Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr. Theo Härder

XML

Techniken und Einsatzbeispiele

Seminararbeit

von

Marcus Trapp

Betreuer:

Dipl.-Inform. Budi Surjanto

Februar 2000

1. Einleitung 2
2. XML-Techniken 2
2.1. XML 2
2.1.1. Logische Dokumentstruktur 2
2.1.1.1. Prolog 3
2.1.1.2. Dokumentinstanz 3
2.1.1.2.1. Elemente 3
2.1.1.2.2. Attribute 4
2.1.2. Physikalische Dokumentstruktur 5
2.2. DTD 6
2.2.1. Wohlgeformtheit und Gültigkeit 7
2.2.2. Aufbau 7
2.2.2.1. Deklaration 7
2.2.2.2. Syntax 7
2.3. XLink 9
2.3.1. Elemente und Attribute 10
2.3.2. Einfache Verweise 11
2.3.3. Erweiterte Verweise 11
2.4. XSL 13
2.5. XPath 15
2.6. XPointer 16
2.7. Metadaten und RDF 17
3. Einsatzbeispiele von XML 18
3.1. E-Commerce 18
3.2. Dokument- und Content-Management 19
3.3. Informationssuche im Web 20
3.4. Web-Automation 21
4. Zusammenfassung 22
Literatur 22

1. Einleitung

Das World Wide Web gibt Autoren die Möglichkeit ihre Dokumente einem weltweiten Publikum vorzustellen. Es gibt viele Möglichkeiten solche Dokumente zu erstellen, doch ein großer Teil wird in *HTML (Hypertext Markup Language)* geschrieben. HTML basiert auf der Metasprache *SGML (Standard Generalized Markup Language)* und ist eine geeignete Sprache um einfache Dokumente darzustellen. Als jedoch die Größe und Komplexität der Web-Dokumente immer mehr wuchsen, stieß man sehr schnell an die Grenzen dieser Sprache.
Da SGML zu kompliziert ist und so nicht die benötigte Anerkennung und Verwendung fand, beschloß das *World Wide Web Consortium (W3C)* eine Sprache zu entwickeln, die einfacher zu lernen und anzuwenden ist aber auf den gleichen Grundgedanken basiert wie SGML: Trennung der Daten von ihrer Präsentation, Erweiterbarkeit und eine Möglichkeit zur strukturellen Validierung. Diese neue Sprache sollte einfach aufgebaut und maschinell leicht zu verarbeiten sein. So wurde die *Extensible Markup Language (XML)* als eine Teilmenge von SGML entworfen. Ein XML-Parser sollte so klein sein, daß er ohne Probleme in eine Anwendung, z.B. einen Web-Browser, integriert werden kann.
Die folgenden Abschnitte geben einen Überblick über die wichtigsten, mit XML zusammenhängenden Techniken. Im letzten Abschnitt werden einige Einsatzbeispiele von XML aufgezeigt.

2. XML-Techniken

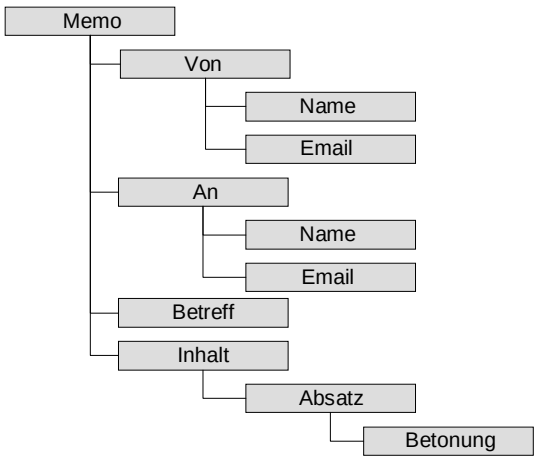
2.1. XML

In den beiden folgenden Abschnitten wird zuerst die logische und danach die physikalische Dokumentstruktur von XML vorgestellt [BPS98, GP98].

2.1.1. Logische Dokumentstruktur

Bevor exakt erklärt wird, wie die logische Dokumentstruktur von XML aufgebaut ist, wird hier zuerst einmal der Aufbau eines Dokuments im allgemeinen veranschaulicht.

Abbildung 2.1.



Wie es sich zeigt, kann man den Dokumentinhalt in Komponenten (z.B. `Von`, `An`, `Betreff`) zerlegen, die wiederum in Komponenten (z.B. `Name`, `Email`) zerlegt werden können. In XML heißen diese Komponenten *Elemente (elements)*. Sie bilden die *logische Struktur* des Dokuments. Ebenso kann man die meisten Dokumente grob in zwei Teile untergliedern. Zuerst kommt ein *Header*, in dem Informationen über das eigentliche Dokument stehen und erst dann folgt die Darstellung des eigentlichen Dokuments. Dies kann man in HTML gut erkennen, denn auch dort steht zuerst ein `HEAD`-Abschnitt und dann ein `BODY`-Abschnitt. Man kann dies aber ebensogut an einer E-Mail-Nachricht erkennen, die Überschriftszeilen hat, die beschreiben, von wem die Nachricht kam, an wen sie gerichtet ist und ähnliches mehr und erst dann folgt der eigentliche Inhalt der Nachricht. XML-Dokumente sind auch in dieser Art zweigeteilt. Sie bestehen aus einem *Prolog (prolog)* und einer *Dokumentinstanz (document instance)*, die dem eigentlichen Dokumentinhalt entspricht. Der Name

Instanz wurde gewählt, da es sich bei einem Dokument um eine Instanz eines Dokumenttyps handelt, die durch die *DTD* (*Dokumenttypdefinintion, engl. Document Type Definition)* des Dokuments definiert ist. Was genau eine DTD ist, wie sie aufgebaut ist und wozu man sie benutzt wird, in Abschnitt 2.2. erklärt.

2.1.1.1. Prolog

Ein XML-Dokument sollte also mit einem Prolog beginnen. Der Prolog besteht aus zwei Teilen: Der *XML-Deklaration (XML-declaration)* und einer *Dokumenttypdeklaration (document type declaration)*. Beide Teile sind optional. Verwendet man aber beide, so muß die XML-Deklaration vor der Dokumenttypdeklaration stehen.
Die XML-Deklaration gibt an, daß es sich um ein XML-Dokument handelt. In ihr muß die Versionsangabe (derzeit 1.0) stehen. Neben der Versionsangabe kann optional die *Kodierungsdeklaration* (*encoding declaration)* stehen, die den verwendeten Zeichensatz angibt und weiterhin eine optionale *standalone*-Angabe, die eigenständige Verarbeitung des Dokuments festlegt.
Die Dokumenttypdeklaration gibt an welche DTD verwendet. Diese befindet sich innerhalb desselben Dokuments oder in einer separaten Datei. Genaueres über die DTD folgt in Abschnitt 2.2.. Ein Beispiel für einen kurzen Prolog sind die ersten beiden Zeilen von Beispiel 2.2.. Er besagt, daß dieses Dokument mit der XML-Version 1.0 konform ist und eine bestimmte DTD `Memo` verwendet, die in der externen Datei `memo.dtd` liegt.

2.1.2. Dokumentinstanz

Die beiden folgenden Abschnitte befassen sich zuerst mit Funktion und Beschreibungsweise von Elementen und danach mit Attributen, mit denen man Elemente näher spezifizieren kann.

