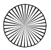
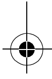
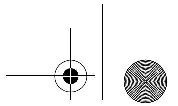
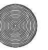
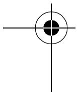
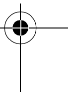





11 XML – Extended Markup Language

XML sind einfache ASCII-Dokumente zum Austausch von Daten zwischen zwei Kommunikationspartnern. XML und die darauf aufbauenden Standards SOAP und WSDL haben sich mit rasanter Geschwindigkeit zum De-facto-Standard für verteilte Programmierung entwickelt.

11.1 XML als Lingua franca des Internet

XML wurde als die Lingua franca des Internet bekannt, also als eine Sprache, die jeder spricht und jeder versteht, ohne dass die Verwendung durch eine Organisation vorgeschrieben wurde. Damit ist XML zu etwa dem geworden, was die englische Sprache für die Global Economy ist.

XML ist pures ASCII

XML ist eine einfache, gleichzeitig menschen- und maschinenlesbare Sprache. Dokumente in XML haben einen hierarchischen Aufbau, bestehen aber ausschließlich aus druckbaren Zeichen und verwenden ASCII-Code zur Übertragung. XML ist HTMLsehr ähnlich, wobei XML aber strengere Anforderungen an die Struktur des Dokuments stellt.

Einfaches Parsing und Debugging

Während die strenge Strukturierung es einfach macht, Parser für die Dokumente zu schreiben, garantiert die Verwendung von ASCII, dass die Dokumente im Falle von Störungen, auch von einem Menschen ohne weitere Hilfsmittel gelesen und analysiert werden können.

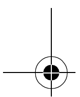
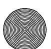
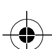
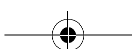
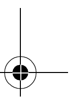
Das nachfolgende Beispiel zeigt, wie eine gewöhnliche zwei-dimensionale Tabelle als XML-Dokument dargestellt werden kann. Die Kreuztabelle in Tabelle 1.1 kann in einem XML-Dokument wie in Tabelle 1.1 gezeigt repräsentiert werden.

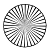
Name	Weight	Gender
Cow	420	F
Pig	120	M

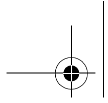
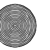
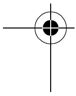
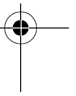
Tabelle 11.1 Einfache Kreuztabelle einer Animal-Farm

Listing 11.1 Beispiel einer einfachen XML-Farm

```
<?xml version="1.0" encoding="UTF-8"?>
<Farm xmlns="x-schema:U:\examples\XML\farmschema.xml">
  <Animal>
```



```
    <Name>Cow</Name>
    <Weight>420</Weight>
    <Gender>F</Gender>
  </Animal>
  <Animal>
    <Name>Pig</Name>
    <Weight>120</Weight>
    <Gender>M</Gender>
  </Animal>
</Farm>
```

11.1.1 Kleine Anatomie eines XML-Dokuments

In einer traditionellen IT-Umgebung hätten wir wahrscheinlich ein Dokument ähnlich dem folgenden erzeugt:

```
Cow     000420F
Pig     000120M
```

Flachen Dateien fehlen Informationen über die Struktur

Das Problem mit einer solchen Datei ist offensichtlich: Wenn Sie die Datei bekommen, wissen Sie zunächst nichts oder nicht viel über deren Struktur, weshalb die Datei ohne zusätzliche Informationen weitgehend nutzlos bleibt. Wir wissen nicht, wo die Kolonnen beginnen und wo sie enden, und auch nicht, welche Bedeutung ihnen zukommt. Häufig fehlt genau diese Dokumentation aus vielerlei Gründen und wir müssen aus dem Inhalt raten, wie die Datei aufgebaut ist.

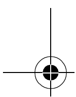
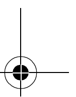
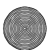
Strukturinformationen werden hinzugefügt

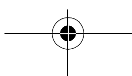
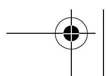
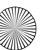
Aus diesem Grund packen wir die Daten in eine harmonische Struktur, indem wir um jedes Feld und jeden Satz jeweils ein öffnendes und ein schließendes Tag packen. Im folgenden Beispiel sind die Felder jedes Datensatzes von den Feldnamen umrahmt.

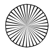
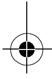
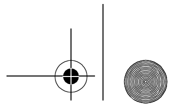
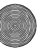
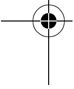
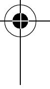
```
<Name>Cow</Name><Weight>420</Weight><Gender>F</Gender>
<Name>Pig</Name><Weight>120</Weight><Gender>M</Gender>
```

Nur druckbare Zeichen sind erlaubt

Beim Senden von Daten durch ein elektronisches Medium bestehen eine Reihe von Restriktionen. So ist es zum Beispiel nicht möglich, Steuerzeichen wie Carriage-Return <CR> oder Line-Feed <LF> beliebig in den Datenstrom eines HTTP-Dokuments einzufügen, denn diese Zeichen werden bereits als Sonderzeichen für das Protokoll selbst verwendet. Zwei aufeinander folgende <CR><LF>, also zwei Zeilenschaltungen hintereinander, signalisieren das Ende eines Abschnitts, in der Regel das Ende des

HTTP-Headers oder des ganzen Dokuments. Manche Protokolle erlauben auch nur eine begrenzte Anzahl von Zeichen innerhalb einer Zeile, was aber bei HTTP nicht zutrifft.

Zeilen werden durch ein Tag markiert

In einem XML-Dokument umgeht man dieses Problem von vornherein, indem man Zeilen auch wieder durch ein öffnendes und ein schließendes Tag markiert. Im Beispiel stellt jede Zeile ein Tier da, also bekommt es als Klammer das Tag <Animal>.

```
<Animal><Name>Cow</Name><Weight>420</Weight>
<Gender>F</Gender></Animal>
<Animal><Name>Pig</Name><Weight>120</Weight>
<Gender>M</Gender></Animal>
```

Um das Ende eines Dokuments sicher erkennen zu können, erfordert XML noch, dass das erste Tag als Envelope (Rahmen) für das ganze Dokument gewertet werden muss. Dieses Tag darf auch kein weiteres Mal im ganzen Dokument vorkommen.

```
<Farm>
  <Animal><Name>Cow</Name><Weight>420</Weight>
  <Gender>F</Gender></Animal>
  <Animal><Name>Pig</Name><Weight>120</Weight>
  <Gender>M</Gender></Animal>
</Farm>
```

Metainformationen für den XML-Parser

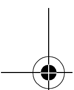
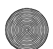
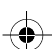
Zuletzt fügen wir dem XML-Dokument noch ein paar administrative Informationen hinzu. In dem Tag `<?xml>` können Informationen als Attribute mitgegeben werden, die für den Parser des Dokuments gegebenenfalls von Interesse sein könnten. In unserem Beispiel wird die XML-Version mitgegeben, mit der das Dokument konform geht, und der Name der Zeichensatzvariante, die in dem Dokument verwendet wird. Mögliche Zeichensätze sind unter anderem UTF-8, ANSI oder WINDOWS.

