Proseminar
Web Site Building Techniques
XML

Birgit Mohr, 204210132
Jon Theegarten, 203210044

11.01.2006

Inhaltsverzeichnis

1. Einleitung

Was ist XML?

XML (Extensible Markup Language) kann am besten als ein Standard zur Speicherung von strukturierten, aber uninterpretierten, Daten beschrieben werden. Die Daten werden dafür über HTML-ähnliche Tags strukturiert, die für den Anwender frei definierbar sind. XML legt nur den Aufbau solcher Datendokumente fest; somit bleibt die Interpretation der Daten im konkreten Anwendungsfall dem Parser überlassen. So ist es möglich eine einzelne, strukturierte Datenbasis auf verschiedene Arten darzustellen. Ein Beispiel hierfür wären Wetterdaten, die einmal als Graphik, einmal als Tabelle und ein weiteres Mal als Text ausgegeben werden.

Historische Hintergründe

Entwickelt wurde XML als eine vereinfachte Teilmenge der weit komplexeren Metasprache SGML. Als kleine Übersicht über die Komplexität der Sprachen soll hier die Länge ihrer Definitionen dienen. Die Definition von SGML umfasst 500 Seiten, die von XML nur 26. Dennoch ist XML in der Lage gut 80% der Funktionalität von SGML zu übernehmen, was ein Grund dafür ist, warum heutzutage XML im Anwendungsbereich generell SGML vorgezogen wird.
Obwohl XML erst seit einiger Zeit große Bedeutung gewonnen hat ist sie keine besonders junge Technologie. Bereits gegen 1960 wurde bei IBM an einer Generalized Markup Language (GML) gearbeitet und geforscht. 1986 wurde eine Version davon dann als SGML von der International Standards Organization (ISO) übernommen. Durch seine Komplexität disqualifizierte sich SGML allerdings als einfache Markup Language, die z.B. im Web hätte Anwendung finden können. So entstand HTML, die nur eine spezielle Implementierung von SGML darstellt. Diese dagegen war aufgrund ihrer Einfachheit bald nicht mehr ausreichend und wird seit dem ständig erweitert. Das Problem dabei ist, dass HTML nicht standardisiert weiterentwickelt wurde, sondern von mehreren Seiten. Microsoft und Netscape fügten eigene Tags hinzu, die jeweils vom Browser des anderen nicht interpretiert werden konnten, während der Endnutzer keinen Einfluss auf die Sprache hatte.
Neue Technologien wie CSS oder dynamic HTML erweitern zwar die Möglichkeiten für Websites, zeigen aber auch das Problem dabei:
Was mit der Sprache möglich ist und wie sie zu interpretieren ist wird von den Browserherstellern entschieden.

Es gibt also die beiden Extreme SGML, welches voll erweiterbar, aber extrem komplex ist, und HTML, welches einfach, aber nicht erweiterbar ist. Um diese Lücke zu schließen wurde XML geschaffen, eine weniger komplexe Teilmenge von SGML, die dennoch den größten Teil von dessen Mächtigkeit übernimmt.

2. Fachtermini

In Verbindung mit dem Thema XML werden häufig Fachtermini verwendet, die zunächst näher erläutert werden müssen.

Wohlgeformtheit
Ein XML-Dokument ist wohlgeformt (well formed), falls es sämtliche Regeln für XML einhält. Beispielsweise muss für jedes öffnende Element (z.B. <item>) ein schließendes Element vorhanden sein (z.B. </item>).

Gültigkeit
Soll XML für den Datenaustausch verwendet werden, ist es von Vorteil, wenn das Format mittels einer Grammatik (z.B. einer Dokumenttypdefinition (DTD) oder eines XML-Schemas) definiert ist. Ein XML-Dokument, welches wohlgeformt ist und ein durch eine Grammatik beschriebenes Format einhält, heißt gültig (valid).

Parser
Im Allgemeinen wird ein Parser dazu verwendet, einen Text in eine neue Struktur zu übersetzen, z. B. in einen Syntaxbaum, welcher die Hierarchie zwischen den Elementen ausdrückt. HTML-Code ist beispielsweise für einen Computer zunächst einmal nichts anderes als ein Text, also eine Aneinanderreihung von Buchstaben und Zeichen. Der in einem Webbrowser enthaltene Parser analysiert das HTML und erstellt daraus eine Beschreibung der Webseite als Datenstruktur, welche die Grafik-Engine des Browsers anschließend graphisch auf den Bildschirm überträgt.

Zur Analyse des Texts verwenden Parser in der Regel einen separaten lexikalischen Scanner (auch Lexer genannt). Dieser zerlegt die (als simple Aneinanderreihung von Zeichen vorliegenden) Eingabedaten in Token (Eingabesymbole bzw. „Wörter“, die der Parser versteht); weil die Zerlegung in Tokens einer regulären Grammatik folgt, ist der Scanner meist ein endlicher Automat. Diese Token dienen als atomare Eingabezeichen des Parsers.

Der eigentliche Parser als Implementierung eines abstrakten Automaten (meist realisiert als Kellerautomat) kümmert sich dagegen um die Grammatik der Eingabe, führt eine syntaktische Überprüfung der Eingangsdaten durch und erstellt in der Regel aus den Daten einen Ableitungsbaum (in Anlehnung an das Englische gelegentlich auch als Parse-Baum bezeichnet). Dieser wird danach zur Weiterverarbeitung der Daten verwendet; typische Anwendungen sind die semantische Analyse, Codegenerierung in einem Compiler oder Ausführung durch einen Interpreter.

3. Regeln für XML-Dateien / Aufbau eines XML-Dokuments

Deklaration als XML-Datei

Jede XML-Datei sollte mit einer Auszeichnung beginnen, die den Bezug zu XML herstellt. Dazu dient die XML-Deklaration am Beginn der Datei.

```
<?xml version=”1.0” encoding=”utf 8” standalone=”yes”>
```

Version bestimmt die verwendete XML version, encoding den benutzten Zeichensatz. Standalone gibt an, ob diese XML-Datei eine externe DTD verwendet. Die Reihenfolge der Attribute ist als version – encoding – standalone festgelegt.
Die Deklaration ist allerdings optional, wenn keine verwendet wurde wird UTF-8 (internationaler Kodierung auf Basis der ISO/IEC-10646-Norm mit mindestens 8 Bit Zeichenbreite) als Datensatz verwendet.

Umschließende Tags

In XML muss jedes geöffnete Tag auch wieder geschlossen werden.

```
<text>Dies hier ist nur Text.</text>
```

Eine Ausnahme bilden leere Tags, die keine Daten umschließen.

```
<dieses-Tag-ist-leer />  =  <dieses-Tag-ist-leer></dieses-Tag-ist-leer>
```

Groß-/Kleinschreibung

Tags sind case-sensitive, d.h. XML unterscheidet im Gegensatz zu klassischem HTML strikt zwischen Groß- und Kleinschreibung.

```
<Text> Dies ist nur Text.</text>
```
Dies würde einen Fehler erzeugen!

```
<text> Dies ist nur Text.</text>
```
So ist es richtig!

“Nesting”

Tags müssen auf einer Ebene bleiben und dürfen z.B. weder ihr root-tag verlassen, noch in eines ihrer unter-tags hinein reichen.