2.1.1.2.1. Elemente

Ein XML-Dokument besteht aus *Markup-Elementen (markup elements)* (kurz *Elemente, engl. elements*), die jeweils andere Elemente enthalten können. Ein Element kann leer sein oder Zeichenfolgen enthalten. In XML nennt man diese Zeichenfolgen *Zeichendaten (character data)*. Elemente ohne Inhalt werden als *leere Elemente (empty elements)* bezeichnet. Den in Abbildung 2.1. gezeigten Aufbau nennt man die *Baumstruktur (tree structure)* des Dokuments. Ist ein Element in einem anderen enthalten, so ist es ein *Unterelement (sub element)*. Das Element, daß alle anderen Elemente enthält heißt *Wurzelelement (root element)*. Manchmal wird das Wurzelelement auch *Dokumentelement (document element)* genannt. Die beiden Begriffe sind Synonyme.
Die Beziehung zwischen einem Element und seinen Unterelementen wird durch Kanten repräsentiert (vgl. Abbildung 2.1.). Als Alternative zur Baumstruktur bietet sich die Graphenstruktur an, bei der *Referenzen* zwischen den Elementen und *Verweise (Links)* als Kanten repräsentiert werden.
Die soeben genannten Begriffe und die Syntax von XML soll folgendes Beispiel verdeutlichen.

Beispiel 2.2.

```
<?xml version="1.0"?>
<!DOCTYPE Memo SYSTEM "http://www.fck.de/DTDs/memo.dtd">
<Memo>
     <Von>
         <Name>Otto Rehagel</Name>
         <Email>Otto_Rehagel@fck.de</Email>
     </Von>
     <An>
        <Name>Vorstand</Name>
        <Email>Vorstand@fck.de</Email>
     </An>
     <Betreff>Spielereinkauf</Betreff>
     <Inhalt>
            <Absatz>Vorstand, ich glaube wir
                    <Betonung>muessen</Betonung> unbedingt noch
                    einige neue Spieler kaufen, sonst wird das
                    wohl nichts mit der Meisterschaft.
             </Absatz>
     </Inhalt>
</Memo>
```

In diesem Beispiel ist das `Memo`-Element das Wurzelelement. Die Elemente `Von`, `An`, `Betreff` und `Inhalt` sind direkte Unterelemente von `Memo`.

Ein Markup-Element besteht aus einem Tag-Paar: dem *Beginn-Tag (start tag)* und dem *Ende-Tag (end tag)*. Die Zeichendaten eines Markup-Elements werden durch das Beginn- und Ende-Tag umrahmt. Der Name eines Markup-Elements ist im Beginn-Tag durch eine öffnende und schließende spitze Klammer eingeschlossen ("<",">"). Der Name eines Markup-Elements wird auch *Generischer Bezeichner (generic identifier)* genannt. Das zugehörige Ende-Tag wird wie das Beginn-Tag geschrieben, nur zusätzlich mit einem Schrägstrich ("/") nach der öffnenden Klammer. Zum Beispiel:

```
<Name>Otto Rehagel</Name>
```

Ein leeres Markup-Element wird wie in Beginn-Tag mit einem Schrägstrich vor der schließenden Klammer geschrieben. Zum Beispiel:

```
<Name/>
```

Wichtig ist auf die Groß- und Kleinschreibung zu achten. Im Gegensatz zu HTML unterscheidet XML nämlich sehr wohl zwischen generischen Bezeichnern die groß- bzw. kleingeschrieben sind oder eine Mischung aus beidem. Es wird zwar nicht vorgeschrieben, daß man alles groß oder alles kleinschreiben muß, aber man muß dann konsistent damit weiterarbeiten. Beginnt ein Element wie in Beispiel 2.2. mit `<Name>`, so muß es mit `</Name>` beendet werden und nicht mit `</name>`, `</NAME>` oder `</NaMe>`.

2.1.1.2.2. Attribute

Es gibt auch Elemente, die zusätzlich noch *Attribute (attributes)* haben, die das Element näher beschreiben. Attribute haben Namen und Werte.

Beispiel 2.3.

```
<Spieler Nummer="24">Harry Koch</Spieler>
<Spieler Nummer="24" Tore="5">Harry Koch</Spieler>
```

In einer DTD werden alle Attribute für einen Elementtyp definiert und bestimmt, welche Wertetypen sie annehmen können. Man unterscheidet drei verschiedene Attributwertetypen: Zeichendaten, Aufzählungen und Schlüsselwörter (z.B. `ID`, `IDREF` usw.). Dies wird in Abschnitt 2.2. erklärt. Innerhalb eines Elements darf der Name eines Attributs nur einmal verwendet werden. Es darf also keine zwei Attribute mit gleichem Namen innerhalb eines Elements geben. Verschiedene Elemente dürfen aber natürlich Attribute mit gleichem Namen haben.
Durch die Möglichkeit mit Attributen zu arbeiten, gibt es nun verschiedene Möglichkeiten ein und dasselbe darzustellen. In Bezug auf Beispiel 2.3. werden nachfolgend drei verschiedene Darstellungsmöglichkeiten gezeigt:

Beispiel 2.4.

(i) Version mit Unterelementen
```
<Spieler>
        <Nummer>24</Nummer>
        <Tore>5</Tore>
        <Name>Harry Koch</Name>
</Spieler>
```

(ii) Version nur mit Attributen
```
<Spieler Nummer="24" Tore="5" Name="Harry Koch"/>
```

(iii) Gemischte Version
```
<Spieler Nummer="24" Tore="5">Harry Koch</Spieler>
```

Es gibt selbstverständlich mehr als diese drei Möglichkeiten, um diesen Sachverhalt darzustellen, aber es gibt keine beste Möglichkeit. Da dies nicht der Fall ist, sollte man sich einmal für eine Möglichkeit entscheiden und diese dann auch beibehalten. Dies kann man durch die Verwendung einer DTD sicherstellen. Diese sorgt dafür, daß sich nicht nur dafür das verschiedene Dokumente des gleichen Typs immer gleich aufgebaut sind, sondern auch innerhalb eines Dokumentes bleibt der Aufbau immer gleich. Auf Beispiel 2.4. bezogen heißt das, daß nicht ein `Spieler` mit Version (i) und ein anderer mit Version (iii) beschrieben wird.