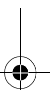
```
<?xml version="1.0" encoding="UTF-8"?>
```

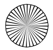
Namensraum

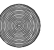
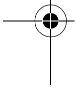
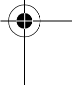
Die Spezifikation von XML erfordert, dass die Struktur des Dokuments verifizierbar ist. Dazu wird das erhaltene Dokument mit einem Schema-Dokument verglichen. Dieses Referenzdokument nennt man einen Namespace, deshalb auch der Name des Attributs `xmlns` für XML-Namespace.

```
<Farm xmlns="x-schema:U:\examples\XML\farmschema.xml">
```



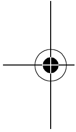
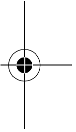


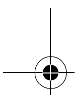




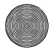
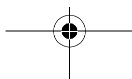
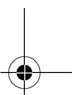
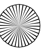
Das folgende Listing zeigt exemplarisch die Anatomie eines XML-Dokuments *farmschema.xml*. In Abbildung 11.1 ist sie nochmals grafisch aufbereitet.

Listing 11.2 Anatomie eines XML-Dokuments

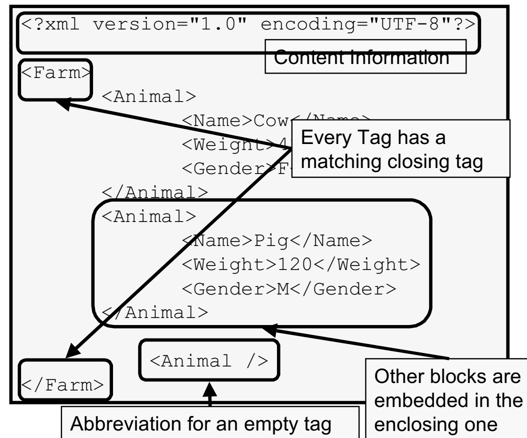
```
<?xml version="1.0" encoding="UTF-8"?>

<!--W3C Schema generated by XML Spy v3.0.7 NT (http://
www.xmlspy.com)-->
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
        xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="Name" content="textOnly"
  dt:type="string"/>
  <ElementType name="Weight" content="textOnly"
  dt:type="int"/>
  <ElementType name="Gender" content="textOnly"
  dt:type="string"/>
  <ElementType name="Family" content="textOnly"
  dt:type="string"/>

  <ElementType name="Animal" content="mixed">
    <element type="Name" minOccurs="1" maxOccurs="*"/>
    <element type="Family" minOccurs="1"
    maxOccurs="*"/>
    <element type="Weight" minOccurs="1"
    maxOccurs="*"/>
    <element type="Gender" minOccurs="1"
    maxOccurs="*"/>
  </ElementType>

  <ElementType name="Farm" content="mixed">
    <element type="Animal" minOccurs="1"
    maxOccurs="*"/>
  </ElementType>
</Schema>
```



Kap11.fm Seite 209 Dienstag, 22. April 2003 2:33 14



Hierarchy Represented by Above XML Document

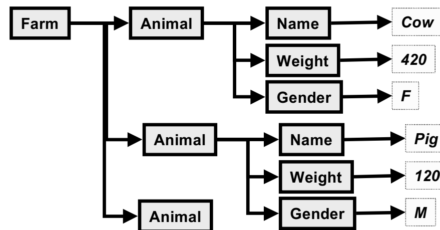


Abbildung 11.1 Anatomie eines XML-Dokuments

11.2 XSD-Schema und DTD

XSD, XML-Schema-Definition

XSDs – *XML-Schema-Definition* - sind XML-Dokumente, die den erlaubten Inhalt und die Syntax anderer XML-Dokumente formal beschreiben. Zwar sind XSDs nicht zwingend, aber sehr hilfreich bei der Bestimmung formaler Aspekte eines Dokuments, zum Beispiel einer Datenbanktabelle. Darüber hinaus helfen XML-Schemen bei geschickter Anwendung auch, die »Geschwätzigkeit« von XML-Dokumenten zu reduzieren.

Validierung

Ein XML-Dokument, dem ein Schema zugeordnet ist, kann gegen dieses verprobt werden, man nennt dies eine *Validierung*.

Grob gesprochen, ist ein XML-Schema wie ein Eintrag in einem Data-Dictionary. Es legt fest, wie die Datentypen der Elemente und Attribute aussehen, wie die Namen der Tags lauten und welche Datentypen die Werte der Elemente annehmen dürfen.

Aufbau eines Schemas

Ein XML-Schema ist selbst ein gültiges XML-Dokument und verwendet dafür eine Reihe von vordefinierten Tags, die als Teil der XML-Sprache festgelegt wurden. Da ein XML-Schema selbst ein XML-Dokument ist,

kann es genau genommen wiederum gegen ein anderes Dokument validiert werden. Das Dokument, das die Validierungsinformation für ein Schema enthält, beschreibt die erlaubten Tags des Schemas. Dies wird vor allem benutzt, um die Verprobung gegen eigene Datentypen oder Datenstrukturen zu ermöglichen, zum Beispiel kann bestimmt werden, ob die Datenbank einen Type `Float` unterstützt.

Listing 11.3 XSD – XML-Schema zur Animal-Farm

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsd:schema PUBLIC "-//W3C//DTD XMLSCHEMA
19991216//EN" "" [
  <!ENTITY % p 'xsd:'>
  <!ENTITY % s ':xsd'>
]>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XML-
Schema">
  <xsd:complexType name="TyFamily"
  content="elementOnly">
    <xsd:sequence>
      <xsd:element name="TyName">
        <xsd:simpleType base="xsd:string">
          <xsd:enumeration value="Cow"/>
          <xsd:enumeration value="Pig"/>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="TyGender"
  content="elementOnly">
    <xsd:sequence>
      <xsd:element name="TyGender">
        <xsd:simpleType base="xsd:string">
          <xsd:enumeration value="F"/>
          <xsd:enumeration value="M"/>
        </xsd:simpleType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
```

```
  <xsd:complexType name="AnimalType" content="elemen-
tOnly">
    <xsd:sequence>
      <xsd:element name="Family" type="TyFamily"/>
      <xsd:element name="Weight" type="xsd:short"/>
      <xsd:element name="Gender" type="TyGender"/>
    </xsd:sequence>
  </xsd:complexType>

<xsd:element name="Farm">
  <xsd:complexType content="elementOnly">
    <xsd:sequence>
      <xsd:element name="Animal" type="AnimalType"
        minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="xmlns:xsi"
      type="xsd:uriReference" use="default"
      value="http://www.w3.org/1999/XMLSchema-
      instance"/>
    <xsd:attribute name="xsi:noNamespaceSchemaLocation"
                  type="xsd:string"/>
    <xsd:attribute name="xsi:schemaLocation"
                  type="xsd:string"/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```

DTD, Document Type Definition

Alternativ zu den XML-Schemen gibt es noch weitere Validierungsmechanismen. Ebenfalls weit verbreitet sind DTDs (*Document Type Definition*), die vor allem zur Beschreibung von Datenbanktabellen herangezogen werden.