Falsch:

```
<kategorie>
    <text1>Text1
    <text2>Text2</kategorie></text2>
</text1>
```

Richtig:

```
<kategorie>
    <text1>Text1</text1>
    <text2>Text2</text2>
</kategorie>
```

Attribute

Ein XML-Tag darf so viele Attribute haben, wie der Autor wünscht. Die Werte werden dabei in Anführungszeichen gesetzt.

```
<news author="Tagesschau.de" date="29.05.05" time="21:00">Neueste Nachrichten!</news>
```

Elementnamen

Elementnamen unterliegen einigen grundlegenden Regeln:

- Keine Leerzeichen oder Gleichheitszeichen
- Darf nicht mit "xml" oder einer Zahl beginnen
- Doppelpunkt nur begrenzt verwendbar (reserviert für Namensräume)

Reservierte Zeichen in XML

Wie bei HTML gibt es auch in XML reservierte Zeichen, die umschrieben werden müssen, da sie sonst vom Parser (falsch) interpretiert werden.

<	=	<
>	=	>
&	=	&
"	=	"
'	=	'

Beispiel:

```
<Element>dieses Element wird notiert als &lt;Element&gt;...&lt;/Element&gt;</Element>
```

Die Ausgabe sieht so aus:

dieses Element wird notiert als <Element>...</Element>

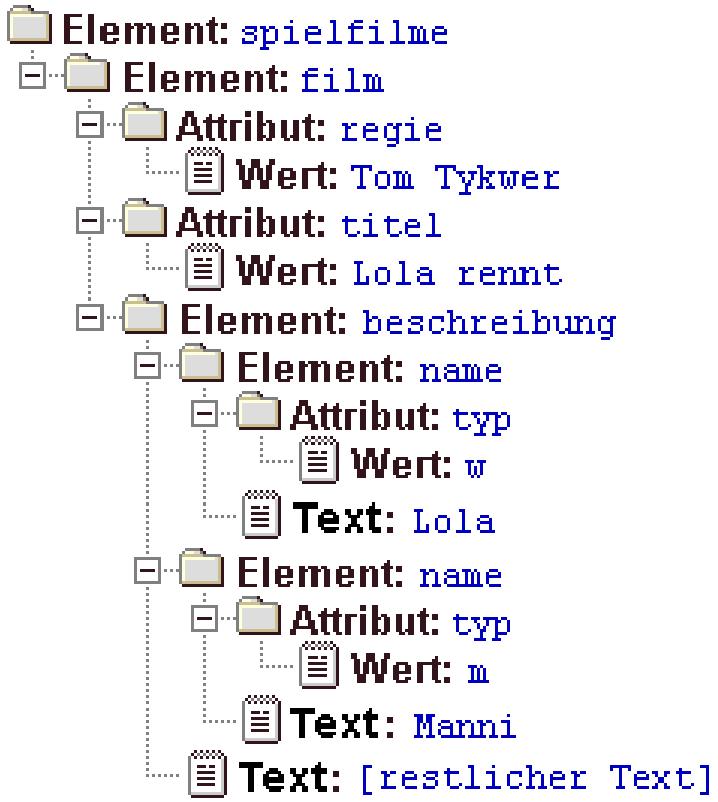
Baumstruktur

Ein XML-gerechtes Dokument besteht aus Elementen, Attributen, ihren Wertzuweisungen, und dem Inhalt der Elemente, der aus Text oder aus untergeordneten Elementen bestehen kann, die ihrerseits wieder Attribute mit Wertzuweisungen und Inhalt haben können. Es gibt Elemente mit und ohne Attribute, Elemente, innerhalb deren viele andere Elemente vorkommen können, und solche, innerhalb deren nur Text vorkommen kann, und sogar leere Elemente, die keinen Inhalt haben können. Die Struktur, die aus diesen Bestandteilen und ihren Grundregeln entsteht, lässt sich als Baumstruktur begreifen.

Beispiel-Code:

```
<spielfilme>
 <film regie="Tom Tykwer" titel="Lola rennt">
  <beschreibung>
   <name typ="w">Lola</name> rennt für <name typ="m">Manni</name>, der
    100000 Mark liegengelassen hat und noch 20 Minuten Zeit hat, das Geld
    auszuliefern.
  </beschreibung>
 </film>
</spielfilme>
```

Und die Baumstruktur dazu:



4. Kerntechnologien

Metasprachen

Um die Struktur von XML-Dokumenten zu beschreiben, bedient man sich so genannter Schemasprachen. Die zwei bekanntesten sind DTD und XML Schema.

DTD

Eine DTD (Dokumenttypdefinition) ist eine Beschreibung eines XML-Dokuments. Sie wurde zusammen mit XML standardisiert. Mit einer DTD kann allerdings nicht sehr strikt beschrieben werden, wie eine XML-Datei aussehen darf. Ein weiterer Nachteil ist die Tatsache, dass die DTD in einer eigenen Sprache abgefasst werden muss.

Die Dokumenttypdefinition ist eine Deklaration in SGML- und XML-Dokumenten, die die Struktur eines solchen Dokuments festlegt.

Eine DTD hat den Zweck, ein bestimmtes Auszeichnungsproblem zu lösen und bestimmt damit die Struktur: HTML bzw. XHTML für Webseiten, SVG für 2D-Vektorgrafiken, DocBook für technische Dokumentationen, TEI für geisteswissenschaftliche Texte und noch viele weitere.

In einer DTD werden Elemente, Attribute von Elementen und Entitäten definiert. In bestimmten SGML-Deklarationen werden Besonderheiten der Syntax-Verwendung vorgegeben, beispielsweise Abkürzungen oder lokale Umdefinitionen von Zeichen. Konkret heißt das, dass in einer DTD die Reihenfolge und die Verschachtelung der Elemente und die Art des Inhalts von Attributen festgelegt wird.

Die Verwendung einer DTD ist allerdings optional und findet vor allem Verwendung wenn man nicht nur wohlgeformten, sondern auch gültigen XML-Code erzeugen möchte. Für die Gültigkeit ist Vorraussetzung, dass die Datei die Bedingungen einer DTD erfüllt.

Regeln

Namen

Dürfen enthalten:
- Buchstaben (groß und klein)
- Ziffern 0 bis 9
- “_, -, ., :” Der “:” ist allerdings für Namesräume reserviert.

Das erste Zeichen eines Namens darf keine Ziffer sein. Üblich ist es mit einem Buchstaben, oder höchstens einem Unterstrich zu beginnen. Namen dürfen keine Leerzeichen enthalten und nicht mit “xml” beginnen.

Definition von Elementtypen

Elemente sind das, was in einer XML-Datei in einem Tag definiert wird. Z.B.

```
<datum>...</datum>
```

Elementtypen werden immer nach folgendem Schema in der DTD definiert:

```
<!ELEMENT Name (Inhalt)>
```

Der Name ist hierbei, innerhalb der Namensregeln von XML, frei wählbar.
Inhalt bestimmt den Typ des Inhaltes dieses Elementes und kann je nach Anforderung sehr komplex aussehen. Die einfachste Einstellung ist (#PCDATA), wodurch das Element beliebig viel Text, aber keine weiteren inneren Elemente enthalten kann.

