Gliederung

Extensible Markup Language (XML)

Wofür steht XML?

- XML = eXtensible Markup Language (erweiterbare Auszeichnungssprache)
- plattformunabhängiger Austausch von Daten
- selbsterklärende Daten

Wie sieht XML aus?

```
<?xml version="1.0"?>
<!--Kommentar und Verarbeitungsanweisungen sind hier erlaubt-->
<!DOCTYPE books SYSTEM "http://myserver.com/books.dtd">
<!-- weitere Kommentare und Verarbeitungsanweisungen erlaubt -->

<books>
        <book category="reference">
                <author>Nigel Rees</author>
                <title>Sayings of the Century</title>
                <price>8.95</price>
        </book>
        <book category="fiction">
                <author>Evelyn Waugh</author>
                <title>Sword of Honour</title>
                <price>12.99</price>
        </book>
</books>

<!--Kommentare und Verarbeitungsanweisungen sind hier erlaubt -->
```

Dokumentgliederung

- Prolog
- Rumpf
- Epilog

Elemente

- Markup-Zeichen (&, <) im Text verboten
- Namen müssen mit Buchstaben, Unterstrich oder Doppelpunkt anfangen.
- Zeichenfolge "xml" (in beliebiger Reihenfolge oder Groß-/Kleinschreibung) verboten
- Nutzung des Doppelpunktes vom W3C nicht empfohlen (Trennung zwischen Namensraum und lokalem Namen)

Attribute

- Name-Wert-Paare
- pro Element ein Attribut nur einmal erlaubt
- Spezialattribute
  - xml:lang
  - xml:space

Entity-Referenzen

- Einbinden von Textbausteinen an beliebigen Stellen
- Syntax: "&" + Entity-Name + ";"
- Definition von Entities im Abschnitt über DTDs

Vordefinierte Entities

Entity	Zeichen
&amp;	&
&apos;	'
&gt;	>
&lt;	<

&quot;	"

Verarbeitungsanweisungen (processing instructions)

- einzige Festlegung des W3C: syntaktischer Aufbau
- "<?" Anweisung "?>"
- Verhalten von Parsern nicht vorgeschrieben

Kommentare

- syntaktischer Aufbau: "<!--" viel wichtiger Kommentar "-->"
- Kommentartext nicht geparst -> kein Escaping nötig
- nicht erlaubt:
  - Kommentare innerhalb eines Tags
  - zwei Bindestriche innerhalb eines Kommentars

DocumentType Definition (DTD)

Konzept

- Vereinheitlichung und Verdeutlichung von Dokumentstrukturen
- Überprüfung der syntaktischen Gültigkeit
- Reduzierung des Validierungsaufwands in der Zielanwendung
- implizite Dokumentation
- Problemstellung für spez. Dokument festgelegt

Syntax

- eigene Syntax (nicht in XML-Notation)
- Schwächen bei komplexen Zusammenhängen
- gut geeignet, um einfache Dokumentstrukturen festzulegen

Operatoren

Operator	Bedeutung
,	Sequenz
|	Auswahl
?	Optionalität eines Elements
*	Beliebig häufiges Auftreten
+	Mindestens ein Vorkommen

einfache Elemente

- <!ELEMENT foo EMPTY>
- <!ELEMENT foo ANY>
- <!ELEMENT foo #PCDATA>

komplexe Elemente

- <!ELEMENT foo (bar)*>
- <!ELEMENT foo (A, (B | C)+, D?)>
- <!ELEMENT foo (A*, B?, (C+ | (D, E)) | F)>

Attribute

- <!ATTLIST foo bar CDATA "value" ent ENTITY #IMPLIED>
- <!ATTLIST foo (value1| value2| value3) #REQUIRED>

Vorgabewerte

- #IMPLIED
- #REQUIRED
- #FIXED
- #FIXED + Vorgabewert
- Vorgabewert

Attributypen

- ID
- IDREF
- IDREFS
- CDATA
- ENTITY
- ENTITIES
- NMTOKEN
- NMTOKENS
- NOTATION
- Wert aus einer Aufzählung

ENTITY

- Platzhalter für sich wiederholende Inhalte
- General-Entities und Parameter-Entities
- keine Leerzeichen (Whitespace) innerhalb einer Entityreferenz
- keine Zirkelbezüge zwischen Entities erlaubt

XML-Schema Definition Language (XSD)

Kennzeichen von Schema

- erweiterte Funktionalität im Vergleich mit DTD
- vollständig getypt
- eigene Datentypen durch Vererbung
- (Daten-)Integrität von Dokument wird sichergestellt

Schema-Deklaration

- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

Elementdeklaration

- <xsd:element name="foo" type="xsd:string">

vordefinierte Elemente (Auswahl)