2.1.2. Physikalische Dokumentstruktur

In den vorangegangenen Abschnitten wurden Elemente und Attribute erklärt. Sie bilden die logische Struktur eines XML-Dokuments. Als nächstens werden nun *Entitäten (entities)* erklärt, die die *physikalische Struktur* eines XML-Dokuments ausmachen. XML-Dokumente können in mehreren Stücken (Entitäten) organisiert werden. Der XML-Parser setzt dann das eigentliche Dokument aus den einzelnen Entitäten zusammen.
Man könnte sich eine solche Entität als eine Schachtel mit einem Aufkleber mit ihrem Namen darauf vorstellen. Der Aufkleber wäre dann der *Entitätenname (entity name)* und der Inhalt der Schachtel, der *Entitäteninhalt (entity content)*. Um nun an den Inhalt der Schachtel zu kommen, muß man sie nur über ihren Namen ansprechen.
Der Entitäteninhalt kann nun wiederum Text sein und somit von einem XML-Prozessor geparst werden. Eine Entität mit geparstem Inhalt nennt man *geparste Entität (parsed entity)*. Eine *nicht geparste Entität (unparsed entity)* wird von einem XML-Prozessor nicht verarbeitet und kann daher neben Text auch andere Daten beinhalten, wie z.B. Graphik oder andere Multimedia-Daten.
Des weiteren unterscheidet man zwischen *externen (external)* und *internen (internal)* Entitäten. Man spricht von einer internen Entität, wenn der Entitätinhalt in dasselbe Dokument integriert ist, in der die Entität auch deklariert ist. Ein einfache interne Entität wäre z.B. eine Abkürzungsdeklaration.

```
<!ENTITY FCK "1. FC Kaiserslautern">
```

Wird der Inhalt von einer anderen Stelle im Web, oder auch aus einer anderen Datei, geholt, so spricht man von einer externen Entität. Hierzu ein einfaches Beispiel.

```
<!ENTITY FckLogo SYSTEM "http://www.fck.de/wapppen.gif" NDATA GIF>
```

In diesem Beispiel gibt `NDATA` an, daß es sich um eine nicht geparste Entität handelt, in einer *Notation*[1] `GIF`, d.h. in einem Bild.
Ein Autor eines XML-Dokuments verweist dann über eine *Entitätenreferenz (entity reference)* auf eine Entität, und der XML-Prozessor ersetzt diese Referenz durch den Inhalt. Diese Ersetzung wird *Inklusion (inclusion)* genannt. Eine solche Referenz hat die Form `&Entitätenname;`. Auch hierzu ein kurzes Beispiel:

```
<!ENTITY FCK "1. FC Kaiserslautern">
<Text>Otto Rehagel ist Trainer des &FCK;.</Text>
```

Weiterhin kann man zwischen *allgemeinen* und *Parameterentitäten (general* und *parameter entities)* unterscheiden. Allgemeine Entitäten können überall in einem Dokument stehen, wohingegen Parameterentitäten zum Anlegen von DTD verwendet werden. Parameterentitäten werden auch leicht anders deklariert als allgemeine Entitäten. Darauf wird hier allerdings nicht näher eingegangen (siehe [BPS98, GP98]).
Bei doppelter Deklaration einer Entität ist immer die erste gültig und alle folgenden werden ignoriert. Die *Dokumententität (document entity)* ist die Entität, in der der XML-Prozessor seinen Durchlauf startet. Sie wird an keiner Stelle deklariert, hat keinen Namen und es kann auch nicht auf sie verwiesen werden.

2.2. DTD

Wie es in den vorherigen Abschnitten schon an mehreren Stellen angesprochen wurde, sollte man zu seinem XML-Dokument eine DTD anlegen [BPS98, GP98]. Wozu braucht man eine solche DTD? Ein Mensch kann sofort bei einem ihm vorliegenden Dokument sagen, um was für ein Dokument es sich handelt. Ist es ein Buch mit vielen Namen und Telefonnummern, dann wird es sich um ein Telefonbuch handeln und nicht um ein Kochbuch oder eine E-Mail. Dokumente, die sich in vielen ihrer Markup-Elementen (vgl. Abschnitt 2.1.1.) unterscheiden, gehören mit großer Wahrscheinlichkeit nicht zum gleichen Dokumenttyp.
In XML kann man den Dokumenttyp mit einer DTD festlegen. Kurz gesagt, enthält eine DTD eine Reihe von Definitionen für Elemente, Attribute und Entitäten. Ähnlich wie Formulare, Dokument- oder Formatvorlagen helfen sie der Standardisierung. Man könnte sagen, eine strenge DTD, die keine Wahlmöglichkeit offen läßt, ist wie ein Formular und eine flexiblere DTD eher wie eine Formatvorlage, die z.B. besagt, daß eine Liste mindestens zwei Elemente haben muß oder Fußnoten nicht innerhalb von Fußnoten vorkommen dürfen.
Was aber in der Praxis von weit aus größerer Bedeutung an einer DTD ist, ist die zuverlässige Verarbeitung von Dokumenten durch Software. Hierzu ein kurzes Beispiel.
Bezogen auf Beispiel 2.2. könnte ein Benutzer wissen wollen von wem dieses Memo kommt. Sein Programm könnte dann einfach im `Von`-Element nachsehen und den Namen ausgeben. Was aber, wenn es gar kein `Von`-Element im XML-Dokument gibt? Oder was wäre, wenn es zwei Von-Elemente gäbe. Wie sollte sich das Programm dann verhalten? Eine DTD könnte dafür sorgen, daß so etwas nicht vorkommt.

[1] Notationen werden in verschiedenen Teilen eines XML-Dokuments verwendet, um Dateninhaltsnotationen zu beschreiben. Näheres hierzu findet man in [BPS98, GP98].

Selbst dieses kurze Beispiel zeigt auf, wie wichtig es sein kann, eine formale DTD anzulegen. Hat man sich dafür entschieden, sorgt XML dafür, daß man sich auch daran hält.

2.2.1. Wohlgeformtheit und Gültigkeit

XML zwingt sozusagen dazu, syntaktisch korrekte Dokumente zu erstellen. Den Begriff "korrekt" gibt es aber in XML nicht. Korrekt hat in XML zwei Bedeutungen. Wenn ein Dokument verständliches Markup hat, d.h. wenn es mit dem XML-Standard konform ist, dann ist es *wohlgeformt (well-formed)*. Der XML-Prozessor wird an einem Dokument, das nicht wohlgeformt ist, keine Korrektur versuchen. Es wird dann nicht verarbeitet. Die zweite Bedeutung von "korrekt" bezieht die DTD mit ein. Ein XML-Dokument ist *gültig (valid)* wenn es mit seiner DTD konform ist.
Ein Dokument das keine DTD hat ist aber nicht automatisch *ungültig*, denn es verletzt nicht seine DTD. Es ist aber auch auf keinen Fall gültig, da es nicht an seiner DTD validiert werden kann. Unter diesem Gesichtspunkt ist es besser von *typgerecht (type-valid)* und *nicht typgerecht (non-type-valid)* statt gültig und ungültig zu sprechen.

2.2.2. Aufbau

Nachdem nun die Funktion einer DTD und ihre Bedeutung für die Gültigkeit von XML-Dokumenten erklärt wurde, wird in den nächsten Abschnitten aufgezeigt, wie man eine DTD deklariert und mit welcher Syntax sie beschrieben wird.

2.2.2.1. Deklaration

Man unterscheidet zwischen einer *externen* und *internen* DTD. Bei einer externen DTD werden die DTD-Deklarationen in einer separaten Datei abgelegt, auf die die Dokumenttypdeklaration zeigt. In Beispiel 2.2. (2. Zeile) erfolgt die DTD-Deklaration ausschließlich extern.
Bei der internen Deklaration wird nicht auf eine andere Entität verwiesen, sondern die DTD-Deklarationen werden innerhalb desselben Dokuments untergebracht.
Wollte man beispielsweise die DTD aus Beispiel 2.2. intern deklarieren, würde `SYSTEM` wegfallen, und es stände an Stelle der URL der Inhalt von `http://www.fck.de/DTDs/memo.dtd`, eingegrenzt von zwei eckigen Klammern (`"["`,`"]"`) (vgl. Beispiel 2.5.). Dieser Teil heißt die *interne Untermenge (internal subset)*. Man kann auch eine Kombination aus beiden Ansätzen verwenden. So könnte man einen Teil extern auslagern und einen anderen intern deklarieren. In diesem Fall spricht man bei dem ausgelagerten Teil von der *externen Untermenge (external subset)*.