Listing 11.4 DTD zur Animal-Farm

```
<?xml version="1.0" encoding="UTF-8"?>
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="Name" content="textOnly"
  dt:type="string"/>
  <ElementType name="Family" content="textOnly"
  dt:type="string"/>
  <ElementType name="Weight" content="textOnly"
```

```
  dt:type="int"/>
  <ElementType name="Gender" content="textOnly"
  dt:type="string"/>

  <ElementType name="Animal" content="mixed">
    <element type="Name" minOccurs="1" maxOccurs="*"/>
    <element type="Family" minOccurs="1"
    maxOccurs="*"/>
    <element type="Weight" minOccurs="1"
    maxOccurs="*"/>
    <element type="Gender" minOccurs="1"
    maxOccurs="*"/>
  </ElementType>

  <ElementType name="Farm" content="mixed">
    <element type="Animal" minOccurs="1"
    maxOccurs="*"/>
  </ElementType>
</Schema>
```

Es mag hilfreich sein, dass ein Schema weitgehend analog zu einer Typdeklaration einer modernen Programmiersprache ist, wie Sie es in Tabelle 1.2 sehen können. Allerdings kann man mit Schemen noch deutlich komplexere Definitionen bewerkstelligen.

ABAP IV	Visual Basic	XML-Schema

Types:
TyFamily
Type
C(12)
	Public Enum TyFamily
	  Cow = 1
	  Pig = 2
	End Enum
		<xsd:element name="TyName">
		  <xsd:simpleType
		  base="xsd:string">
		    <xsd:enumeration
		    value="Cow"/>
		    <xsd:enumeration
		    value="Pig"/>
		  </xsd:simpleType>
		</xsd:element>

Tabelle 11.2 Vergleich der Datenstrukturen in ABAP, VB und XML

ABAP IV	Visual Basic	XML-Schema
Types: TyGender Type C(1)	Public Enum TyGender male = 1 female = 2 End Enum	<xsd:element name="TyGender"> <xsd:simpleType base="xsd:string"> <xsd:enumeration value="F"/> <xsd:enumeration value="M"/> </xsd:simpleType> </xsd:element>
Types: Begin Of Animal Name(32), Family Type TyFamily, Weight Type P, Gender Type TyGender, End Of Animal.	Public Type Animal Name As String Family As TyFamily Weight As Long Gender As TyGender End Type	<ElementType name="Animal" content="mixed"> <element type="Name"/> <element type="Family"/> <element type="Weight"/> <element type="Gender"/> </ElementType>

Tabelle 11.2 Vergleich der Datenstrukturen in ABAP, VB und XML (Forts.)

Anzumerken bleibt noch, dass ein Schema sowohl in dem XML-Dokument mitgeführt als auch in einer separaten Datei abgespeichert werden kann. Ansonsten ist das Erstellen eines XML-Schemas nicht sehr schwierig, lediglich die produzierten Ergebnisse sind manchmal etwas schwer zu lesen.

11.3 DOM – Document Object Model

Reale Datenstrukturen sind meistens hierarchisch

Zu den Kernkompetenzen der Webentwicklung gehört das Arbeiten mit Dokumenten. Die Grundidee ist es, dass sich Datenströme in aller Regel in eine hierarchische Baumstruktur abbilden lassen. Auch wenn die Daten dem Inhalt nach unterschiedlich sind, bleibt die Struktur von verwandten Dokumenten meistens gleich. Eine solche Struktur eines Dokuments nennt man ein *Document Object Model* (DOM). In einer gewissen Weise ist dies auch eine Renaissance des hierarchischen Datenbankmodells.

11.3.1 DOM-Beispiele

DOM ist ein einfacher Datenbaum

Im Grunde genommen sind DOMs selbst erklärend. Es sind einfache Bäume mit wenigen Anmerkungen, die besagen, wie oft ein bestimmter Ast vorkommen kann oder muss. Es folgen einige Beispiele für Dokumente und möglicher DOMs.

Listing 11.5 Microsoft-Word-DOM

```
Document
    +- [1…*] Paragraphs
          +- [1…*] Words
                +- [1…*] Characters
    +-             +- [1…*] Fields (Pictures, Formulae
                                    etc.)
```

Auch einfache Textdateien haben ein DOM, das natürlich auch entsprechend einfach ist.

Listing 11.6 DOM einer einfachen Textdatei

```
Text Document
      +- [1…*] Characters
```

Ein DOM ist notwendigerweise eindeutig. Je nach Bedarf kann man auch Komplexität hinzufügen oder weglassen, wie am Beispiel einer Textdatei zu erkennen ist.

Listing 11.7 Alternatives DOM einer Textdatei

```
Text Document
             +- [1…*] Paragraphs
                                +- [1…*] Characters
```

Auch IDocs haben ein DOM.

Listing 11.8 SAP-IDoc-DOM

```
IDoc Document
       +- [1…1] Control record
                   +- [1…*] Segments
                                 +- [1…1] Segment info
                                 +- [1…1] Data
                                        +- [1…*]
                                       Segment fields
```

Listing 11.9 Beispiel eines SAP-IDoc

```
EDI_DC40  043000000000001234540B 3012  MATMAS03
  MATMAS   DEVCLNT444  PROCLNT100
E2MARAM001                444Somedata
E2MAKTM001
```

Ebenso SOAP-Dokumente:

Listing 11.10 SOAP-DOM

```
SOAP Document
      +- [1…1] SOAP Envelope
                    +- [0…1] SOAP Header
                    ¦     +- [0…*]Header Block
                    ¦
                    ¦
                    +- [1…1] SOAP Body
                                 +- [1…*]Message Body
                                 +- [1…*]Elements
```