Ein Element kann aber auch mehrere andere Elemente enthalten:

```
<!ELEMENT telefonnummer (vorwahlnummer, durchwahlnummer)>
<!ELEMENT vorwahlnummer (#PCDATA)>
<!ELEMENT durchwahlnummer (#PCDATA)>
```

Soll ein Unterelement mehrfach vorkommen können wird dies über “+” (eine oder mehrer Wiederholungen) oder “*” (beliebig viele Wiederholungen) dargestellt. Ein “?” macht ein Element optional (ein oder kein Vorkommen), zwei in einer runden Klammer durch eine “|” getrennte Elemente sind zueinander alternativ. Ein Beispiel:

```
<!ELEMENT adressen (adresse)*>

<!ELEMENT adresse (anrede?, name, (postfach | wohnanschrift), plzort)>

<!ELEMENT anrede (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT postfach (#PCDATA)>
<!ELEMENT wohnanschrift (#PCDATA)>
<!ELEMENT plzort (#PCDATA)>
```

Die Elementfolge muss aber nicht immer starr definiert sein.

```
<!ELEMENT text (#PCDATA | subjekt | prädikat | objekt)*>
<!ELEMENT subjekt (#PCDATA)
<!ELEMENT prädikat (#PCDATA)
<!ELEMENT objekt (#PCDATA)
```

Obiges Beispiel ließe eine beliebige Aneinanderreihung von Elementen der Typen “subjekt”, “prädikat” und “objekt” zu. Kein korrektes Deutsch, aber eine zulässige DTD. ;)

Außerdem gibt es noch die Schlüsselworte ANY (das einem Element beliebigen Inhalt erlaubt) und EMPTY (das keinen Inhalt erlaubt).
Ein <!ELEMENT anything ANY> könnte somit beliebig viele weiter Elemente, bzw. beliebig viel Text enthalten, ein <!ELEMENT nothing EMPTY> dagegen gar nichts.

Schlussendlich lassen sich Elemente noch zu einer Entity zusammenfassen z.B. um verschiedene Arten Text an mehr als einer Stelle zuzulassen, ohne jedes Mal alle aufführen zu müssen.
Ein abschließendes Beispiel:

```
<!ENTITY % text "hervorgehoben | hinweisend | auffordernd | zeilenumbruch">

<!ELEMENT buch
(titel,impressum,inhaltsverzeichnis,(kapitel+,stichwortverzeichnis)>

<!ELEMENT titel              (haupttitel,untertitel)>
<!ELEMENT haupttitel          (#PCDATA)>
<!ELEMENT untertitel          (#PCDATA)>

<!ELEMENT impressum           (cip,copyright,verlag)>
<!ELEMENT cip                 (#PCDATA | %text;)*>
<!ELEMENT copyright           (#PCDATA | %text;)*>
<!ELEMENT verlag              (#PCDATA | %text;)*>

<!ELEMENT inhaltsverzeichnis    (ivz_ueberschrift,(ivz_kapiteleintrag |
ivz_abschnittseintrag)*)>
<!ELEMENT ivz_ueberschrift      (#PCDATA)>
<!ELEMENT ivz_kapiteleintrag    (#PCDATA)>
<!ELEMENT ivz_abschnittseintrag (#PCDATA)>

<!ELEMENT kapitel             (kap_ueberschrift,(abschnitt)+)>
<!ELEMENT kap_ueberschrift      (#PCDATA)>

<!ELEMENT abschnitt           (ab_ueberschrift,(fliesstext | aufzaehlung |
grafik)+)>
<!ELEMENT ab_ueberschrift       (#PCDATA)>

<!ELEMENT fliesstext          (#PCDATA | %text;)*>
<!ELEMENT aufzaehlung           (aufzaehlungspunkt)+>
<!ELEMENT aufzaehlungspunkt     (#PCDATA | %text;)*>
<!ELEMENT grafik              (grafikdatei)>
<!ELEMENT grafikdatei         (#PCDATA)>

<!ELEMENT stichwortverzeichnis  (svz_ueberschrift,(svz_eintrag)*)>
<!ELEMENT svz_ueberschrift      (#PCDATA)>
<!ELEMENT svz_eintrag         (#PCDATA)>

<!ELEMENT hervorgehoben       (#PCDATA)>
<!ELEMENT hinweisend          (#PCDATA)>
<!ELEMENT auffordernd         (#PCDATA)>
<!ELEMENT zeilenumbruch       EMPTY>
```

Mit einer solchen DTD ließen sich Bücher in XML recht detailliert speichern.

Attribute

Elementen lassen sich auch Attribute zuweisen.

```
<!ELEMENT Elementname (Inhalt)>
<!ATTLIST Elementname
      Attributname_1 Inhalt [#REQUIRED|#IMPLIED|#FIXED "Wert"|Defaultwert]
      Attributname_n Inhalt [#REQUIRED|#IMPLIED|#FIXED "Wert"|Defaultwert]
>
```

Kommentare

Kommentare in DTD sind identisch mit denen in HTML definiert:

```
<!-- dies ist ein Kommentar. -->
```

Bedingte Abschnitte
Es ist möglich in der DTD einzelne Definitionen ausdrücklich aus- oder einzuschließen. Dies geschieht über die Schlüsselworte IGNORE und INCLUDE:

```
<![INCLUDE[<!ELEMENT IchWerdeEingeschlossen (#PCDATA)>]]>
<![IGNORE[<!ELEMENT IchWerdeVomParserIgnoriert (#PCDATA)>]]>
```

XML-Schema (XSD)

XSD ist die modernere Art XML-Dateien zu definieren und wird vom W3C empfohlen. Im Gegensatz zu DTDs werden XML-Strukturen in XSD in einem XML-Dokument beschrieben. Außerdem werden statt wie bei DTD nur (#PCDATA), das beliebige Daten enthält, in XSD unterschiedliche Datentypen unterstützt. Diese werden in einfache und komplexe Typen unterschieden.

Vorteile von XSD gegenüber DTDs:
- Sind erweiterbar
- Selbst in XML geschrieben
- Unterstützung für viele Datentypen
- Reguläre Ausdrücke
- Unterstützen Namensräume

Wozu werden XML-Schemata benötigt? Ein XML-Schema kann folgendes leisten:

- Definition der in einer XML-Datei vorkommenden Elemente
- Definition und Festlegung von Reihenfolge und Anzahl der Unterelemente
- Definition der erlaubten Attribute
- Definition von eigenen Datentypen
- Definition von Standard- und festen Werten

Regeln

Einbinden eines XML-Schemas

Ob und welche XSD eine XML-Datei verwendet wird zu Beginn über ein Attribut des Wurzel-Elementes definiert:

```
<?xml version=”1.0”>

<wurzel xmlns=”http://www.domain.de” xmlns:xsi=”http://www.domain.de/XML-Schema-Instanz” xsi:schemaLocation=”http://www.domain.de wurzel.xsd”>
```

XSD kann im Gegensatz zu DTD nicht innerhalb der XML-Datei definiert sein und benötigt immer eine eigene Datei.