2.2.2.2. Syntax

Nach der Deklaration kommt nun eine Beschreibung der Syntax einer DTD. Bekanntermaßen besteht ein XML-Dokument aus Elementen. Damit ein XML-Dokument gültig sein kann, muß jedes seiner Elemente einem in der DTD deklarierten *Elementtyp* entsprechen. Eine Elementtypdeklaration hat beispielsweise die Form:

```
<!ELEMENT Email (#PCDATA)>
```

`(#PCDATA)` gibt hier die Art der Inhaltsspezifikation an. Es gibt insgesamt vier Arten von Inhaltsspezifikationen: *Leerer Inhalt (*`EMPTY` *content), Beliebiger Inhalt (*`ANY` *content), Kombinierter Inhalt (mixed content)* und *Elementinhalt (element content)*.
Wäre das Element `Email` aus dem obigen Beispiel ein leeres Element, so müßte man es wie folgt deklarieren:

```
<!ELEMENT Email EMPTY>
```

Bei einem Elementtyp mit beliebigem Inhalt, kann dieser sowohl Texte, als auch andere, beliebige Elemente enthalten. Man kann keine Aussage treffen. Auf obiges Beispiel bezogen:

```
<!ELEMENT Email ANY>
```

Elementtypen mit kombiniertem Inhalt können entweder aus Zeichendaten und untergeordneten Elementen bestehen oder nur aus Zeichendaten. Die zuletzt genannte Variante, die keine untergeordneten Elemente erlaubt, definiert man, auf das obige Beispiel bezogen, wie folgt:

```
<!ELEMENT Email (#PCDATA)>
```

Hierbei steht #PCDATA für *parsed character data*. Elementtypen die neben Zeichendaten auch untergeordnete Elemente zulassen könnte man wie folgt darstellen:

```
<!ELEMENT Email (#PCDATA|Element1|Element2)*>
```

Die beiden Zeichen "|" und "*" werden gleich beim letzten Elementtyp noch erklärt. Der Elementinhaltstyp erlaubt nur untergeordnete Elemente, die in der Elementinhaltsspezifikation enthalten sind und keine Zeichendaten. Hierbei wird in einem *Inhaltsmodell (content model)* angegeben, welche Unterelemente, in welcher Reihenfolge erlaubt sind. An dieser Stelle müssen noch verschiedene Zeichen erklärt werden.

,	Steht für eine *Sequenz* oder *Abfolge (sequence)*. Es gibt die Reihenfolge des Auftretens der Elemente an *(UND-Verknüpfung)*.
|	Steht für eine *Alternative (choice) (ODER-Verknüpfung)*.
?	Gibt an, daß diese Komponente der Elementinhaltsspezifikation optional ist (null- oder einmal).
*	Gibt an, daß diese Komponente der Elementinhaltsspezifikation optional und wiederholbar ist (null- oder mehrmals).
+	Gibt eine zwingend erforderliche Komponente der Elementinhaltsspezifikation an (ein- oder mehrmals).

Die letzten drei Zeichen ("?", "*", "+") bezeichnet man auch als *Indikatoren für das Auftreten (occurence indicators)*. Ein Beispiel für den letzten Elementtyp wäre dann:

```
<!ELEMENT Email (Element1,
                 (Element2, Element3)+,
                 Element4?)>
```

Dies bedeutet dann, daß das Element Email aus Element1 und dann aus einer beliebigen Folge von Element2 und Element3 besteht (eines von beiden mindestens einmal). Eventuell beinhaltet es als letztes Unterelement auch noch Element4.
Nachdem nun geklärt wurde, wie Elemente innerhalb einer DTD deklariert werden, kommen nun die Attribute der Elemente an die Reihe. Diese werden in einer *Attributlistendeklaration (attribute list declaration)* beschrieben. Bezogen auf Beispiel 2.4. (iii) könnte eine solche Deklaration wie folgt aussehen:

```
<!ATTLIST Spieler Nummer CDATA #REQUIRED 99
                  Tore   CDATA #REQUIRED 0>
```

Dabei gibt CDATA den *Attributtyp*, #REQUIRED den *Status* des Attributs und 99 bzw. 0 seinen *Default-Wert* an. Im allgemeinen sind sowohl der Status, als auch der Default-Wert nur optional anzugeben. Der Status eines Attributs kann folgende Werte annehmen:

#REQUIRED	Das Attribut ist erforderlich und muß somit angegeben werde.
#IMPLIED	Das implizite Attribut kann weggelassen werden. Es bekommt vom Anwendungsprogramm einen Wert zugewiesen.
#FIXED	Das Attribut hat einen fixen, nicht veränderbaren Wert.

Die Typen, die ein Attribut annehmen kann, sind in der folgenden Tabelle erklärt:

`CDATA`	Das einfachste Attribut. Es werde Zeichendaten erwartet.
Aufzählungstyp	Beliebig viele Auswahlmöglichkeiten können hier angegeben werden, ähnlich wie bei den Auswahllisten in der Elementtypdeklaration. Jede einzelne Auswahlmöglichkeit muß der Konvention von *name token*[2] entsprechen.
`NOTATION`	Muß einer speziellen *Notationsdeklaration* entsprechen.
`ID`	Muß innerhalb eines Dokuments eindeutig sein. Könnte man benutzen um z.B. einem bestimmten Abschnitt einen Namen zu geben und dann mit diesem Namen darauf zu verweisen. In diesem Fall würde man das Zielelement mit dem `ID`-Attribut kennzeichnen.
`IDREF`	Das auf das Element mit `ID`-Attribut verweisende Element wird mit `IDREF` markiert. Der Wert von `IDREF` muß die `ID` eines Elements sein.
`IDREFS`	Wenn ein Element auf mehrere Elemente, mit jeweiligem ID-Attribut verweist. Die Werte von `IDREFS` müssen die `ID`s von Elementen sein.
`ENTITY`	Muß eine deklarierte Entität sein.
`ENTITIES`	Müssen deklarierte Entitäten sein.
`NMTOKEN`	Muß der Konvention von *name token* entsprechen.
`NMTOKENS`	Muß der Konvention von *name token* entsprechen.

Nachdem nun erklärt wurde wie eine DTD aufgebaut ist, wird hier jetzt eine DTD für Beispiel 2.2. angegeben.

Beispiel 2.5.

```
<!DOCTYPE Memo[
      <!ELEMENT Memo (Von, An, Betreff, Inhalt)>
      <!ELEMENT Von (Name, Email)>
      <!ELEMENT An (Name, Email)>
      <!ELEMENT Name (#PCDATA)>
      <!ELEMENT Email (#PCDATA)>
      <!ELEMENT Betreff (#PCDATA)>
      <!ELEMENT Inhalt (Absatz+)>
      <!ELEMENT Absatz (#PCDATA|Betonung)*>
      <!ELEMENT Betonung (#PCDATA)>
]>
```

2.3. XLink

Nachdem bisher der Grundaufbau von XML erklärt wurde, folgt nun eine Beschreibung der darauf aufbauenden XML-Techniken. Den Anfang macht *XLink (XML Linking Language)*, der Verweismechanismus von XML [DOT00].
Im Vergleich zum Verweismechanismus von HTML, der nur einseitige Referenzierungen ermöglicht, bietet XLink darüber hinaus ein mächtigeres Verweiskonstrukt, wie Verweise zu multiplen Ressourcen, multi-direktionale Verweise und Verweise zu ausschließlich lesbaren Ressourcen. Weiterhin lassen sich XLink-Verweise durch verschiedene Attribute spezifizieren, die das Verhalten, die Semantik und die Richtung der Verweise festlegen.
An dieser Stelle sollte auf jeden Fall angemerkt werden, daß sich XLink immer noch in der Entwicklungsphase befindet und noch nicht zum Standard verabschiedet wurde.