Listing 11.11 Beispiel eines SOAP-Dokuments

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/
  soap/envelope/">
  <soap:Body>
…
  </soap:Body>
</soap:Envelope>
```

11.4 XML-Parser

XML-Parser sind fertige Bibliotheken, die ein XML-Dokument lesen und es in eine leichter zu verarbeitende Baum- oder Matrixstruktur umwandeln.

Performance

Es ist bei weitem der mühsamste und auch langweiligste Teil, ein XML-Dokument von seinen Tags zu befreien und die Daten in geeignete interne Datenstrukturen umzuwandeln. Diese Arbeit nehmen Ihnen vorgefertigte XML-Parser ab, von denen es mittlerweile jede Menge gibt. Die Parser unterscheiden sich oft wesentlich in der Performance, denn man kann langsam oder auch schnell durch einen rekursiv strukturierten Baum wandern.

Programmcode aus Schema generieren

Viele Parser bieten auch noch zusätzlich die Möglichkeit, aus einem Schema oder einem WSDL-Dokument den Programmcode zu einer Programmiersprache zu generieren. Die meisten Tools existieren für die Java-Welt, während Microsoft alles in seinem SOAP-Toolkit zusammenfasst.

11.4.1 DOM oder SAX

Es gibt grundsätzlich zwei verschiedene Vorgehensweisen beim Parsen von XML-Dokumenten. Die einen bilden das Document Object Model des XML-Dokuments ab und heißen deshalb auch DOM-Parser, während die anderen das XML-Dokument Stück für Stück abarbeiten. Sie werden SAX-Parser genannt. Der Unterschied liegt demnach in der Technik, wie das Dokument verarbeitet wird.

DOM-Parser

DOM-Parser bilden das Dokument als Baum ab

Das Dokument wird vom Parser komplett analysiert und dann in ein hierarchisches Objekt umgewandelt. Dies ist vergleichbar einem Compiler bei den Programmiersprachen, die das Quellprogramm zunächst übersetzen, bevor sie es verarbeiten.

SAX-Parser

SAX-Parser arbeiten Dokumente Stück für Stück ab

Dieser Parser ist gedacht für sehr große XML-Dokumente. Er parst jeweils nur das nächste Token des Dokuments und übergibt dann die Verarbeitung an die rufende Applikation. Die Realisierung ist meistens eventgesteuert, das heißt, der Parser löst ein bestimmtes Event aus, wenn er eine logische Einheit des Dokuments erkannt hat, und überlässt dem Eventhandler die Verarbeitung. Nach Abarbeiten des Events führt der Parser seine Arbeit bis zum nächsten Ereignis fort. Diese Vorgehensweise ist mit einem Interpreter zu vergleichen, der Statement für Statement liest und jeweils sofort ausführt.

11.4.2 Der Microsoft XML-Parser

Microsoft packt seinen Parser geschickterweise gleich in den Internet Explorer und schlägt damit zwei Fliegen mit einer Klappe. Zum einen wird der Internet Explorer voll XML-kompatibel und darüber hinaus hat jeder Windows-Anwender, und damit jeder, der einen Computer auf seinem Schreibtisch hat, auch die XML-Libraries von Microsoft. Da Standards nicht verordnet werden, sondern entstehen, schafft Microsoft damit die Basis, seine Vorstellungen als Standard durchzusetzen, denn was jeder hat, hat die größten Chancen, zum Standard zu werden.

Dennoch ist die Microsoft XML-Klasse (*msxml4.dll*) eine sehr mächtige Bibliothek, die fast alle Funktionalitäten enthält, die man für die Entwicklung von Webservices benötigt. Die MSXML-Klasse verfügt sowohl über einen DOM-Parser als auch über einen SAX-Parser.

XML Document ActiveX – Microsoft.XMLDOM

11.4.3 XML-Parser Libraries for Java

Es gibt eine ganze Reihe konkurrierender XML-Parser für Java, die meisten davon entstammen der Open Software Foundation wie zum Beispiel *Apache XERXES*.

jDOM

Als Beispiel für einen Java-Parser wurde hier *jDOM* ausgewählt. jDOM ist ein Projekt einer Gruppe von Open-Source-Entwicklern. Es ist ein für Entwickler kostenlos vertriebener voll funktionsfähiger DOM-Parser und DOM-Treemanager, der von Konzept und Anwendung her sehr dem MSXML von Microsoft ähnelt.

Listing 11.12 Darstellung der Animal-Farm mit jDOM für Java

```
package jdomtest;
import org.jdom.*;
import org.jdom.output.*;

public class simpledom {
  Document myDoc;
  Element root;

  public simpledom() {   } /* This is the constructor !
*/

  private void makedoc() {
    root = new Element("Shop");
    myDoc = new Document(root);
  }
/* Here we add a single Animal element to our document
*/
  public void AddAnimal(String name, String family, int
weight, String gender)  {
      Element animal = new Element("Animal");
      Attribute attName = new Attribute("Name", name);
```

```
        /* Set the name as attribute "<Animal
           Name="Elsa">" */
        animal.setAttribute(attName);
        root.addContent(animal);
        /* Add the family element "<Family>Cow</Family>"
        */
        Element elem = new Element("Family");
        elem.addContent(family);
        animal.addContent(elem);
        /* Add the Weight element and convert int to
           String" */
        elem = new Element("Weight");
        elem.addContent(Integer.toString(weight));
        animal.addContent(elem);
        /* Add the Gender element "<Gender>F</Gender>" */
        elem = new Element("Gender");
        elem.addContent(gender);
        animal.addContent(elem);  }

    public String XML() {
      String XMLstring;
        XMLOutputter xmlout = new XMLOutputter(" ",true);
        XMLstring = xmlout.outputString(myDoc);
        return XMLstring;
    }
  /* I think toSTring is the logical name for this */
    public String toString() {
      return this.XML();
      }
    public static void main(String[] args) {
      simpledom TestSimpledom = new simpledom();
      System.out.println("Hello jDOM");
      TestSimpledom.makedoc();
      TestSimpledom.AddAnimal("Elsa", "Cow", 420, "F");
      System.out.println("*** Result from
      Document.toString()");
      System.out.println(TestSimpledom.myDoc.toString());
      System.out.println("*** Result from this.XML()");
      System.out.println(TestSimpledom.XML());
      System.out.println("*** Result from
```

```
    this.toString()");
    System.out.println(TestSimpledom.toString() );
  }
```

11.4.4 XML-Browser und Editoren

XMLSPY

XML-Dokumente können Sie jederzeit mit dem Internet Explorer ab Version 5 oder mit Netscape ab Version 6 ansehen. Allerdings sind die Formatierungsmöglichkeiten der Browser doch sehr begrenzt. Als Notlösung mögen die Browser ihren Sinn haben, aber für komplexe Dokumente oder gar zum Edieren von XML-Dokumenten eignen sie sich nicht.

Einer der leistungsstärksten XML-Editoren

Es gibt eine Vielzahl von XML-Editoren auf dem Markt, viele davon sind gratis in der Public Domain zu finden und auch günstige Shareware findet man zum Beispiel bei *download.com*. Ein solcher Editor, der auch als Shareware begonnen hat, ragt aber aus der Masse heraus, es ist XMLSPY der österreichischen Firma *Altova* (*www.xmlspy.com*). XMLSPY bietet vom Viewer bis zum Schema-Generator oder XSLT-Debugger fast alles, was man für das Arbeiten mit XML benötigt, und ist somit eine überragende XML-IDE.

11.4.5 Namespaces – Namensräume

Jeder XML-Webservice benötigt einen eindeutigen Namensraum, englisch *namespace*, damit eine beliebige Client-Anwendung die verschiedenen Webservices auseinander halten kann. Jeder Webservice sollte seinen eigenen Namespace festlegen und den Verweis auf den Namespace als eindeutige URN im Internet veröffentlichen.

Für Entwicklungszwecke können Sie auch auf vordefinierte Namespaces des W3C zurückgreifen oder auf Microsofts *http://tempuri.org*, der von Microsofts SOAP-Development-Toolkit als Default-Namespace verwendet wird.

Namespace muss eindeutig sein

Der Namespace sollte so eindeutig sein, dass damit auch Ihr Webservice eindeutig von anderen unterschieden werden kann, mit anderen Worten: Der Namespace soll identifizierend für einen Webservice sein. Das erreicht man am einfachsten durch Zufügen des eigenen Domain-Namens.

Namespace muss eine öffentliche URN sein

Der Verweis auf den Namespace ist eine URN, muss aber nicht immer auf eine URL im Web verweisen. Wichtig ist lediglich, dass der Nutzer des Webservices anhand der URN den Namespace eindeutig bestimmen kann. Häufig genügt es dem Parser, die URN zu erkennen, er hat aber

ansonsten den Namespace bereits hart codiert eingebaut. Die folgenden Beispiele stammen aus Microsoft.NET und setzen die Namespace-Referenz in den verschiedenen Programmiersprachen jeweils auf »http://microsoft.com/webservices/«:

Listing 11.13 Beispiele für Namensraum-Referenzen Microsoft.NET

C#