Einfache Typen

Sind zunächst die in Programmiersprachen üblichen:

- xsd:string
- xsd:decimal
- xsd:integer
- xsd:boolean
- xsd:date
- xsd:time

In XML Schemas gibt es viele unterschiedliche Datentypen, so dass die Prüfung des Dokuments wesentlich präziser erfolgen kann als bei den DTDs. Eine Liste aller wichtigen Datentypen finden Sie in der folgenden Tabelle.

Typ	Beschreibung	Beispiele (durch Komma getrennt)
Zeichenkettentypen		
string	Zeichenkette	Helmut Meier
language	ISO639-1-Kürzel einer Sprache	de, eo, tr, ja
normalizedString	Zeichenkette, bei der LF, CR und TAB durch ein Leerzeichen ersetzt werden	Abstand ist egal
token	Zeichenkette, bei der jeder Whitespace durch ein Leerzeichen ersetzt wird und führende und folgende Leerzeichen entfernt werden	Abstand ist egal
ID	(nur bei Attributen) eindeutige ID, darf nicht rein numerisch sein	A123445
IDREF	(nur bei Attributen) Referenz auf eine ID	A123445
IDREFS	(nur bei Attributen) Referenzen auf IDs	A123445 A12443 BAGHOE
NMTOKEN	(nur bei Attributen) Zeichenkette, die ein Token darstellt	hallo
NMTOKENS	(nur bei Attributen) Zeichenkette, die ein oder mehrere Token darstellt	hallo eins
ganzzahlige numerische Typen		
integer	Ganzzahl	-12345, -1, 0, 126789
positiveInteger	positive Ganzzahl	1, 234234
negativeInteger	negative Ganzzahl	-1 -234234
nonNegativeInteger	positive Ganzzahl oder 0	0, 23433
nonPositiveInteger	negative Ganzzahl oder 0	-3442, 0
byte	8-bit-Integer mit Vorzeichen	-1, 125
unsignedByte	8-bit-Integer ohne Vorzeichen	0, 250
short	16-bit-Integer mit Vorzeichen	-1, 12555
unsignedShort	16-bit-Integer ohne Vorzeichen	1, 12833
int	32-bit-Integer mit Vorzeichen	-12345, -1, 0, 126789
unsignedInt	32-bit-Integer ohne Vorzeichen	4545352
long	64-bit-Integer mit Vorzeichen	-1, 12378967543233
unsignedLong	64-bit-Integer ohne Vorzeichen	0, 12378967543233

Datum-/Zeit-Typen		
time	Uhrzeit, ggf. mit Zeitzone	11:20:00.000, 12:20:00.123-1:00, 14:00Z
date	Datum (immer ISO8601-Format!)	2004-02-15
gMonthDay	Monat und Tag	12-25
gYear	Jahr (bringt aber nichts gegenüber int)	2000
gYearMonth	Jahr und Monat	2006-12
dateTime	Zeitstempel	2004-02-15T09:00:00, 2005-01-01T03:00:00Z
Duration	Zeitdauer, muss immer mit ”P“ beginnen	P2Y5M3DT10H30M12S, PT3.24S
sonstige Typen		
boolean	logischer Wert	true, false, 1, 0
hexBinary	binärer hex-String	a6b346e2441e
base64Binary	binärer base64-String	aGFsbG8K
anyURI	URI; falls Leerzeichen drin sind, mit %20 codieren	http://www.bg.bib.de/portale

Es gibt aber auch XML-spezifische Datentypen:

- QName: Qualified Name, Bezeichnung innerhalb eines Namensraumes
- anyURI: Universal Resource Identifier
- language: Sprachbezeichnung (de-DE, fr, en-UK)
- ID: Identifikationsattribut in XML-Elementen
- IDREF: Referenz auf ein ID-Attribut

Außerdem zu den einfachen Typen zählen Listen (die mehrere Elemente eines Typs enthalten können) und Unions (die als Vereinigung mehrerer Typen definiert sind). Einfache Typen können weder weitere Unterelemente, noch XML-Attribute besitzen.

```
<xsd:simpleType name="monatInt">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="12"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="monate">
  <xsd:list itemType="monatInt"/>
</xsd:simpleType>
```

Komplexe Elemente

Es gibt vier Arten von komplexen Elementen:

- Leere Elemente
- Elemente, die nur Text enthalten dürfen
- Elemente, die nur andere Elemente enthalten dürfen
- Elemente, die Text und weitere Elemente gemischt enthalten dürfen.

Art	Beschreibung / Beispiele
Aufzählung – erschöpfende Aufzählung aller erlaubten Werte; nebenstehend zwei Varianten. Bei der zweiten Variante wird ein Typ definiert, der auch von anderen Elementen benutzt werden kann; dies gilt genauso auch für alle weiteren Restriktionen.	<xsd:element name="automobil"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:enumeration value="Audi"/> <xsd:enumeration value="VW"/> <xsd:enumeration value="BMW"/> <xsd:enumeration value="Opel"/> <xsd:enumeration value="Mercedes"/> <xsd:enumeration value="Porsche"/> </xsd:restriction> </xsd:simpleType> </xsd:element> <xsd:element name="automobil" type="autoTyp" /> <xsd:simpleType type="autoTyp"> <xsd:restriction base="xsd:string"> <xsd:enumeration value="Audi"/> <xsd:enumeration value="VW"/> <xsd:enumeration value="BMW"/> <xsd:enumeration value="Opel"/> <xsd:enumeration value="Mercedes"/> <xsd:enumeration value="Porsche"/> </xsd:restriction> </xsd:simpleType>
Länge – Angabe von exakter, minimaler und/oder maximaler Länge	<xsd:element name="kennwort"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:length value="8"/> <!-- exakt 8 Zeichen --> </xsd:restriction> </xsd:simpleType> </xsd:element> <xsd:element name="passwort"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:minLength value="5"/> <xsd:maxLength value="10"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
Muster – Angabe eines regulären Ausdrucks, dem der Inhalt genügen muss	<xsd:element name="artikelcode"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:pattern value="[A-Z]{2}-[0-9]{3,5}"/> </xsd:restriction> </xsd:simpleType> </xsd:element>

	<xsd:element name="ISBN"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:pattern value="[-0-9]{13}|unbekannt|ohne"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
Wertbeschränkung – Angabe von Minimal- und/oder Maximalwerten	<xsd:element name="laenge"> <xsd:simpleType> <xsd:restriction base="xsd:double"> <xsd:minInclusive value="10.0"/> <xsd:maxExclusive value="20.0"/> </xsd:restriction> </xsd:simpleType> </xsd:element> <xsd:element name="ISBN"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:pattern value="[-0-9]{13}"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
Whitespace – Angabe, wie der XML-Prozessor mit Whitespace umgehen soll. Ohne Angabe (oder mit preserve) bleibt Whitespace erhalten; replace ersetzt alle Whitespaces durch ein Leerzeichen, während collapse zusätzlich noch führende und folgende Leerzeichen entfernt.	<xsd:element name="adresse"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:whiteSpace value="replace"/> </xsd:restriction> </xsd:simpleType> </xsd:element> <xsd:element name="beschreibung"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:whiteSpace value="collapse"/> </xsd:restriction> </xsd:simpleType> </xsd:element>
Format – maximale Anzahl Stellen festlegen; fraction-Digits bestimmt die maximale Anzahl Nachkommastellen, total-Digits die Gesamtanzahl der Stellen	<xsd:element name="preis"> <xsd:simpleType> <xsd:restriction base="xsd:decimal"> <xsd:fractionDigits value="2"/> <xsd:totalDigits value="10"/> </xsd:restriction> </xsd:simpleType> </xsd:element> <xsd:element name="beschreibung"> <xsd:simpleType> <xsd:restriction base="xsd:string"> <xsd:whiteSpace value="collapse"/> </xsd:restriction> </xsd:simpleType> </xsd:element>

Sequenzen

Bei einer Sequenz kommen die beschriebenen Elemente nur in genau der definierten Reihenfolge vor.
So sieht die Definition eines komplexen Elements aus, das andere Elemente in festgelegter Reihenfolge enthalten darf bzw. muss:

```
<xsd:element name="mitarbeiter">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="vorname" type="xsd:string"/>
      <xsd:element name="familienname" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Auswahl

Eine Auswahl ist eine exklusive Oder-Verknüpfung, d.h. es darf nur eines, der definierten Kindelemente vorkommen.