[2] Bestimmten Dingen in XML muß man einen Namen gegeben. Die erlaubten Zeichenfolgen für Namen sind in der XML-Spezifikation festgelegt. Eines dieser Formate zur Namengebung ist ein *name token* (vgl. [BPS98, GP98]).

2.3.1. Elemente und Attribute

XLink stellt insgesamt sechs verschiedenen Elementtypen zur Verfügung, die im folgenden näher beschrieben werden:

- `simple`-Elemente verbinden genau eine *lokale Ressource (local resource)* und eine *entfernte Ressource (remote resource)*.
- `extended`-Elemente verfügen über die volle Funktionalität von XLink
- `locator`-Elemente beschreiben die entfernten Ressourcen, die am Verweis beteiligt sind.
- `arc`-Elemente stellen Regeln für die an der Traversierung beteiligten Ressourcen bereit.
- `title`-Elemente stellen Beschreibungen eines Verweises zur Verfügung, die von Menschen gelesen werden können.
- `resource`-Elemente beschreiben die lokalen Ressourcen, die am Verweis beteiligt sind.

Dabei werden die letzten vier Elementtypen nur im Zusammenhang mit der Verwendung des `extended`-Elementtyps eingesetzt. Der Einsatz von `simple`-Elementen wird in Abschnitt 2.3.2. näher beschrieben und der Einsatz der restlichen fünf Elementtypen in Abschnitt 2.3.3.. XLink-Elemente werden durch folgende acht Attribute spezifiziert:

- Das `type`-Attribut gibt den XLink-Elementtyp des Verweises an. Die möglichen Werte sind `simple`, `extended`, `locator`, `arc`, `resource` oder `title`.
- Das `href`-Attribut beschreibt entweder die Adresse eines `locator`-Elements oder die Adresse der entfernten Ressource eines `simple`-Elements. Sein Wert muß eine URI-Referenz sein *(Uniform Resource Identifier[3])*. Um auf bestimmte Teile innerhalb der entfernten Ressource zu zeigen, kann man einen *XPointer* (vgl. Abschnitt 2.7.) angeben.
- Das `role`-Attribut gehört zusammen mit dem nachstehenden `title`-Attribut zu den *semantischen Attributen (semantic attributes)*. Das `role`-Attribut wird verwendet, um die Funktion einer entfernten Ressource in einer maschinenlesbaren Art und Weise anzugeben.
- Das `title`-Attribut hat die gleiche Funktion wie das `role`-Attribut, nur das sein Wert für Menschen besser zu lesen ist.
- Das `show`-Attribut ist ebenso wie das nachstehenden `actuate`-Attribut, ein Attribut, das das Verhalten eines Verweises *(behavior attribute)* beschreibt. Das `show`-Attribut gibt an, wie die entfernte Ressource nach der Traversierung angezeigt werden soll. Die möglichen Werte sind:
  - `new` bedeutet, daß ein neues Fenster geöffnet wird, um die Ressource darzustellen.
  - `replace` gibt an, daß eine Ersetzung der bisherigen Ressource stattfindet.
  - `embed` bedeutet, daß die Ressource in die bereits geöffnete Ressource eingebunden wird.
  - `undefined` gibt ein noch nicht spezifiziertes Verhalten an.
- Das `actuate`-Attribut beschreibt, wann die Traversierung eins Verweises erfolgen soll. Mit dem Wert `onLoad` erfolgt die Traversierung automatisch mit dem Laden der Startressource, und mit dem Wert `onRequest` erfolgt sie erst auf Verlangen (z.B. durch anklicken des Benutzers). Auch hier gibt es wieder einen nicht näher spezifizierten Wert `undefined`.
- Das `from`-Attribut, ebenso wie das `to`-Attribut, kann nur in einen `arc`-Element angegeben werden. Es gibt an, von welchen Ressourcen aus eine Traversierung ausgelöst werden

[3] *URI* ist in [IETF RFC 2396] definiert.

kann *(starting resources)*. Als Wert hat es den Wert des `role`-Attributs eines `locator`- oder `resource`-Elements.
- Das `to`-Attribut gibt an, bei welchen Ressourcen die Traversierung endet *(ending resources)*.

Es ist anzumerken, daß die XLink-Spezifikation definiert, welche Attribute in welchen Elementtypen angewendet werden dürfen. Die Attribute sind global *(global attributes)* und können daher in jedem Markup-Element eingesetzt werden.

	simple	extended	locator	arc	resource	title
type	X	X	X	X	X	X
href	X		X			
role	X	X	X	X	X	
title	X	X	X	X	X	
show	X	X		X		
actutate	X	X		X		
from				X		
to				X		

2.3.2. Einfache Verweise

Ein *einfacher Verweis (simple link)* ist mit dem HTML-Element `A` vergleichbar, wobei er natürlich durch die oben beschriebenen Attribute zusätzliche Funktionalität erhält. Nachfolgend wird an einem Beispiel verdeutlicht, wie man einen einfachen Verweis deklariert, der mit einem `simple`-Elementtyp realisiert wird:

Beispiel 2.6.

In der DTD:
```
<!ELEMENT Spieler ANY>
<!ATTLIST Spieler
     xlink:type    (simple)             #FIXED     "simple"
     xlink:href    CDATA                #REQUIRED
     xlink:role    CDATA                #IMPLIED
     xlink:title   CDATA                #IMPLIED
     xlink:show    (new|embed|replace)            "replace"
     xlink:actuate (onLoad|onRequest)             "onRequest"
>
```

In der Dokumentinstanz:
```
<Spieler xlink:href="http://www.fck.de/Spielerliste.xml"
         xlink:role="playerlist"
         xlink:title="Liste aller FCK-Spieler"
         xlink:show="new"
         xlink:actuate="onRequest">
         Hier geht es zu einer Liste aller FCK-Spieler.
</Spieler>
```

2.3.3. Erweiterte Verweise

Erweiterte Verweise (extended links) bieten die volle Funktionalität von XLink. Ein erweiterter Verweis beinhaltet eine beliebige Mischung aus `locator`-, `arc`-, `title`- und `resource`-Elementen, mindestens aber ein `locator`-Element. Die `locator`-Elemente be-

schreiben die entfernten, und die `resource`-Elemente beschreiben die lokalen Ressourcen, die an dem Verweis beteiligt sind.
Bei einem erweiterten Verweis können mehr als zwei Ressourcen beteiligt sein. Außerdem kann sich ein erweiterter Verweis außerhalb der beteiligten Ressourcen befinden. Letzteres wird *Out-of-Line-Verweis (out-of-line link)* genannt. Bei Out-of-Line-Verweisen geht man davon aus, daß Information über *Verweisungen (external linksets)* existiert, welche angeben, wo die Verweise zu finden sind. Ein Out-of-Line-Verweis hat nur `locator`-Unterelemente. Sobald er ein `resource`-Element enthält, handelt es sich um einen *In-Line-Verweis (in-line link)*.
Out-of-Line-Verweise ermöglichen eine multi-direktionale Traversierung ebenso, wie eine Traversierung von einer Ressource zu starten, auf die der Benutzer, der den Verweis kreiert hat, nur Lesezugriffsrecht besitzt.
Da die erweiterten Verweise prinzipiell ungerichtet sind, kann man die Richtung der Traversierung durch `arc`-Elemente angeben. Das folgende Beispiel soll einige der Möglichkeiten von erweiterten Verweisen demonstrieren:

Beispiel 2.7.:

In der DTD:

```
<!ELEMENT WEBSITE (NAME, HOMESITE*, CONNECTION*) >
<!ATTLIST WEBSITE
  xmlns:xlink  CDATA     #FIXED
                           "http://www.w3.org/1999/xlink/namespace/"
  xlink:type  (extended) #FIXED  "extended"
  xlink:title  CDATA     #IMPLIED
  xlink:role   CDATA     #IMPLIED
>
<!ELEMENT NAME (#PCDATA)>
<!ATTLIST NAME
   xlink:type  (resource) #FIXED    "resource"
   xlink:role   CDATA     #IMPLIED
   xlink:title  CDATA     #IMPLIED
>
<!ELEMENT HOMESITE (#PCDATA)>
<!ATTLIST HOMESITE
   xlink:type  (locator)  #FIXED    "locator"
   xlink:href   CDATA     #REQUIRED
   xlink:role   CDATA     #IMPLIED
   xlink:title  CDATA     #IMPLIED
>
<!ELEMENT CONNECTION EMPTY>
<!ATTLIST CONNECTION
   xlink:type    (arc)      #FIXED  "arc"
   xlink:from     CDATA     #REQUIRED
   xlink:to       CDATA     #REQUIRED
   xlink:show    (new|
                  replace|
                  embed|
                  undefined)#IMPLIED
   xlink:actuate (onLoad|
                  onRequest|
                  undefined)#IMPLIED
>
```

In der Dokumentinstanz:

```
<?xml version="1.0"?>
<WEBSITE xmlns:xlink="http://www.w3.org/1999/xlink/namespace/"
         xlink:type="extended" xlink:title="International Fanclubs">
  <NAME xlink:type="resource" xlink:role="source">
    International Fanclubs
  </NAME>
  <HOMESITE xlink:type="locator"
            xlink:title="Fanclubs in Deutschland"
            xlink:role="de"
            xlink:href="http://www.FCK.de/clubs.xml"/>
  <HOMESITE xlink:type="locator"
            xlink:title="Fanclubs in Schweden"
            xlink:role="se"
            xlink:href="http://www.FCK-Schweden.se/clubs.xml"/>
  <HOMESITE xlink:type="locator"
            xlink:title="Fanclubs in der Schweiz"
            xlink:role="ch"
            xlink:href="http://www.FCK-Schweiz.ch/clubs.xml "/>
  <CONNECTION xlink:type="arc" xlink:from="source"
              xlink:to="ch"    xlink:show="replace"
              xlink:actuate="onRequest"/>
  <CONNECTION xlink:type="arc" xlink:from="source"
              xlink:to="se"    xlink:show="replace"
              xlink:actuate="onRequest"/>
  <CONNECTION xlink:type="arc" xlink:from="source"
              xlink:to="de"    xlink:show="replace"
              xlink:actuate="onRequest"/>
</WEBSITE>
```

Das `WEBSITE`-Element beschreibt einen erweiterten In-Line-Verweis mit 4 Ressourcen:

- Der Text "`International Fanclubs`", eine lokale Ressource
- Das Dokument `www.FCK.de/clubs.xml`, eine entfernte Ressource
- Das Dokument `www.FCK-Schweden.se/clubs.xml`, eine entfernte Ressource
- Das Dokument `www.FCK-Schweiz.ch/clubs.xml`, eine entfernte Ressource

Mit den Elementen `CONNECTION`, vom Typ `arc`, wird nun die Richtung der Traversierung angegeben. Dabei steht im `from`-Attribut immer der Wert des `role`-Attributs von der Ressource, von der der Verweis ausgeht, und im `to`-Attribut, der Wert des `role`-Attributs von der Ressource, auf die der Verweis zeigt. Wie genau eine Traversierung zu mehreren Ressourcen stattfindet, und wie ein solcher erweiterter Verweis im Browser angezeigt wird, ist noch nicht geklärt. Eine Möglichkeit wäre z.B. neue Fenster mit allen Ressourcen zu öffnen oder ein *Pull-Down-Menü* anzuzeigen, aus dem der Benutzer eine Ressource auswählt.

2.4. XSL

In den vorangegangenen Abschnitten wurde erklärt, wie ein XML-Dokument aufgebaut ist und wie man es syntaktisch richtig beschreibt. Danach wurde die Funktion einer DTD verdeutlicht, durch die man das Dokument strukturell validieren kann. Zuletzt wurde gezeigt wie man das Dokument mit Hilfe von XLink mit Verweisen versehen kann. Aber wie wird ein XML-Dokument dargestellt. Wie einleitend schon erklärt, ist in XML der Inhalt unabhängig von der Darstellung gespeichert. Das Element `Ueberschrift` hat nicht wie z.B. `h1` in HTML eine darstellende Bedeutung. Dies hat natürlich seine Vorteile. Dadurch wird eine medienunabhängige Präsentation von XML-Dokumenten ermöglicht. So könnte man z.B. ein

XML-Dokument mit der Liste aller Spieler des 1.FCK einmal im Web, auf einem Poster, auf einer CD-ROM oder in einem Lexikon auf unterschiedlichste Weise darstellen, ohne das Ursprungsdokument zu verändern, da im XML-Dokument nur der eigentliche Inhalt und nicht die Darstellungsform gespeichert ist.
Um XML-Dokumente auf diese verschiedenen Arten präsentieren zu können wurde *XSL (Extensible Stylesheet Language)* entwickelt [Dea99, Cha99]. XSL ist eine Sprache für die Transformation und Formatierung von XML-Daten. Nachfolgend wird das Prinzip von XSL anhand von *XSLT (XSL Transformation Language)* erklärt. Die Grundidee von XSLT ist die Ersetzung der Tags im XML-Dokument entsprechend einer passenden *Formatvorlage (stylesheet)*. Eine XSLT-Formatvorlage ist eine Ansammlung von *Transformationsregeln (transformation rules)*, die auf dem ursprünglichen XML-Dokument arbeiten und ein neues XML-Dokument erzeugen. Dabei wird das ursprüngliche Dokument nicht zerstört oder verändert. Da eine Baumstruktur für die Transformationsverarbeitung benutzt wird, wird das Ursprungsdokument auch als *Ursprungsbaum (source tree)* und das Ergebnisdokument als *Ergebnisbaum (result tree)* bezeichnet.
Eine Regel besteht aus einem *Muster (pattern)* und einer *Schablone (template)*. Das Anwenden der Regeln beginnt mit der Regel für das Wurzelelement. Es wird also solange der Ursprungsbaum durchsucht, bis ein *Knoten (node)* gefunden wird auf den ein Muster paßt. Anhand der passenden Schablone wird dann ein Stück des Ergebnisbaums erzeugt.
Hier erst einmal ein kurzes Beispiel:

Beispiel 2.8.: Konvertierung eines XML-Dokuments in *XHTML*[4]**:**

Ein Fragment eines XML-Dokuments:

```
<Inhalt>
       <Absatz>XSLT <Fremdwort>(XSL Transformation Language)</Fremdwort>
               ist eine <Betonung>phantastische</Betonung> Sprache um
               XML-Dokumente in XHTML <Fremdwort>(Extensible HTML)
              </Fremdwort>zu konvertieren.
       </Absatz>
</Inhalt>
```

Das XSLT-Stylesheet:

```
<xsl:stylesheet xmlns:xsl="http://w3.org/XSL/Transforam/1.0"
                xmlns="http://w3.org/TR/xhtml1"
                indent-rules="yes">
<!-- Regel 1 --> <xsl:template match="/">
                     <html>
                     <head>
                       <title>Ein XSLT-Beispiel</title>
                     </head>
                     <body>
                       <xsl:apply-templates/>
                     </body>
                     </html>
                  </xsl:template>
<!-- Regel 2 --> <xsl:template match="Absatz">
                     <xsl:apply-templates/>
                  </xsl:template>
<!-- Regel 3 --> <xsl:template match="Betonung">
                     <b><xsl:apply-templates/></b>
                  </xsl:template>
<!-- Regel 4 --> <xsl:template match="Fremdwort">
                     <i><xsl:apply-templates/></i>
                  </xsl:template>
```

[4] *XHTML (Extensible Hypertext Markup Language)* ist eine Reformulierung von HTML 4.0 als eine XML-Anwendung [XHT99].

Das Ergebnis-Dokument:

```
<html>
  <head>
    <title>Ein XSLT-Beispiel</title>
  </head>
  <body>
     XSLT <i>(XSL Transformation Language)</i> ist eine
     <b>phantastische</b> Sprache um XML-Dokumente in XHTML
     <i>(Extensible HTML)</i> zu konvertieren.
  </body>
</html>
```

Die Präsentation im Browser:
XSLT *(XSL Transformation Language)* ist eine **phantastische** Sprache um XML-Dokumente in XHTML *(Extensible HTML)* zu konvertieren.

Jedes `template`-Element beschreibt eine Transformationsregel. Sein `match`-Attribut beschreibt das Muster und sein Inhalt die Schablone. Das Muster `"/"` in `Regel 1` bezieht sich auf das Wurzelelement `Inhalt`. Das `apply-templates`-Element ist eine XSLT-Anweisung, die besagt, daß der Inhalt des Elements, zu dem das Muster paßt, rekursiv weiterverarbeitet wird. Es gibt auch noch andere solche Anweisungen, die hier aber nicht weiter erklärt werden (z.B. `for`- oder `repeat`-Schleifen). Knoten, also Elemente, Attribute und Zeichendaten, auf die kein Muster paßt werden von XSLT einfach in den Ergebnisbaum kopiert. Im obigen Beispiel sind das alle Zeichendaten.
Haben mehrere Regeln das gleiche Muster, so wird die Auswahl der anzuwendenden Regel durch die Prioritätsangaben im jeweiligen `priority`-Attribut entschieden. So könnte eine Regel mit `priority="0.5"` heißen "Setze alle Absätze im `Inhalt`-Element in Schriftgröße 8". Eine später stehende Regel mit `priority="1.0"` besagt aber "Setze den ersten Absatz im `Inhalt`-Element in Schriftgröße 10". Auf diesen ersten Absatz treffen nun beide Regeln zu. So wird für den ersten Absatz die zweite Regel und für alle anderen die erste Regel verwendet. Wären allerdings die Muster beider Regeln exakt gleich, dann soll ein XSLT-Prozessor entweder eine Fehlermeldung liefern oder die letzte Regel auswählen.

2.5. XPath

Im vorangegangenen Abschnitt wurde erklärt wie XSLT innerhalb des Ursprungsdokuments nach bestimmten Mustern sucht. Diese Muster können einfach nur die Elementnamen sein oder auch komplexere Sachverhalte betreffen. Wie beschreibt man nun ein solches Muster, wie z.B. "Der erste Absatz im `Inhalt`-Element", das im vorangegangenen Abschnitt schon genannt wurde. Dies geschieht mit Hilfe von *XPath (XML Path Language)* [CD99, Cha99]. XPath ist die Sprache zum Beschreiben von Mustern in XSLT. Mit XPath kann man bestimmte Knoten *absolut*, d.h. ihre Position bezüglich des Wurzelelements, oder *relativ* bezüglich eines *Kontextknotens (context node)* angeben.
Die Beschreibung einer Position erfolgt mit Hilfe eines *Pfadausdrucks (path expression)*. Ein Pfadausdruck besteht aus mehreren *Anweisungen (steps)*, die durch Schrägstriche (`"/"`) getrennt sind, wobei eine Anweisung aus einer *Basis (basis)* besteht, gefolgt von einer optionalen Liste von *Prädikaten (predicates)*. Eine Basis besteht aus einer *Achse (axis)* und einem *Knotentest (node test)*. Sie hat die Form *Achsenname::Knotentest*. Anhand des folgenden Beispiels wird die Funktionsweise von XPath erklärt.

```
id("Mannschaft")/descendant::Spieler/child::Name[position()=last()]
```

Zuerst wird der einzige Knoten im Ursprungsbaum gesucht, dessen `ID`-Attribut den Wert "Mannschaft" enthält. Dies ist der Kontextknoten. Die Basis, mit Achse `descendant` und Knotentest `Spieler`, liefert eine Liste von allen `Spieler`-*Nachfahren* (Unterelementen) des Kontextknotens, in der Reihenfolge, wie sie im Ursprungsdokument auftreten. Von diesen `Spieler`-Elementen ausgehend, wird nun eine Liste aller `Name`-*Nachfolger* (direkte Unterelemente) zurückgeliefert, die wiederum durch das Prädikat `[position()=last()]` gefiltert wird, so daß nur das jeweils letzte `Name`-Nachfolger der `Spieler`-Elemente zurückgeliefert wird. Die Funktionen `position` und `last` sind aus der *XPath core function library*, die noch weitere Funktionen enthält, um Listenoperationen durchzuführen. Diese werden hier aber nicht weiter erklärt.
Die Bedeutung der Achsen `descendant` und `child`, sowie die weiteren, von XPath zur Verfügung gestellten, Achsen werden in der folgenden Tabelle beschrieben:

`descendant`	Gibt alle Nachfahren des Kontextknotens an, d.h. alle Unterknoten mit deren Unterknoten usw.*
`child`	Gibt alle Nachfolger des Kontextknotens an, d.h. alle direkten Unterknoten.*
`following`	Gibt alle Knoten nach dem Kontextknoten an, ohne deren Nachfahren.*
`following-sibling`	Gibt Geschwisterknoten nach dem Kontextknoten an.*
`ancestor`	Gibt alle Vorfahren des Kontextknotens an, d.h. den Vorgänger, dessen Vorgänger usw.**
`parent`	Gibt den direkten Vorgänger des Kontextknotens an.
`preciding`	Gibt alle Knoten vor dem Kontextknoten an, ohne deren Vorfahren.*
`preciding-sibling`	Gibt Geschwisterknoten vor dem Kontextknoten an.**
`self`	Gibt den Kontextknoten an.
`descendant-or-self`	Gibt den Kontextknoten und seine Vorfahren an.*
`ancestor-or-self`	Gibt den Kontextknoten und seinen Nachfahren an.**

* In der Reihenfolge des Auftretens im Ursprungsdokument ** In umgekehrter Reihenfolge

Die Syntax von Pfadausdrücken ist sehr umfangreich. Neben der Möglichkeit auch noch die Attribute der Elemente mit einzubeziehen, gibt es auch noch eine *abgekürzte Syntax (abbreviated syntax)*. Auch hier wird wieder auf [CD99, Cha99] verwiesen.