Elementname	Beschreibung
all	alle Kindelemente genau einmal oder keinmal in beliebiger Reihenfolge
any	beliebiger wohlgeformter Inhalt
attribute	Deklaration eines Attributs
choice	Auswahl unter den Kindelementen. Es darf genau eins davon pro Knoten auftreten
complexContent	kein reiner Text als Kind erlaubt
complexType*	leitet einen complexType-Deklaration ein
element	leitet eine Element-Deklaration ein
enumeration	Aufzählungselement für Vorgabewerte
extension	Erweiterung eines Basistyps (Kind: restriction)
list	Kind von simpleType: Liste von Elementen
maxInclusive	Kind von restriction: maximal zulässiger Wert am Element / Attribut
maxLength	Kind von restriction: maximale Länge einer Liste
minInclusive	Kind von restriction: minmal zulässiger Wert am Element / Attribut
minLength	Kind von restriction: minimale Länge einer Liste
pattern	Kind von restriction: regulärer Ausdruck für erlaubte Element-/Attributwerte
restriction	Kind von simpleType: Beschränkung eines Basistyps auf zulässige Werte
schema	Einleitung des Schemas; Festlegung von Namensräumen
simpleContent*	nur simpleContent erlaubt (keine Attribute oder Elemente)
simpleType	Einleitung einer simpleType-Deklaration

simpleType

- beinhalten alle eingebauten Datentypen
- Grundlage für abgeleitete einfache Datentypen

```
<xsd:simpleType name="myInteger">
        <xsd:restriction base="xsd:integer">
                <xsd:minInclusive value="10000"/>
                <xsd:maxInclusive value="99999"/>
        </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="xsd:SKU">
        <xsd:restriction base="xsd:string">
                <xsd:pattern value="\d{3}-[A-Z]{2}"/>
        </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="listOfMyIntType">
  <xsd:list itemType="myInteger"/>
```

```
</xsd:simpleType>
```

Datentypen (Auswahl)

Datentyp	Wertebereich / Beispiel
string, normalizedString, token	Confirm this is electric
integer	..., -1, 0, 1, ...
positiveInteger	1, 2, ...
nonNegativeInteger	0, 1, 2, ...
unsignedLong	0, 1, ..., 18446744073709551615
byte	-128, ..., 0, ..., 127
boolean	true, false, 1, 0
dateTime	2005-11-16T19:30:00.000+2:00

complexType

- zusammengesetzter Datentyp aus:
  - Elementen
  - Attributen
  - Referenzen

```
<xsd:complexType name="USAddress">
        <xsd:sequence>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="street" type="xsd:string"/>
                <xsd:element name="city" type="xsd:string"/>
                <xsd:element name="state" type="xsd:string"/>
                <xsd:element name="zip" type="xsd:string"/>
        </xsd:sequence>
        <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

Extensible Stylesheet Language: Transformation (XSLT)

Konzept

- Umwandlung eines Ausgangsdokuments in ein Zieldokument unter Verwendung eines XML-Stylesheets
- Struktur des Eingangsdokuments wird bearbeitet (nicht die Daten)
- Arbeit wird durch XSL-Prozessor erbracht
- Daten werden Prozessor mittels API bereitgestellt
- bekannte APIs:
  - Document Object Model (DOM)
  - Simple API for XML (SAX)
- Selektion der Knoten mittels XPath

Syntax

- identisch zu XML
- spezieller Namensraum für XSL-Elemente: xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
- Anweisungen (Templates) zwischen <stylesheet>-Tags
- Auftrittsformen:
  - in XML-Dokument eingebettet
  - als separate Datei

XSL-Elemente (Auswahl)

Element	Beschreibung
apply-templates	Prozessor sucht nach passenden Templates
attribute	Attribut zu Element
call-template	spezielles Template nach seinem Namen aufrufen
choose	Einleitung einer Verarbeitungsauswahl (Kinder: when, otherwise)
copy	aktuellen Knoten in die Ausgabe kopieren
copy-of	einen Knoten (mit Kindern) in die Ausgabe kopieren
element	beliebiges Element in der Ausgabe erzeugen
for-each	Iteration über best. Elemente
if	bedingte Verarbeitung (kein else)
import	Einbinden eines externen Stylesheets (veränderter Baum)
include	Einbinden eines externen Stylesheets (reine Ersetzung)
otherwise	default-fall bei choose
output	Festlegen des Ausgabeformats (XML, HTML, Text)
param	Parameter bei Templates oder Stylesheet
sort	Sortierung einer Iteration (for-each) oder apply-templates
stylesheet	Einleitung des Stylesheets und Festlegung der Namensräume
template	Quasi Funktion in XSL. Sammlung von Verarbeitungsanweisungen für best. Elemente
value-of	Ausgabe eines Attribut- oder Elementtextes
variable	eine Variable
when	Option im choose (entspricht if, mehrere "when" möglich)
with-param	Parameterübergabe

XML-Query Language (XQuery)

- XQuery noch keine W3C-Empfehlung
- XQuery ist keine XML-Sprache. XQueryX eine in XML-Syntax verfasste XQuery-Sprache

Konzept

- Verarbeitungsmöglichkeit für verschiedenste Dokumentarten:
  - Dokumente für menschliche Leser (Indizierung, Suche)
  - datenorientierte Dokumente
  - Dokumente mit gemischtem Inhalt (Text + eingebette Daten)
- basiert auf XPath und XML-Schema-Datentypen
- liest / liefert ein Dokument-Fragment oder atomaren Wert

Anforderungen an W3C XML-Query Group

- Syntaxanforderungen:
  - MUSS leicht von Menschen lesbar sein
  - KANN mehr als eine Syntax haben
  - eine Abfragesprache MUSS in XML-Syntax verfasst werden
- Protokollunabhängigkeit MUSS gewahrt bleiben
- definierte Fehlerzustände MÜSSEN erzeugt werden können
- Aktualisierbarkeit MUSS gewahrt bleiben

Ausdrucksformen in XQuery

- Pfad-Ausdrücke
- FLWOR ("flower") Ausdrücke (For - Let - Where - Order - Return)
- Listenausdrücke
- bedingte Ausdrücke
- quantifizierte Ausdrücke
- Datentyp-Asudrücke

FLWOR-Beispiel