```
<xsd:element name=”motor” type=”motorTyp” />
<xsd:complexType name=”motorTyp”>
<xsd:choice>
<xsd:element name=”kWLeistung” type=”xsd:positiveInteger”/>
<xsd:element name=”PSLeistung” type=”xsd:positiveInteger”/>
        </xsd:choice>
    </xsd:complexType>
```

Die Leistung ist also entweder in kW oder in PS anzugeben, nicht aber in beidem (weil man es ineinander umrechnen kann). Die Leistung eines Motors kann man also in einer dieser beiden Formen angeben:

```
<motor>
   <kWLeistung>100</kWLeistung>
</motor>
```

```
<motor>
   <PSLeistung>136</PSLeistung>
</motor>
```

Alle

Bei der Verwendung von “all” müssen alle genannten Kindelemente genau einmal vorkommen, wobei die Reihenfolge egal ist.

Häufigkeit

Häufigkeitsindikatoren legen fest wie oft ein Element vorkommen darf. Über minOccurs=”anzahl” und maxOccurs=”anzahl” lassen sich beliebige Ober- und Untergrenzen festlegen. Soll die Anzahl nicht beschränkt sein, verwendet man “unbounded”.

Gemischter Inhalt

Soll ein komplexer Typ neben enthaltenen Elementen auch Text erlauben (mit beliebiger Position), so muss man den Typ als “mixed” kennzeichnen.

```
<xsd:element name="bericht" type="berichtTyp"/>
<xsd:complexType name="berichtTyp" mixed="true">
   <xsd:sequence>
      <xsd:element name="fett" type="xsd:string" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
```

Hier wird ein Element namens bericht vom Typ berichtTyp beschrieben, das Text enthält und darüber hinaus beliebig viele Kindelemente namens fett vom Typ string. Es handelt sich also um einen solchen Text:

```
<bericht>
In diesem <fett>Bericht</fett> sind Teile des Textes
zwischendrin <fett>auch mal fett</fett> gedruckt. Es ist
also eigentlich ein Text mit ein paar <fett>Auszeichnungen</fett>.
</bericht>
```

Attribute

Um einem Element Attribute zu erlauben, muss es in jedem Fall als komplexer Typ definiert werden, auch wenn es nur einfachen oder sogar gar keinen Inhalt hat.

```
<xsd:element name="produkt">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:positiveInteger"/>
  </xsd:complexType>
</xsd:element>
```

Es gibt verschiedene Arten von Atriibuten:

- Pflichtattribute
- Eindeutige Attribute
- Verweisende Attribute
- Attribute mit festem Wert

Pflichtattribute

Um das Attribut zu einem Pflichtattribut zu machen, muss man noch eine use-Klausel hinzufügen:

```
<xsd:element name="produkt">
  <xsd:complexType>
    <xsd:attribute name="id"
          type="xsd:positiveInteger"
          use="required">
  </xsd:complexType>
</xsd:element>
```

.

Eindeutige Attribute

Ein Attribut kann auch vom Typ ID sein, dessen Inhalt dann auch als XML-Name zulässig sein muss, also nicht mit einer Ziffer beginnen darf. Die Werte der ID-Attribute von Elementen in einem Dokument müssen eindeutig sein – daher auch ”ID“ für Identifikation. Wenn man den Inhalt einer Relationen-Tabelle exportiert, könnte man hier einen künstlichen Primärschlüssel hineinschreiben. Die Eindeutigkeit würde dann nicht nur vom relationalen Datenbanksystem, sondern auch von der XML-Software sichergestellt, sofern sie auf Gültigkeit prüft.

Verweisende Attribute

Querverweise innerhalb eines Dokuments fordert man mit dem Typ IDREF, dessen Inhalt mit dem Inhalt des ID-Attributs eines anderen Elements übereinstimmen muss.

