standard hat sich durchgesetzt

Als Syntax zur Übermittlung von Parametern über eine URL hat sich der CGI-Standard durchgesetzt. Ursprünglich wurde CGI so verwendet, dass man den Namen eines physisch existierenden Programms angegeben hatte und die Parameter mit Fragezeichen (»?«) und Ampersand (»&«) getrennt an die URL anfügte.

9.1.2 Ein Beispiel mit einem winzigkleinen Fehler

Webservices am Beispiel von Amazon.com

Webservices mit Amazon.com eignen sich gut zum Lernen der Technologien, da der Service öffentlich zugänglich ist und die Inhalte allgemein verständlich sind. Das folgende Szenario kommuniziert mit *http://xml.Amazon.com*. Alle Bücher werden weltweit durch den eindeutigen Produktcode ISBN (*International Standard Book Number*) identifiziert. Amazon hat für seine internen Dienste die Bezeichnung ASIN (*Amazon Standard Item Number*) eingebaut, die neben Büchern auch die restlichen Artikel des Sortiments identifiziert. Für Bücher ist die ASIN identisch mit der ISBN. Ausgehend von einer bekannten ISBN, wollen wir die Webservices von Amazon.com nutzen, um die Katalogdaten des passenden Buchs zu ermitteln.

Im Folgenden sehen Sie, wie die Kommunikation aufgebaut ist. Dabei hat sich in dieses Beispiel ein kleiner syntaktischer Fehler eingeschlichen, der dazu führt, dass Amazon.com statt der gewünschten Katalogdaten nur eine Fehlermeldung zurücksendet. So eine Situation führt in einer klassi-

schen Schnittstellenentwicklung zu aufwändigen Nachfragen beim Kommunikationspartner oder zum Nachsuchen in Dokumentationen. Früher hätten wir also zum Telefon gegriffen, versucht, bei Amazon.com einen Ansprechpartner zu finden, der auf gut Glück den Grund des Fehlers analysiert. Das Beispiel wird zeigen, wie es uns gelingt, anhand der WSDL (*Web Service Description Language*) des Webservices den Fehler selbst zu finden.

ISBN-Abfrage bei Amazon.com via SOAP

Hinter dem folgenden XML-Dokument versteckt sich eine Suche nach einem Buch im Katalog von *Amazon.com*. Diese Anfrage wird durch SOAP (*Simple Object Access Protocol*) übermittelt. Um den Blick auf das Wesentliche zu lenken, schauen wir uns das Dokument zunächst in etwas bereinigter Form an, befreit von administrativen Informationen.

Listing 9.1 SOAP-Anfrage an Amazon.com

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <AsinSearchRequest>
      <AsinSearchRequest>
        <asin>3528057297</asin>
        <tag>logosworldcom</tag>
        <type>lite</type>
        <dev-tag>12345678901234</dev-tag>
      </AsinSearchRequest>
    </typens:AsinSearchRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 9.2 XML-Dokument in Baumdarstellung

```
AsinSearchRequest
       +- asin
       ¦         +- 3528057297
       +- tag
       ¦         +- logosworldcom
       +- type
       ¦         +- lite
       +- dev-tag
                 +- 00000000000000
```

Wenn alles gut geht, wird dieser SOAP-Request von dem empfangenden Webservice bei Amazon.com verarbeitet und das Ergebnis wieder in ein SOAP-Dokument, diesmal als SOAP-Response, verpackt und an den Requester zurückgeschickt.

Listing 9.3 SOAP-Response von Amazon.com

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <AsinSearchRequestResponse>
      <return>
        <Details>
           <Url>http://www.Amazon.com/
           exec/obidos/3528057297</Url>
           <Asin>3528057297</Asin>
           <ProductName>The SAP R/3 Guide to EDI and
           Interfaces</ProductName>
           <Authors>
             <Author>Axel Angeli</Author>
             <Author>Ulrich Streit</Author>
             <Author>Robi Gonfalonieri</Author>
           </Authors>
           <ListPrice>$50.95</ListPrice>
         </Details>
       </return>
    </AsinSearchRequestResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 9.4 SOAP-Response in Baumdarstellung

```
AsinSearchRequestResponse
       +- asin
       ¦          +- 3528057297
       +- Url
       ¦          +- http://www.Amazon.com/
                     exec/obidos/3528057297
       +- ProductName
       ¦          +- The SAP R/3 Guide to EDI
                     and Interfaces
```

```
+- Authors
¦           +- Author
¦           ¦       +- Axel Angeli
¦           +- Author
¦                   +- Robi Gonfalonieri
+- ListPrice
            +- $50.95
```

Der abgesandte SOAP-Request hat in dem Beispiel leider einen winzig kleinen Fehler, weshalb uns *Amazon.com* nur eine als SOAP-Response verpackte Fehlermeldung zurücksendet, die uns zunächst auch nicht weiterhilft.