```
<beispiel>
        <bsp1>
        {
        for $d in doc("vortrag.xml")/präsentation/inhalt//kapitel
        where count($d/absatz) >= 5
        order by $d/@nummer descending
        return
          <kapitel überschrift="{$d/@titel}">
                 {
                 for $absatz in $d//absatz
                 where $absatz/überschrift != ''
                 return
                        <thema>{$absatz/überschrift/text()}</thema>
                 }
                </kapitel>
        }
        </bsp1>
        <bsp2>
                {
                        for $i in (1 to 5)
                        return <wert i="{$i}"/>
                }
        </bsp2>
</beispiel>
```

XML-Path Language (XPath)

Konzept

- gezielte Navigation durch XML-Dokument
- Selektion von Knoten
- Verwendung von location steps ausgehend von Kontext-Knoten
- Zusammensetzung eines Knotens aus:
  - Achse
  - Knotentest
  - Prädikat(e)

Achse

- unterteilt Dokument bezüglich Kontext-Knoten
- Start-Umgebung für Knotentest

Achse	Beschreibung
Child	alle Kindelemente
Descendant	alle Nachfahren
Parent	Elternelement
Ancestor	alle Vorfahren
Following-sibling	alle folgenden Geschwister
Preceding-sibling	alle vorangegangenen Geschwister
Following	alle nachfolgenden Knoten (keine Nachfahren, Attribute, Namensräume)
Preceding	alle vorangegangenen Knoten (keine Nachfahren, Attribute, Namensräume)
Attribute	alle Attribute
Namespace	alle Namesräume
Self	der Knoten selbst
Descendant-or-self	Vereinigungsmenge: Knoten plus alle Nachfahren
Preceding-or-self	Vereinigungsmenge: Knoten plus alle Vorfahren

Abkürzende Schreibweisen:

Achse	Abkürzung	Beispiel (lang)	Beispiel (kurz)
Child	(keine Angabe)	/child::Book	Book
Attribute	@	Book/attribute::title	Book/@title
Descendant-or-self	//	Book/descendant-or-self::author	Book//author
self	.	self::node()//author	.//author
Parent	..	parent::node()/Book	../Book

Knoten-Tests

- erlaubt spezifische Auswahl der Elementtypen innerhalb der Achsen
- verschiedene Arten von Knoten-Test:
  - Angabe des Elementnamens
  - Angabe eines "*": findet alle Elemente
  - node(): findet alle Knoten (auch z.B. Text)
  - text(): findet nur Textknoten
  - comments(): findet nur Kommentare
  - processing-instruction(): findet alle Verarbeitungsanweisungen

Prädikate

- Verfeinerung der Knoten-Tests
- boolescher Ausdruck; für jeden Knoten der Ergebnismenge ausgewertet

Beispiele

/Book//author	alle Autoren des (ersten) Buchs unterhalb der Wurzel

/Book[3]//author (/Book[position() = 3]//author)	alle Autoren des dritten Buchs unterhalb der Wurzel
/	die Dokument-Wurzel
//Book[./author = 'John Smith']	alle Bücher mit dem Autor 'John Smith'
//Book[@price < 50][@curr = 'euro']	alle Bücher, deren Preis weniger als 50 Euro ist