```
<xsd:element name="ccc">
   <xsd:complexType>
      <xsd:attribute name="x" type="xsd:ID" use="required"/>
      <xsd:attribute name="y" type="xsd:NMTOKEN" use="optional"/>
      <xsd:attribute name="z" type="xsd:IDREF" use="required"/>
   </xsd:complexType>
</xsd:element>
```

Attribute mit festem Wert

Soll ein Attributwert fest sein, so schreibt man in die Attributdefinition zusätzlich fixed= "festerWert " hinein. Diesen Wert hat das Attribut dann immer – egal, ob es im Dokument angegeben wurde oder nicht. Falls es im Dokument einen anderen Wert hat, so entspricht das Dokument nicht mehr dem Schema.

APIs zur Verarbeitung von XML

SAX

SAX (Simple API for XML) ist eine standardisierte Möglichkeit, wie eine XML-Datei durch einen Parser bearbeitet wird. Hierbei wird ein Datei-Strom in einen Strom von Ereignissen umgewandelt. Programme können sich für einzelne Ereignisse registrieren, um bei Bedarf ihre Arbeit zu verrichten. Die Eingabedaten werden rein sequentiell verarbeitet. Ein Vorteil von SAX ist, dass nicht die gesamte XML-Datei im Speicher vorgehalten werden muss. Das ist aber dann ein Nachteil, wenn man viele Informationen, die über die ganze Datei verstreut sind, zur Verarbeitung benötigt.

DOM

DOM (Document Object Model) ist der zweite Weg, um XML-Dateien auszuwerten und wurde vom W3C standardisiert. Er stellt, wie der Name schon sagt, ein standardisiertes Objektmodell zur Verfügung, mit dessen Hilfe der Inhalt der XML-Datei ausgewertet oder manipuliert werden kann. Zum Aufbauen des Objektbaumes muss jedoch zunächst die gesamte Datei eingelesen werden, wofür möglicherweise viel Speicher benötigt wird. Vorteilhaft ist hingegen, dass dann alle Elemente in einer hierarchischen Struktur vorliegen und auf alle gleichermaßen zugegriffen werden kann. Die Elemente stehen zueinander in Beziehung (Eltern, Geschwister, Kinder). Als Nachteil von DOM kann sich ein hoher Speicherbedarf erweisen; er verhält sich proportional zur Größe der Eingabedatei.

Als Beispiel sei eine Webseite erwähnt, die in XML spezifiziert ist. 100 kB sind hier schon eine beachtliche Größe, die Bearbeitung in einem DOM ist deshalb problemlos. Auf der anderen Seite kann ein Wörterbuch (3 MB Grunddaten) gegebenenfalls Probleme verursachen, wobei es weniger der Speicherplatz an sich sein dürfte sondern die Zugriffsgeschwindigkeit. Beide Modelle haben deshalb ihre Berechtigung in der Anwendung.

5. Anwendungsgebiete

Die XML-Sprachfamilie

So wie sich XML aus SGML entwickelt hat gibt es nun auch eine große Anzahl formaler Sprachen, die sich der Syntax von XML bedienen, also aus XML entwickelt wurden.

Eine kleine Übersicht:

Text-Sprachen

XSL-FO (Textformatierung)
DocBook (Bücher und Artikel vor allem im technischen Umfeld)
XHTML (XML-konformes HTML)
TEI (Text Encoding Initiativ, wird innerhalb der Geisteswissenschaften als quasi-Standard verwendet)

Graphik

SVG (Scalable Vector Graphics, später mehr dazu)
X3D (Extensible 3D, 3D-Modellierungssprache, mit der sich dreidimensionale virtuelle Welten, Lernanwendungen und Spiele realisieren lassen)

Geodaten

GML (Geography Markup Language)
GPX (GPS Exchange Format, XML für GPS-Daten)

Multimedia

SMIL
MPEG-7
Laszlo/LZX

Sicherheit

SAML
XML Signature

Anwendung in Programmen

XML wird auch in der Praxis für eine Reihe von Anwendungen verwendet.

Anwendungen

OpenOffice

Das freie Office-Paket OpenOffice.org verwendet gepackte XML-Dateien, was verschiedene Vorteile bietet. Da XML-Dateien nur plain text enthalten kann man sich sicher sein, sie auch in einigen Jahren noch problemlos öffnen zu können, womit z.B. Microsoft Word in der Vergangenheit bereits Probleme hatte. Des Weiteren lassen sich diese OpenOffice.org-Dateien entpacken und dann als gewöhnliche XML-Dateien benutzen, was sie leicht verwendbar und auch über die OO.org-Programme hinaus editierbar macht (z.B. mit einem eigenen Programm, das Tabelleninhalte automatisch einträgt).

Browser

Die meisten Browser (Firefox, Netscape, IE, Opera) haben mittlerweile integrierte XML-Parser, die das direkte Verwenden von XML-Dateien ermöglichen.

RSS

RSS ist ein Dateiformat für den XML-basierten Austausch von Nachrichten aller Art. Das Kürzel hat verschiedene Auslegungen erfahren, von *Rich Site Summary* oder *RDF Site Summary* bis zu *Really Simple Syndication*. Trotz der verschiedenen Erklärungen geht es bei RSS-Formaten immer darum, Informationen strukturiert abzulegen und sie für die automatisierte Verarbeitung durch RSS-Leseprogramme bereitzustellen. RSS wurde geschaffen, um Nachrichten von Internetportalen zu verbreiten, und hat sich inzwischen zu einem weitverbreitetem Standard für den automatisierten Austausch von Nachrichten und menschlicher Kommunikation (Weblogs, Diskussionsforen) entwickelt. Den Produzenten von RSS-Strömen ermöglicht das Format Interessenten einfach und schnell auf neue Inhalte hinzuweisen, für Leser ergibt sich durch die automatisierte Abfrage und Aufbereitung von RSS-Strömen eine große Zeitersparnis, da sie nicht mehr jede Webseite persönlich besuchen müssen und die aufbereiteten Formate nach den jeweiligen Interessenschwerpunkten filtern können.

Der folgende Beispielcode zeigt einen einfachen RSS-Feed auf Basis von RSS 2.0. Der kursive Text muss jeweils durch die eigenen Inhalte ersetzt werden.

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

<rss version="2.0">

  <channel>
    <title>Titel des Feeds</title>
```

```
    <link>Adresse der Webpräsenz</link>
    <description>Kurze Beschreibung des Feeds</description>
    <language>de-de</language>
    <copyright>urheberrechtliche Informationen</copyright>
    <pubDate>Datum der Erstellung</pubDate>
    <image>
      <url>Soll eine Grafik, z.B. ein Logo, eingebunden werden,
          hier dessen Adresse eintragen</url>
      <title>Titel des Bildes</title>
      <link>Adresse, mit der das Bild verknüpft werden soll, z. B.
           die Adresse der Webpräsenz des Herausgebers</link>
    </image>

    <item>
      <title>Titel des ersten Artikels</title>
      <description>Eine kurze Zusammenfassung des Artikels</description>
      <link>Adresse zur Gesamtansicht des Artikels</link>
      <author>Autor des Artikels</author>
    </item>

    <item>
      <title>Titel des zweiten Artikels</title>
      <description>
        <![CDATA[
        <h1>Hier kann auch der vollständige HTML-Inhalt
          des Artikels stehen</h1>
        <p>…</p>
        ]]>
      </description>
      <author>Autor des Artikels</author>
    </item>

    …

  </channel>

</rss>
```

SVG

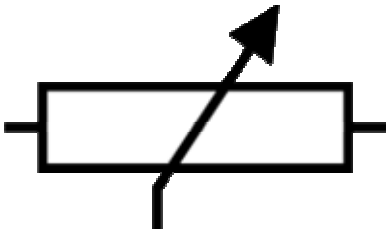
Scalable Vector Graphics (SVG) ist ein Standard zur Beschreibung zweidimensionaler Vektorgrafiken in der XML-Syntax.

SVG wurde im September 2001 vom W3C als Empfehlung eingestuft und wird von der nächsten Generation der gebräuchlichen Webbrowser größtenteils nativ unterstützt. Zur Zeit wird dazu oft ein Plugin wie z. B. der SVG-Viewer von Adobe benötigt.

Beispiel eines durch SVG erstellten Bildes:

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/TR/2003/REC-SVG11-20030114/DTD/svg11.dtd">
 <svg xmlns="http://www.w3.org/2000/svg"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      viewBox="-2 -5 105 55" version="1.1"
      width="107" height="60">
 <!-- Anschlüsse links und rechts -->
 <polyline points="0 25  100 25" fill="none" stroke="black"
      stroke-width="2.5"/>
 <!-- Das Widerstandsrechteck -->
 <rect x="10" y="15" width="80" height="20" fill="white" stroke="black"
      stroke-width="2.5" />
 <!-- Der Schleifer -->
 <polyline points="65 5  40 40  40 50" fill="none" stroke="black"
      stroke-width="2.5"/>
 <!-- Die Pfeilspitze wird gedreht -->
 <polygon points="60 5  70 5  65 -5" stroke="black" stroke-width="2.5"
      transform="rotate(33.7 65 5)" />
 </svg>
```

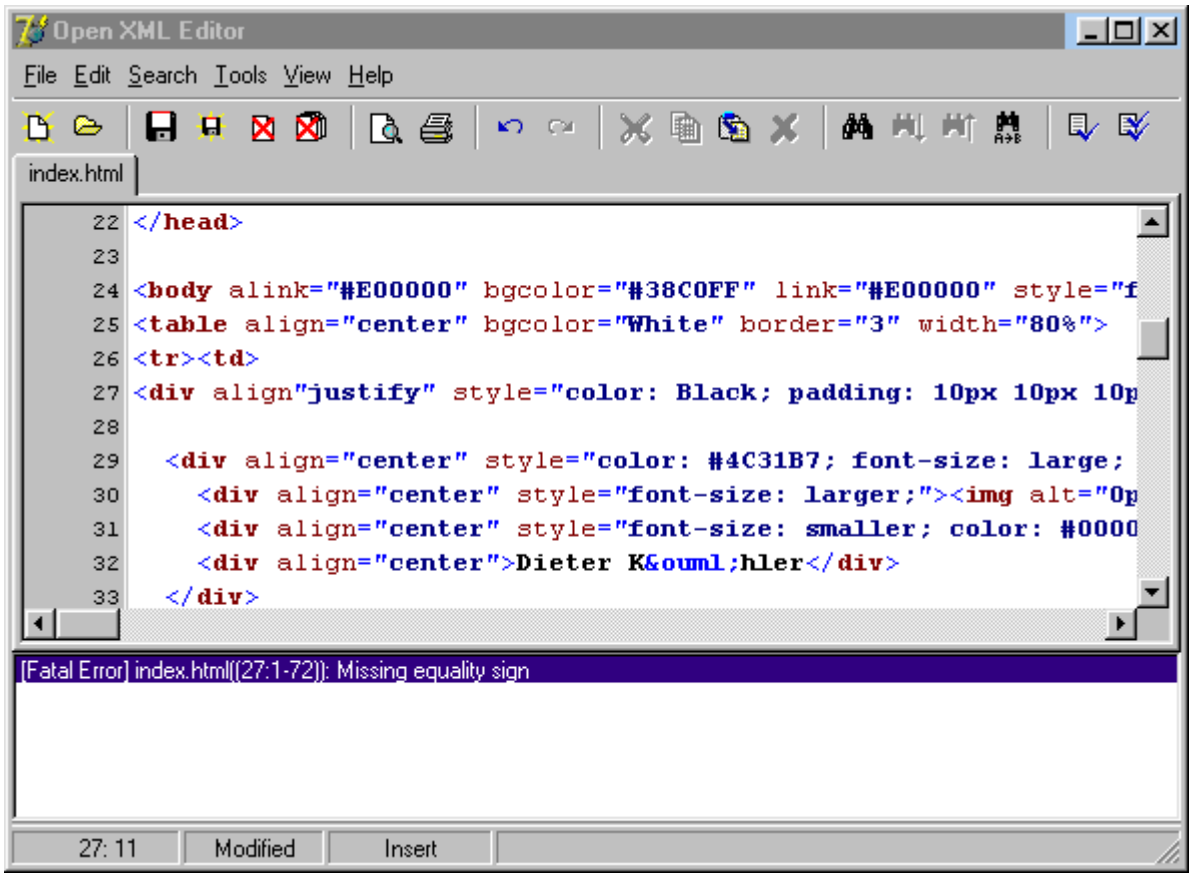
Das Bild sieht dann so aus und stellt einen variablen Widerstand dar:



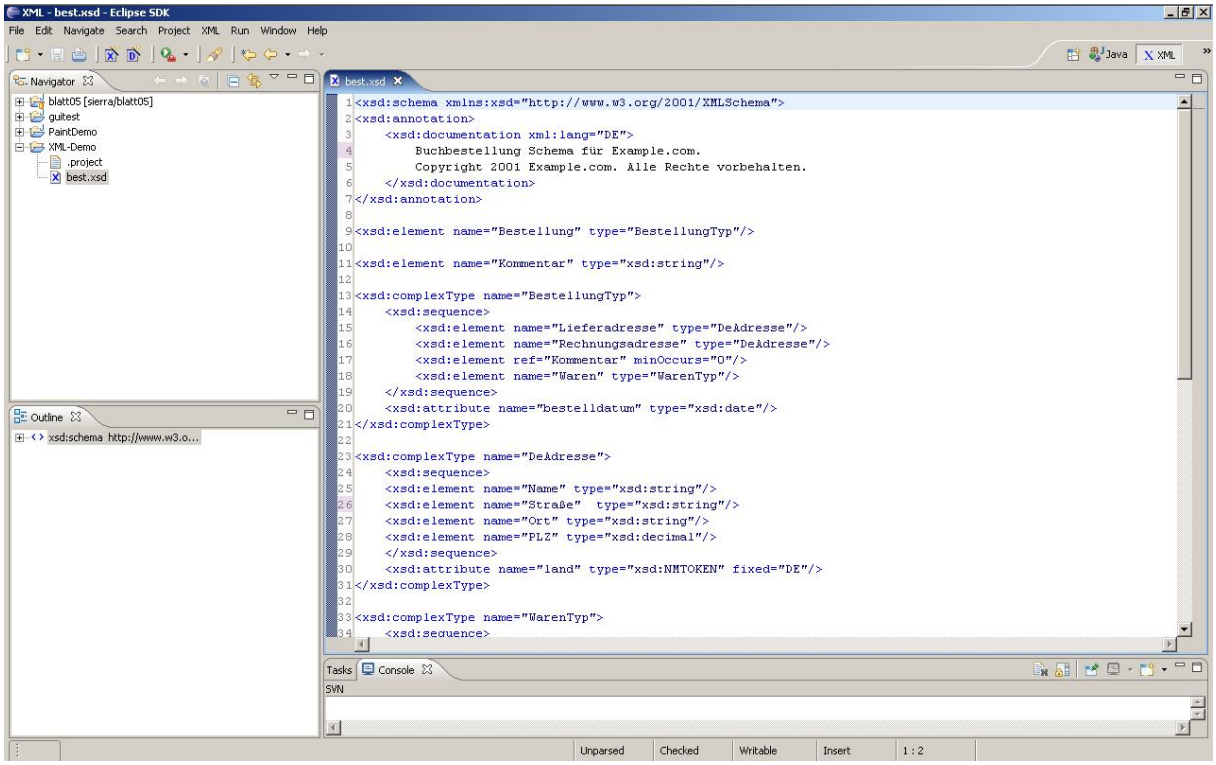
6. Editoren

Wie für die meisten Auszeichnungs- und Programmiersprachen gibt es auch für XML eine Reihe von Editoren. Das reicht im Open Souce Bereich von absolut rudimentären Programmen, wie dem “Open XML Editor” bis hin zu Plug-Ins für etablierte IDEs wie Eclipse (z.B. unter www.xmlbuddy.com/).
Des Weiteren gibt es eine Reihe von kommerziellen Editoren (z.B. Epic).

Open XML Editor



Eclipse


XML - best.xsd - Eclipse SDK
File Edit Navigate Search Project XML Run Window Help
Navigator
blatt05 [sierra/blatt05]
guitest
PaintDemo
XML-Demo
.project
best.xsd
Outline
xsd:schema http://www.w3.o...
best.xsd
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:annotation>
<xsd:documentation xml:lang="DE">
Buchbestellung Schema für Example.com.
Copyright 2001 Example.com. Alle Rechte vorbehalten.
</xsd:documentation>
</xsd:annotation>
<xsd:element name="Bestellung" type="BestellungTyp"/>
<xsd:element name="Kommentar" type="xsd:string"/>
<xsd:complexType name="BestellungTyp">
<xsd:sequence>
<xsd:element name="Lieferadresse" type="DeAdresse"/>
<xsd:element name="Rechnungsadresse" type="DeAdresse"/>
<xsd:element ref="Kommentar" minOccurs="0"/>
<xsd:element name="Waren" type="WarenTyp"/>
</xsd:sequence>
<xsd:attribute name="bestelldatum" type="xsd:date"/>
</xsd:complexType>
<xsd:complexType name="DeAdresse">
<xsd:sequence>
<xsd:element name="Name" type="xsd:string"/>
<xsd:element name="Straße" type="xsd:string"/>
<xsd:element name="Ort" type="xsd:string"/>
<xsd:element name="PLZ" type="xsd:decimal"/>
</xsd:sequence>
<xsd:attribute name="land" type="xsd:NMTOKEN" fixed="DE"/>
</xsd:complexType>
<xsd:complexType name="WarenTyp">
<xsd:sequence>
Tasks Console
SVN
Unparsed Checked Writable Insert 1 : 2

A. Anhang

Quellverzeichnis

[1] http://www.icaen.uiowa.edu/~bli/xml_proj/final-1.html
[2] http://www.spoono.com/xml/tutorials/tutorial.php?id=2
[3] http://de.selfhtml.org/
[4] http://de.wikipedia.org/wiki/XML
[5] http://www.w3schools.com/schema/default.asp
[6] https://www.bg.bib.de/portale/xml/pdf/XML-Schema.pdf
[7] http://de.wikipedia.org/wiki/XSD
[8] http://de.wikipedia.org/wiki/Parser
[9] http://de.wikipedia.org/wiki/Namensraum
[10] http://xmlbuddy.com/
[11] http://www.arbortext.com/html/epic_editor_overview.html
[12] http://de.wikipedia.org/wiki/RSS
[13] http://de.wikipedia.org/wiki/Scalable_Vector_Graphics
[14] http://www.et.fh-merseburg.de/person/meinike/PDF/TMs-SVGLitRes.pdf

Glossar

Attribute

Ein Attribut (v. lat: *attribuere* = zuteilen, zuordnen) wird gemeinhin als die Zuordnung eines Merkmals zu einem konkreten Objekt verstanden. Ein Attribut definiert und beschreibt ein konkretes Objekt. Zu einem Attribut gehört weiter ein ihm zugeordneter Wert, der Attributwert.

Ein Beispiel: Merkmal "Farbe" ist das Attribut, das ich beschreiben möchte, "Grün" wäre ein dazugehörender Wert, der die Farbe noch genauer definiert.