Listing 9.5 SOAP-Error von Amazon.com

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>Bad Request</faultstring>
      <detail>AsinSearchRequest: There must be a keys
      named 'type' and 'asin' and 'devtag'in the input
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 9.6 SOAP-Error als Baum

```
SOAP-ENV:Fault
       +- faultcode
       ¦           +- 3528057297
       +- detail
       ¦           +- AsinSearchRequest: There
       ¦              must be a keys named
       ¦              'type' and 'asin' and
       ¦              'devtag'in the input
       ¦
       +- faultstring
                   +- detail
                          +- Robi Gonfalonieri
```

Dies ist die Hölle für jeden Schnittstellenprogrammierer: Der Kommunikationspartner am anderen Ende versteht meine Anfrage nicht und die gesendete Antwort gibt uns leider nicht den entscheidenden Hinweis. Denn im Beispiel hier haben wir die geforderten Parameter `type`, `asin` und `devtag` korrekt angegeben. Oder doch nicht?

Was war geschehen? Die Anfrage haben wir der Dokumentation von Amazon.com zu den Webservices entnommen. Um es schon vorwegzunehmen: In dem Beispiel befand sich ein winzig kleiner Fehler, der mit bloßem Auge kaum erkennbar ist. Da wir uns auf die sekundäre Dokumentation verlassen hatten, waren wir letztlich Opfer von Übertragungsfehlern oder mangelnder Aktualität.

Geht es so nicht den meisten Schnittstellenentwicklern? Eigentlich wohl definierte – weil in einer Programmiersprache festgelegte – Dateiformate werden einem EDI-Partner als Excel-Sheet, Word-Datei, gefaxte Textdatei oder auch gar nicht übermittelt. Fast immer endet es damit, dass die Bedeutung und Struktur einer Übertragungsdatei aus den übermittelten Daten heraus geraten wird.

WSDL gibt formale Beschreibung der Syntax

Da Amazon.com aber seine Dienste als vollständigen Webservice zur Verfügung stellt, haben wir Zugriff auf die Originalquelle in Form des zugrunde liegenden WSDL-Dokuments.

9.1.3 Die WSDL zum Amazon.com-Service

WSDL – Webservices Description Language

Die *WSDL* (*Web Services Description Language*) ist eine Sprache, die mit Hilfe von XML-Dokumenten die Struktur der Requests und Responses eines Webservices beschreibt. Die Beschreibung ist so ausführlich, dass man mit ihrer Hilfe die Struktur des erforderlichen SOAP-Request-Dokuments bestimmen kann.

WSDL muss öffentlich zugänglich sein

Das WSDL-Dokument wird an einer Stelle im Internet abgelegt, so dass die aktuelle Version jederzeit sowohl vom Sender als auch vom Empfänger des Dokuments zugreifbar ist. Außerdem verprobt der Empfänger eines SOAP-Dokuments dieses gegen das WSDL-Dokument, bevor es weiterverarbeitet wird. Das folgende Listing zeigt die WSDL-Beschreibung einer Amazon.com-Suche.

Listing 9.7 WSDL-Definition einer Suche nach ISBN bei *Amazon.com*

```
<definitions targetNamespace=
"urn:PI/DevCentral/SoapService" name="AmazonSearch">
  <types>
    <xsd:schema>
```

```
      <xsd:complexType name="AsinRequest">
        <xsd:all>
          <xsd:element name="asin" />
          <xsd:element name="tag" />
          <xsd:element name="type" />
          <xsd:element name="devtag" />
          <xsd:element name="offer" />
          <xsd:element name="offerpage" minOccurs="0"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:schema>
  </types>
  <message name="AsinSearchRequest">
    <part name="AsinSearchRequest"
    type="typens:AsinRequest"/>
  </message>
  <message name="AsinSearchResponse">
    <part name="return" type="typens:ProductInfo"/>
  </message>
</definitions>
```

Namen der erlaubten Bezeichner

Dem WSDL-Dokument kann man entnehmen, dass ein `AsinRequest`, das ist die Anfrage nach einer ISBN, die wir in dem SOAP-Dokument übermittelt haben, die Elemente `asin`, `tag`, `type`, `devtag`, `offer`, `offerpage` enthalten darf. Wir können damit das SOAP-Dokument Schritt für Schritt aufbauen.