```
[WebService(Namespace="http://microsoft.com/webservices/
")]
public class MyWebService {
    // implementation
}
```

Visual Basic.NET

```
<WebService(Namespace:="http://microsoft.com/webser-
vices/")> Public Class MyWebService
    ' implementation
End Class
```

Visual J#.NET

```
/**@attribute WebService(Namespace="http://micro-
soft.com/webservices/")*/
public class MyWebService {
    // implementation
}
```

11.5 XSLT – eXtended Style Sheet Language Transformations

Auf XML basierende Skriptsprache

Die eXtended Stylesheet Language ist eine auf XML basierende Skriptsprache, die verwendet wird, um XML-Dokumente in ein anderes Dokument zu transformieren. Die durch XSLT definierten Dokumente nennt man *XSL-Stylesheets*.

Formatierung im Browser gedacht

XSL wurde ursprünglich entworfen, um Formatierungen auf die Darstellung von XML in einem Browser zu ermöglichen, ähnlich den Cascading Stylesheets. XSL ist eine auf Templates (Mustervorlagen) basierende Programmiersprache, die XML-Dokumente durchläuft und jedes Element in einen beliebigen anderen String umsetzen kann. Dies bedeutet auch, dass das Ergebnis einer Transformation kein XML-Dokument sein muss. XML kann also auch zum Beispiel in HTML oder PDF transformiert werden.

11.6 HTML-Seiten mit XSL-Stylesheets und XML

Es gibt Dutzende von Büchern über XML und XSL, vermutlich mittlerweile mehr als 100. Allen diesen Büchern ist gemeinsam, dass ihre Autoren ein sicher vorzügliches Wissen von XSL und XML haben und dementsprechend all die schönen und fantastischen Möglichkeiten ausbreiten.

Allerdings vergessen sie hinter den dargestellten Möglichkeiten, dass XML und XSL vor allem eines sein sollen: nämlich einfach! Ich werde mich deshalb auf den häufigsten Anwendungsfall von XSL beschränken, der Konversion von XML-Dateien in ein darstellbares HTML-Format.

11.6.1 Inventurtabelle der Animal-Farm

Gehen wir wieder von der Animal-Farm aus, die wir schon bei der Darstellung von XML gesehen haben. Unser Ziel wird es sein, die Daten des XML-Dokuments als HTML-Tabelle darzustellen.

Name	Family	Gender	Weight
Elsa	Cow	female	420
Rosa	Pig	male	120
Lisa	Chicken	male	3

Tabelle 11.3 Tabelle der Tiere der Animal-Farm

Das HTML zu dieser Tabelle sieht aus wie in dem nachstehenden Beispiel.

Listing 11.14 HTML zur Tabelle der Tiere

```
<html>
  <body>
    <table border="1">
      <tr><th>Name</th><th>Family</th>
      <th>Gender</th><th>Weight</th></tr>

      <tr><td>Elsa</td><td>Cow</td>
      <td>female</td><td>420</td></tr>
      <tr><td>Rosa</td><td>Pig</td>
      <td>male</td><td>120</td></tr>
     <tr><td>Lisa</td><td>Chicken</td>
     <td>male</td><td>3</td></tr>
    </table>
  </body>
</html>
```

Header und Body

Man kann erkennen, dass wir es mit zwei Blöcken zu tun haben, einem Envelope mit den Tags <HTML> und <BODY> sowie der eigentlichen Tabelle, die vom <TABLE>-Tag eingeschlossen ist. Innerhalb der Tabelle finden sich dann die Überschrift mit den <TD>-Tags und die Zeilen mit <TR>.

11.6.2 XSL – Step by step

Wir werden nun Schritt für Schritt ein XSL-Stylesheet aufbauen, das die Transformation des XML-Dokuments vornimmt. Jedes XSL-Stylesheet ist mit <xsl:stylesheet> eingerahmt, besitzt also ein Root-Element mit dem Namen xsl:stylesheet.

Listing 11.15 Ein minimales XSL-Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet>
</xsl:stylesheet>
```

XSLT hat einen Namensraum

Zunächst müssen wir dem Stylesheet einen Namensraum zuweisen. Wie bei jedem XML-Dokument definiert der Namespace bei XSL ebenfalls die Namen der Tags und Attribute, die für das Stylesheet zugelassen sind. Der Standard-Namespace für W3C-kompatible Stylesheets ist:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

Listing 11.16 XSL-Stylesheet mit Namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
</xsl:stylesheet>
```

Dummy-Namensraum

Die Standard-Referenz xmlns:xsl=http://www.w3.org/TR/WD-xsl ist lediglich eine Dummy-Referenz. Im Allgemeinen wird diese nicht jedes Mal aus dem Internet geholt, sondern der Parser hat die darin enthaltenen Definitionen fest eingebaut. Im Grunde gibt es viele solcher Dummy-Referenzen, die aus formalen Gründen mitgeführt werden, aber der Effizienz halber fest eingebaut sind. Erkennt der Parser eine URN aber nicht, wird er diese im Internet nachlesen, findet er die URN nicht, gibt es einen Fehler.

11.6.3 XSL-Templates

Template-Sprache

Templates sind das Herz des XSL-Formattings. Ein solches Template bestimmt eine Reihe von XSL-Definitionen in seinem Body, die dann auf jedes passende XML-Tag angewandt werden. Tatsächlich geht der Parser

das XML-Dokument von oben nach unten durch und ersetzt das passende Element des Templates mit dem Wert aus dem XML. Mit Hilfe eines `match`-Attributs kann man die richtigen Tags herausfiltern. Die Angabe von `match="/"` passt auf alle Elemente des XML-Dokuments.

Listing 11.17 XSL-Stylesheet, das alle XML-Tags wegfiltert

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
      <xsl:template match="/">
      Hiding the root
      </xsl:template>
</xsl:stylesheet>
```

Das vorstehende Beispiel produziert als Ergebnis:

```
Hiding the root
```

11.6.4 XSL auf die Animal-Farm angewandt

Wir sind nun bereit, das Template auf das Animal-Farm-XML-Dokument anzuwenden. In unserem Beispiel arbeiten wir mit einem einzigen Template, das auf die Root des zu transformierenden Dokuments angewandt wird (`match="/"`). Dieses Template erzeugt den Body eines HTML-Dokuments und loopt dann über die Elemente (`<xsl:for-each select="Farm/Animal">`) der Animal-Farm, um für jedes Tier eine HTML-Tabellenzeile (`<tr><td>Cow</td></tr>`) zu erzeugen.