Die Werte, die man zuordnen kann, werden häufig eingegrenzt, der Wertbereich also vordefiniert, d.h. nur einige vorher festgelegte Möglichkeiten der Zuordnung sind erlaubt. Dies hat den Vorteil einer eindeutigen Zuordnung und einer einheitlicheren Sprache. Außerdem erspart es viel Arbeit beim Zuordnen.

Beispiel: Attribut "Farbe", Attributwertebereich "Rot, Blau, Grün, Gelb"

CSS

Cascading Style Sheets ist eine deklarative Stylesheet-Sprache für strukturierte Dokumente (z. B. HTML und XML), die vom World Wide Web Consortium (W3C) spezifiziert wird. Durch die Trennung von Stil und Inhalt wird das Veröffentlichen und Betreuen von Dokumenten wesentlich vereinfacht. CSS wurde vor allem im Hinblick auf HTML entwickelt, ist aber auch für XML-Dokumente anwendbar. CSS ermöglicht

es auch, Inhalte dem jeweiligen Ausgabemedium (z. B. Druck, Projektion, Sprachausgabe usw.) entsprechend abzuändern. Das ist nützlich, um zum Beispiel Weblinks beim Drucken extra aufzuführen und nicht (wie oft bei HTML-Seiten) zu verbergen. Oder um für ein Anzeigemedium wie einen PDA oder ein Mobiltelefon mit geringerer Auflösung die Anzeige zu optimieren (geringere Seitenbreite und -höhe).

Die *Fähigkeiten* von CSS sind vielfältig: Neben diversen, weit über HTML hinausgehenden Fähigkeiten im Farb- und Schriftbereich, die sich allerdings (noch) nicht mit spezialisierten Textsatzsprachen wie TeX messen können, bietet es etwa die Möglichkeit, alle Elemente frei zu positionieren oder Hintergrundbilder festzulegen.

CSS gilt heutzutage als die Standard-Stylesheet-Sprache für das Web.

Entitäten

Als Entität werden in der Informatik unterscheidbare, in der realen Welt eindeutig identifizierbare einzelne Objekte bezeichnet, über die ein Unternehmen Daten speichert. Die Objekte können sowohl eine physische ("real", z.B. "Konto Nr. 12345") oder eine konzeptionelle ("abstrakte", z.B. "Abteilung RK12") Existenz haben.

IDE

Eine integrierte Entwicklungsumgebung ist ein Anwendungsprogramm zur Entwicklung von Software. Unter Entwicklern wird meist die Bezeichnung IDE verwendet, eine Abkürzung des aus dem Englischen stammenden Begriffs *Integrated Development Environment* (auch *Integrated Design Environment*).

Namensräume

Namensraum (englisch name space) ist ein Begriff aus der Informatik.

Ein Name identifiziert ein Objekt. Zur eindeutigen Zuordnung ist jedoch der entsprechende Kontext - eben der Namensraum zu beachten. Die Beschreibung geschieht üblichweise durch die "Punkt"-Notation. Neben der "Punkt"-Notation sind aber auch andere Zeichen gebräuchlich, z. B.: bei Dateinamen "\" oder "/". Einige Namensräume (z.B. Dateisysteme) sind hierarchisch aufgebaut; d.h. Namensräume können selbst wieder aus Namensräumen bestehen. Namensräume werden dazu verwendet, Konflikte bei der Namensvergabe zu verhindern. Graphisch sind Namensräume mit Bäumen equivalent; d.h. Namensräume haben eine Wurzel (einen festen definierten Ausgangspunkt), Knoten (Verzeichnisse) und Blätter (Objekte).

Mit Namensräumen kann ein Autor große Programmpakete mit vielen definierten Namen schreiben, ohne sich Gedanken machen zu müssen, ob die neu eingeführten Namen in Konflikt zu anderen Namen stehen. Im Gegensatz zu der Situation ohne

Namensräume wird hier nicht der ganze Namen neu eingeführt, sondern nur ein Teil des Namens, nämlich der des Namensraumes (im Beispiel oben: mylib).

Die meisten modernen Programmiersprachen unterstützen Namensräume. Die Auszeichnungssprache XML unterstützt ebenfalls Namensräume, wobei dabei der Präfix durch einen Doppelpunkt vom lokalen Namen getrennt wird.

SGML

SGML (engl. Standard Generalized Markup Language) ist eine Metasprache, mit deren Hilfe man verschiedene Auszeichnungssprachen (engl. *markup languages*) für Dokumente definieren kann. SGML ist ein ISO-Standard: *ISO 8879:1986 Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*.