Die Anfrage darf eine Message mit dem Namen `AsinSearchRequest` enthalten:

```
<message name="AsinSearchRequest">
    <part name="AsinSearchRequest"
    type="typens:AsinRequest"/>
</message>
```

Das SOAP-Dokument würde also schon wie folgt aussehen:

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <AsinSearchRequest>
      <AsinSearchRequest>
        . . .
```

```
      </AsinSearchRequest>
    </typens:AsinSearchRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Beispiel einer Typisierung einer Variablen

In der WSDL wurde die Message mit `name="AsinSearchRequest"` als vom `type="typens:AsinRequest"` festgelegt. Die Typdefinition wurde weiter vorne in der WSDL vorgenommen:

```
<xsd:complexType name="AsinRequest">
  <xsd:all>
    <xsd:element name="asin" />
    <xsd:element name="tag" />
    <xsd:element name="type" />
    <xsd:element name="devtag" />
    <xsd:element name="offer" />
    <xsd:element name="offerpage" minOccurs="0"/>
  </xsd:all>
</xsd:complexType>
```

Also lassen wir das SOAP-Dokument wachsen:

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <AsinSearchRequest>
      <AsinSearchRequest>
        <asin>3528057297</asin>
        <tag>logosworldcom</tag>
        <type>lite</type>
  <devtag>12345678901234</devtag>
      </AsinSearchRequest>
    </typens:AsinSearchRequest>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP lässt sich aus WSDL generieren

Somit haben wir den SOAP-Request aus der WSDL abgeleitet. Selbstverständlich kann und soll im Normalfall sogar die Transformation von WSDL in ein SOAP-Dokument durch ein Programm erfolgen. Hierzu stehen mittlerweile eine Vielzahl von SOAP-Prozessoren zur Verfügung, die alle mehr oder weniger das Gleiche tun:

- **beim Senden eines SOAP-Dokuments**
  - Generieren einer SOAP-Template aus einer WSDL
  - alternativ das Generieren von Programmcode aus einer WSDL
  - Abmischen von Anwendungsdaten mit dem SOAP-Template
- **beim Empfangen eines SOAP-Dokuments**
  - Verifizierung des SOAP-Dokuments anhand der WSDL
  - Extraktion der Programmparameter aus dem Dokument
  - Aufruf des Programms zur Ausführung des gewünschten Services
  - Entgegennehmen der Programmergebnisse und Rücksendung an den Requester, wieder als eigenständiges SOAP-Dokument

Und warum hat unsere Anfrage eine Fehlermeldung zurückgegeben? Vergleichen wir dazu das ursprüngliche SOAP-Dokument und die konstruierte Anfrage. Das hatten wir ursprünglich gesendet:

```
<dev-tag>12345678901234</dev-tag>
```

Und das hätte dort stehen müssen:

```
<devtag>12345678901234</devtag>
```

9.1.4 Webservice als HTTP GET-Anfrage

Im Übrigen hätten wir die Anfrage an Amazon.com auch als `HTTP GET`-Statement ausführen können. Das hätte dann so ausgesehen:

Listing 9.8 Amazon-Webservice als Canonical URL

```
http://xml.Amazon.com/onca/xml2?t=webservices-20
&tag=logosworldcom&dev-t=D2H3YO46KJJ615&AsinSearch
=3528057297&type=lite&f=xml
```

9.1.5 Verwendung eines XSLT-Stylesheets

Wenn Sie das Ergebnis lieber nach eigenen Wünschen formatiert zurückhaben wollen, können Sie anstatt des Parameters `f=XML` auch die URL eines XSLT-Stylesheets angeben. In diesem Fall wird dieses XSLT auf das XML-Dokument angewendet und Sie erhalten den transformierten Ausgabestrom zurück. Das XSLT-Stylesheet muss von Ihnen im Internet bereitgestellt werden und sowohl vom rufenden Browser als auch vom Amazon-Webservice zugänglich sein.

Listing 9.9 Anwendung eines XSLT-Stylesheets auf Amazon-XML

```
http://xml.Amazon.com/onca/xml2?t=webservices-20
&tag=logosworldcom&dev-t=D2H3YO46KJJ615&AsinSearch
=3528057297&type=lite&f=http://logosworld.com/bookshop/
amazon.xsl
```

9.1.6 Type- und Namespace-Referenz

Was in den Abbildungen weiter oben fehlte, waren die Referenzen auf Namespace und die Angabe der Datentypen. Mit den Datentypen verhält es sich wie in jeder anderen Programmiersprache auch. Die Namespaces sind komplizierter zu erfassen. Grob gesagt bedeutet es, dass jedes XML-Dokument zu einem Namespace gehören muss. Dieser Namespace ist selbst ein XML-Dokument, das die zulässigen Namen der XML-Tags beschreibt. Damit der Empfänger des XML-Dokuments die Gültigkeit des Dokuments prüfen kann, muss die URI des Namespace mit jedem XML-Dokument gesendet werden. Ob der Empfänger das Dokument tatsächlich gegen den Namespace verprobt, ist zunächst einmal dessen Sache.