Listing 11.18 Ein passendes XSL-Stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <html>
      <head/>
      <body>
        <table border="1">
          <tr>
            <th>Name</th>
            <th>Family</th>
            <th>Gender</th>
            <th>Weight</th>
          </tr>
```

```
        <xsl:for-each select="Farm/Animal">
        <tr>
          <td>
            <xsl:value-of select="@Name"/>
          </td>
          <td>
            <xsl:value-of select="Family"/>
          </td>
          <td>
            <xsl:choose>
              <xsl:when
              test=".[Gender='M']">male</xsl:when>
              <xsl:when
              test=".[Gender='F']">female</xsl:when>
              <xsl:otherwise>unknown</xsl:otherwise>
            </xsl:choose>
          </td>
          <td>
            <xsl:value-of select="Weight"/>
          </td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
</xsl:template>
</xsl:stylesheet>
```

11.6.5 Anatomie des XSL-Stylesheets

Schauen wir uns das Stylesheet noch einmal näher an, um zu verstehen, wie die Transformationen vor sich gehen:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Bezug auf den Standard-Namensraum:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

`match="/"` bedeutet das oberste Element des XML-Dokuments, das heißt, das Stylesheet ist auf alle Elemente des Eingabedokuments anzuwenden.

```
  <xsl:template match="/">
    <html>
      <head/>
      <body>
        <table border="1">
          <tr>
            <th>Name</th>
            <th>Family</th>
            <th>Gender</th>
            <th>Weight</th>
          </tr>
```

Bis an diese Stelle formen wir den statischen Block der HTML-Tabelle.

Ab jetzt gehen wir in einer Schleife über alle Tags <Animal>, die selbst Töchter des Tags <Farm> sind.

```
<xsl:for-each select="Farm/Animal">
    <tr>
      <td>
```

Die folgenden Zeilen geben den Wert des Attributs Name aus, das den <Animal>-Attributen zugewiesen wurde. Attribute werden immer durch Voranstellen des at-Zeichens (@) referenziert. Im Unterschied dazu wird einer Variablen in XSL ein Dollar-Zeichen ($) vorangestellt.

```
<xsl:value-of select="@Name"/>
      </td>
    <td>
```

Als nächste erfolgt die Ausgabe des Wertes des Elements mit dem Tag <Family> unterhalb von <Animal>.

```
<xsl:value-of select="Family"/>
    </td>
    <td>
```

Auch Fallunterscheidungen sind in XSL durch das Statement <xsl:choose> möglich, das den Wert des Elements <Gender> auf verschiedene Werte prüft und diesen durch einen entsprechenden Text ersetzt, also »M« durch »male« und »F« durch »female«. Der Punkt innerhalb des Vergleichs test=".[Gender='M']" referenziert den aktuell bearbeiteten Knoten.

```
          <xsl:choose>
            <xsl:when
            test=".[Gender='M']">male</xsl:when>
            <xsl:when
            test=".[Gender='F']">female</xsl:when>
            <xsl:otherwise>unknown</xsl:otherwise>
          </xsl:choose>
        </td>
        <td>
          <xsl:value-of select="Weight"/>
        </td>
      </tr>
    </xsl:for-each>
  </table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Das nachstehende Listing zeigt ein XML-Dokument, das sich mit obigem XSLT transformieren lässt. Ein Browser transformiert das XML automatisch, wenn der Verweis auf ein Stylesheet wie im nachstehenden Beispiel bereits im Dokument steht.

Listing 11.19 Zu transformierendes XML-Dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="U:\examples\XML\Farm.xsl"?>
<!-- Reference to an existing XSL stylesheet "Farm.xsl"
-->
<Farm>
    <Animal name="Elsa">
<!-- One <Animal> tag for every animal -->
        <Family>Cow</Family>
        <Weight>420</Weight>
        <Gender>F</Gender>
<!-- One tag for each characteristic of the animal -->
    </Animal>
    <Animal name="Rosa">
        <Family>Pig</Family>
        <Weight>120</Weight>
```

```
        <Gender>M</Gender>
    </Animal>
    <Animal name="Lisa">
        <Family>Chicken</Family>
        <Weight>3</Weight>
        <Gender>M</Gender>
    </Animal>
</Farm>
```

Das nachfolgende Diagramm zeigt noch einmal in übersichtlicher Form, wie das Stylesheet in die tabellarische Ausgabe übergeht.

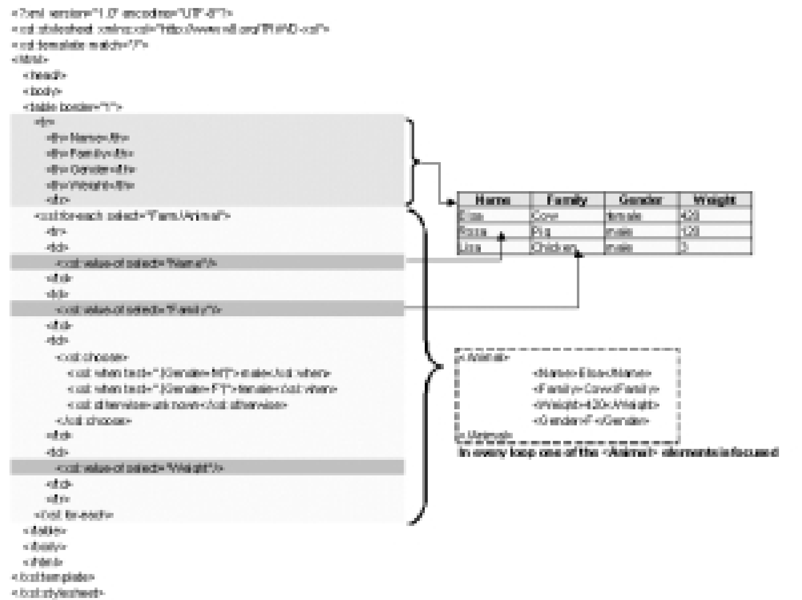


Abbildung 11.2 Synoptische Sicht auf Stylesheet und transformierte Ausgabe

11.6.6 Parsing des XML und Anwendung des XSL

Die Transformation des XML-Dokuments in HTML durch ein Stylesheet benötigt einen Parser, der dies leistet. Die meisten XML-Parser können auch Transformationen mittels XSL durchführen. In einer Microsoft-Umgebung können wir wieder auf MSXML und die darin implementierten Methoden `transformNodeToObject` und `transformNode` zurückgreifen.

Listing 11.20 VBA-Program zur Transformation von XML-Daten mit XSL-Stylesheet und msxml.dll

```
Dim xmlDoc As MSXML2.DOMDocument
Dim xslDoc As MSXML2.DOMDocument

Sub Main()
    Set xmlDoc = New MSXML2.DOMDocument
    Set xslDoc = New MSXML2.DOMDocument

    xmlDoc.Load ("U:\examples\XML\Farm.xml")
    xslDoc.Load ("U:\examples\XML\Farm.xsl")
    Debug.Print xmlDoc.transformNode(xslDoc)
End Sub
```

Listing 11.21 ASP zur Transformation von XML-Daten mit XSL-Stylesheet und msxml.dll

```
<%
Dim xmlDoc As MSXML2.DOMDocument
Dim xslDoc As MSXML2.DOMDocument
Set xmlDoc = CreateObject("Microsoft.XMLDOM")
Set xslDoc = CreateObject("Microsoft.XMLDOM")

xmlDoc.Load ("U:\examples\XML\Farm.xml")
xslDoc.Load ("U:\examples\XML\Farm.xsl")
Debug.Print xmlDoc.transformNodeToObject(xslDoc,
response)
%>
```

11.7 SOAP

Universelle Aufrufmethode für RPC

SOAP steht für *Simple Object Access Protocol*. Es ist ein einfaches Protokoll, das durch Übertragen von XML-Dokumenten Programme auf einem Remote-Computer aufrufen kann. Dabei werden der Name der zu rufenden Methode und die Parameter in einen XML-Tree codiert und die Anfrage an den Server geschickt. Der packt den SOAP-Request aus und ruft dann die gewünschte Methode in geeigneter Weise auf.

Listing 11.22 Aufruf einer Multiplikation via SOAP

```
POST /quickstart/aspplus/samples/services/
MathService/VB/MathService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
```

```
Content-Length: length
SOAPAction: "http://tempuri.org/Multiply"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XML-
Schema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Multiply xmlns="http://tempuri.org/">
      <A>float</A>
      <B>float</B>
    </Multiply>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XML-
Schema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <MultiplyResponse xmlns="http://tempuri.org/">
      <MultiplyResult>float</MultiplyResult>
    </MultiplyResponse>
  </soap:Body>
</soap:Envelope>
```

Listing 11.23 Darstellung eines Arrays in SOAP

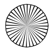
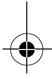
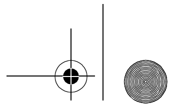
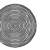
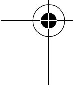
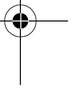
```
Array in Visual Basic
Dim someNumber[2] As Integer
someNumber[0] = "3"
someNumber[1] = "4"
Array in SOAP
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:int[2]"
name="someStrings">
<SOAP-ENC:int>3</SOAP-ENC:int>
<SOAP-ENC:int>4</SOAP-ENC:int>
```

```
</SOAP-ENC:Array>
Mehrdimensionales Array in Visual Basic
Dim someStrings[2,2] As String
someString[0,0] = "AAA"
someString[0,1] = "BBB"
someString[1,0] = "CCC"
someString[1,1] = "DDD"
Mehrdimensionales Array in SOAP
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[2,2]">
<SOAP-ENC:string>AAA</SOAP-ENC:string>
<SOAP-ENC:string>BBB</SOAP-ENC:string>
<SOAP-ENC:string>CCC</SOAP-ENC:string>
<SOAP-ENC:string>DDD</SOAP-ENC:string>
</SOAP-ENC:Array>
```

getUTCTime Das folgende Beispiel stammt aus den Trainingsseiten von *www.altova.com* und bedient sich eines Webservices, um die universelle Zeit zu bestimmen. Die Anfrage wird wieder als SOAP-Dokument formuliert, und in SOAP kommt auch die Antwort vom Server *nanonull.com*. Das Beispiel stellt die einfachste Form einer SOAP-Abfrage dar und zeigt deutlich die formale Dokumentstruktur. Die Abfrage hat keine Parameter, weshalb im Body des SOAP-Dokuments nur der Name des Webservices `<getUTCTime>` steht.

Listing 11.24 SOAP-Request zur Bestimmung der aktuellen UTC-Zeit

```
POST /TimeService/TimeService.asmx HTTP/1.1
Host: www.nanonull.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction:
"http://www.Nanonull.com/TimeService/getUTCTime"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getUTCTime
    xmlns="http://www.Nanonull.com/TimeService/" />
```


```
  </soap:Body>
</soap:Envelope>
```

Die Antwort ist wieder ein HTTP-Dokument, das ein XML-Dokument mit der gewünschten Antwort im Bauch trägt. Man beachte hier wieder, wie mit den Namen der XML-Tags gespielt wird. Der Service heißt `getUTCTime` und das Envelope der Antwort wird aus diesem Namen durch Anhängen von `Response` gebildet, ergo heißt das Envelope-Tag `getUTCTimeResponse`.

Listing 11.25 SOAP-Response von nanonull.com

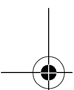
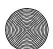
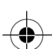
```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getUTCTimeResponse
    xmlns="http://www.Nanonull.com/TimeService/">
      <getUTCTimeResult>string</getUTCTimeResult>
    </getUTCTimeResponse>
  </soap:Body>
</soap:Envelope>
```

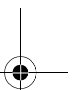
11.8 WSDL-Dokumente

WSDL (*Webservices Description Language*) ist eine Sprache, die mit Hilfe von XML-Dokumenten die Struktur der Requests und Responses eines Webservices beschreibt.

Beschreibt die Form eines RPC-Arufes

Ein WSDL-Dokument ist ein einfaches, fest strukturiertes XML-Dokument, das in formaler Weise beschreibt, wie ein Programm aufgerufen wird. Dahinter steckt die schlichte Idee, dass ein beliebiges rufendes Programm seine Anfrage als XML-Dokument verschickt, dieses von dem empfangenden Server entgegengenommen wird und aus dem gesendeten XML-Dokument automatisch einen korrekten Funktionsaufruf generiert.



Für SOAP beschreibt WSDL die Tags

Gehen wir davon aus, dass die Anfragen an den Server in SOAP übermittelt werden, beschreibt ein WSDL-Dokument, wie die Tags des SOAP-Dokuments auszusehen haben. Die Beschreibung ist so ausführlich, dass man mit ihrer Hilfe die Struktur des erforderlichen SOAP-Request-Dokuments bestimmen kann.

11.8.1 WSDL-DOM

DOM einer WSDL hat eine feste Struktur

Das DOM eines WSDL-Dokuments setzt sich aus einem Envelope mit dem festen Namen `<definitions>` und vier Teilbäumen zusammen, die die Elemente eines Webservices im Detail beschreiben. Die Teilelemente des WSDL sind in Tabelle 1.4 aufgeführt.

Element	Zweck
`<portType>`	Beschreibt die Aktion (das auszuführende Programm) für den Webservice.
`<message>`	Beinhaltet die Nachrichten, die vom Requester an den Server übermittelt werden. Das sind insbesondere die Parameter des gewünschten Funktionsaufrufs.
`<types>`	Definiert die Datenstrukturen (Data Types), die innerhalb des WSDL-Dokuments verwendet werden.
`<binding>`	Beschreibt das Protokoll, mit dem die Daten vom Requester an den Server und zurück übertragen werden.

Tabelle 11.4 Elemente eines WSDL-Dokuments

Damit ergibt sich das im folgenden Listing dargestellte, einfache Dokumentenmodell eines WSDL-Dokuments. Aus der Sicht des Protokolls ist der Dokumentenbaum immer dreistufig.

Listing 11.26 Strukturbaum eines WSDL-Dokuments (WSDL-DOM)

```
WSDL Document
       +- [1…1] <definitions>
       +- [0…*] <types>
       ¦      +- [0…*]type definitons
       ¦
       +- [0…*] <messages>
       ¦      +- [0…*]message definitons
       ¦
       +- [0…*] <portType>
       ¦      +- [0…*]port type definitons
```

```
        ¦
        ¦
        +- [0…*] <binding>
              +- [0…*]binding definitons
```

Der Rumpf eines WSDL-Dokuments sieht folgendermaßen aus:

Listing 11.27 Grundstruktur eines WSDL-Dokuments

```
<definitions>
<types>
   Typdefinitionenens........
</types>

<message>
   Nachrichtendefinition....
</message>

<portType>
   Portdefinition.......
</portType>

<binding>
   Binding-Definition....
</binding>

</definitions>
```

Ein WSDL-Dokument kann also so aussehen, wie im folgenden Listing:

Listing 11.28 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:AsinRe-
quest"/>
  </message>

  <message name="AsinSearchResponse">
    <part name="return" type="typens:ProductInfo"/>
  </message>

  <operation name="AsinSearchRequest">
    <input message="typens:AsinSearchRequest"/>
    <output message="typens:AsinSearchResponse"/>
  </operation>

  <binding name="AmazonSearchBinding"
  type="typens:AmazonSearchPort">
    <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/HTTP"/>
    <!-- Binding for Amazon Web APIs - RPC, SOAP over
    HTTP -->
    <operation name="AsinSearchRequest">
      <soap:operation
      soapAction="urn:PI/DevCentral/SoapService"/>
      <input>
        <soap:body use="encoded"
          encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:PI/DevCentral/SoapService"/>
      </input>
      <output>
        <soap:body use="encoded"
          encodingStyle=
          "http://schemas.xmlsoap.org/soap/encoding/"
             namespace=
             "urn:PI/DevCentral/SoapService"/>
      </output>
```

```
    </operation>
  </binding>

  <service name="AmazonSearchService">
    <!-- Endpoint for Amazon Web APIs -->
    <port name="AmazonSearchPort"
    binding="typens:AmazonSearchBinding">
      <soap:address location=
      "http://soap.Amazon.com/onca/soap2"/>
    </port>
  </service>

</definitions>
```

11.8.2 Beispiel eines WSDL-Dokuments für eine BAPI-Funktion

Weder das SAP R/3 Interface-Repository (IFR) noch der SAP Web AS bieten derzeit WSDL für BAPIs oder IDocs an. Lediglich der SAP Business Connector unterstützt WSDL unmittelbar. Das ist nicht weiter tragisch, da die WSDL sich aus dem im IFR gefundenen Schema einfach transformieren lässt.

Wir haben hier die RFC-Funktion RFC_CUSTOMER_UPDATE ausgewählt, mit der man die Daten eines Debitors in R/3 ändern kann. Das Beispiel (siehe Abbildung 11.19) zeigt in einem Ausschnitt aus dem ganzen WSDL-Dokument die Definition der Datentypen für die RFC-Parameter.

Kap11.fm Seite 236 Dienstag, 22. April 2003 2:33 14

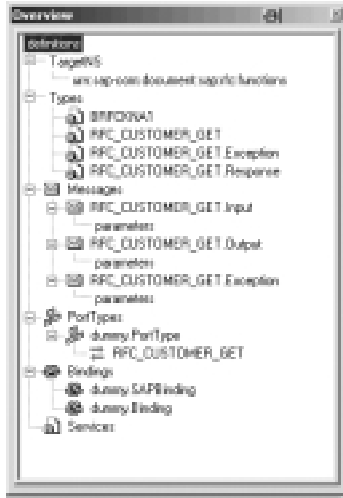


Abbildung 11.3 DOM-Ansicht des WSDL-Dokuments für RFC_CUSTOMER_UPDATE

XSD Schema Extract for RFC_CUSTOMER_UPDATE <types>

Abbildung 11.4 Beispiel eines WSDL-Dokuments für eine BAPI-Function

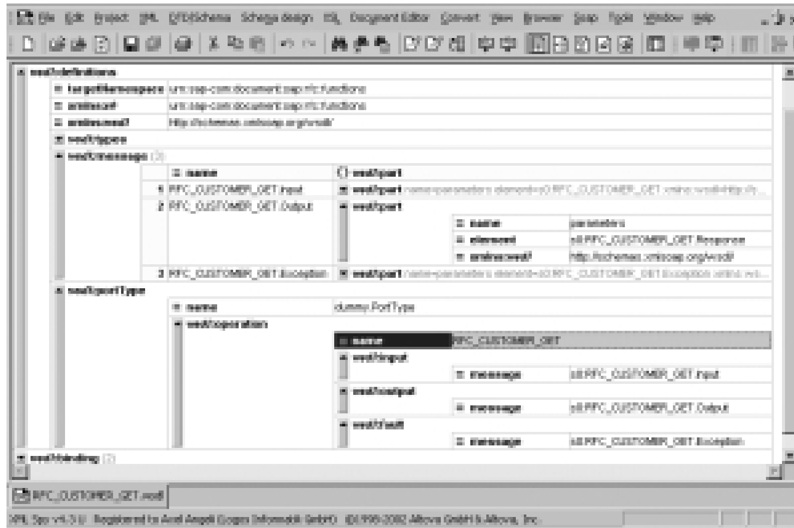


Abbildung 11.5 <messages>-Definitionen für RFC_CUSTOMER_UPDATE (XMLSPY™-Matrix-View)

Abbildung 11.6 <portTypes>-Definitionen für RFC_CUSTOMER_UPDATE (XMLSPY™-Matrix-View)