2.6. XPointer

Im vorigen Abschnitt wurde erklärt, wie man mit XPath auf bestimmte Elemente in XML-Dokumenten verweisen kann. Will man aber nicht nur auf ein bestimmtes Element verweisen, sondern auf eine Region, z.B. einen Abschnitt, ein bestimmtes Wort oder sogar nur auf einen Buchstaben, dann geschieht dies mit Hilfe von *XPointer (XML Pointer Language)* [DD99]. XPointer erweitert die Funktionalität von XPath in zweierlei Beziehung.
Mit der `range`-Achse ist es möglich nicht nur ein Element, sonder einen ganzen Bereich zwischen zwei Elementen anzugeben.

```
range::LocationPath1, LocationPath2
```

Mit dieser Anweisung wird der gesamte Bereich zwischen `LocationPath1` und `LocationPath2` angesprochen. Eventuell tritt `LocationPath1` an verschiedenen Stellen im ursprünglichen Dokument auf. Der Bereich wird jedoch erst gefunden, wenn jedem dieser Elemente auch jeweils genau ein `LocationPath2` folgt. Hierzu ein Beispiel:

```
id("Mannschaft")/range::Spieler[1],following-sibling::Spieler[24]
```

Dabei zeigt der XPointer auf die Region der ersten 24 `Spieler`-Unterelemente eines `Mannschaft`-Elements.
Die zweite Erweiterung von XPointer ist die `string`-Achse. Die `string`-Achse arbeitet nicht mit Elementen (Knoten im Ursprungsbaum), sondern mit Zeichendaten.

`string::n,M,p,l` (`n`, `p`, `l` sind ganze Zahlen, `M` ist eine Zeichenfolge)

Die obige Anweisung zeigt auf eine Zeichenfolge, die `l` Zeichen lang ist. Sie beginnt mit dem `p`-ten Zeichen, vor den letzten Zeichen von `M`, beim `n`-ten Vorkommen von `M` im Ursprungsdokument. Wenn man z.B. auf die Zeichenfolge zeigen möchte, die 10 Zeichen vor dem dritten Auftreten der Zeichenfolge "Spieler" anfängt und 10 Zeichen danach aufhört, so wäre die `string`-Anweisung:

```
string::3,"Spieler",17,27
```

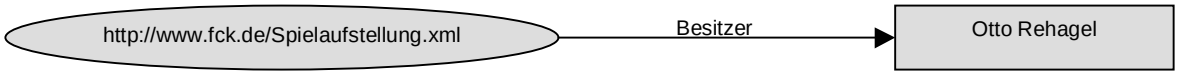
Mit einem XPointer ist es somit auch möglich, auf Regionen eines XML-Dokuments zu verweisen, die nicht wohlgeformt sind.

2.7. Metadaten und RDF

Im folgenden Abschnitt wird *RDF (Resource Description Framework)* vorgestellt, das ermöglicht, *Metadaten* von Ressourcen zu beschreiben [Cha99]. Metadaten sind "Daten über Daten". Die Aufschrift auf einer Videokassette "Jugendgefährdender Inhalt" sind z.B. Metadaten, da sie den Inhalt der Videokassette beschreiben. Ebenso könnte man Fernsehzeitschriften als Sammlung von Metadaten bezeichnen, da sie den Inhalt der Sendungen beschreiben. Im Web beschreiben Metadaten die Daten in Web-Dokumenten. Durch ein solches Modell kann die Effektivität von Suchmaschinen gesteigert werden oder ein verbessertes Bereitstellen bzw. Austauschen von Informationen erzielt werden.
Eine *RDF-Anweisung (statement)* ist ein Trippel aus einem *Subjekt (subject)*, also der zu beschreibenden Ressource, einem *Prädikat (predicate)*, das eine Eigenschaft angibt und einem *Objekt (object)*, das den Wert der Eigenschaft angibt. Möchte man z.B. darstellen, daß Otto Rehagel Besitzer einer Datei `http://www.fck.de/Spielaufstellung.xml` ist, dann müßte man folgendes Trippel angeben:

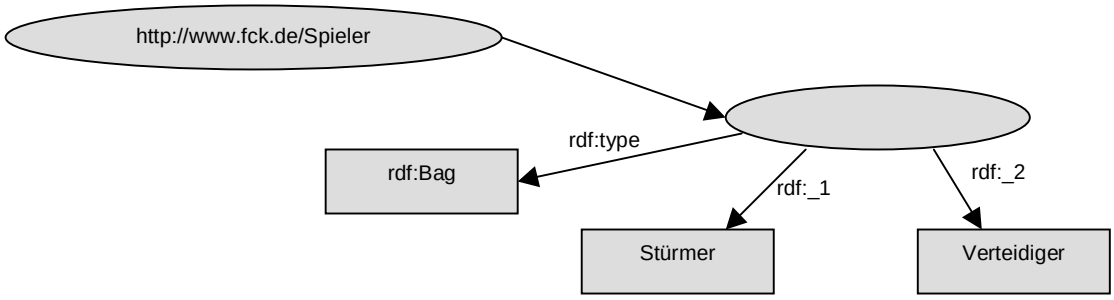
```
(Subjekt:http://www.fck.de/Spielaufstellung.xml;
 Prädikat:Besitzer; Objekt:"Otto Rehagel")
```

In der graphischen Repräsentation einer RDF-Anweisung werden Subjekt durch Ovale, Objekte durch Rechtecke und Prädikate durch gerichtete Pfeile dargestellt:



Ein Objekt muß aber nicht immer nur aus Zeichendaten bestehen. Es kann auch wieder ein eine Ressource sein oder sogar eine Liste von Ressourcen. Eine solche Liste wird in RDF durch einen *Container (container)* dargestellt. Es gibt drei Typen von Containern: *Tasche (bag), Sequenz (sequence)* und *Alternative (alternative)*. Hierzu ein Beispiel:

Abbildung 2.9.



RDF stellt eine konkrete Syntax zur Beschreibung von RDF-Anweisungen in XML zur Verfügung. Die beiden obigen Beispiele wären dann wie folgt zu beschreiben:

Beispiel 2.10.

```
<rdf:Description about="http://www.fck.de/Spielaufstellung.xml">
     <bs:Besitzer>Otto Rehagel</bs:Besitzer>
</rdf:Description>

<rdf:Description about="http://www.fck.de/Spieler">
     <bs:Positionen>
          <rdf:Bag>
               <rdf:li>Stürmer</rdf:li>
               <rdf:li>Verteidiger</rdf:li>
          </rdf:Bag>
     </bs:Positionen>
</rdf:Description>
```

Hiermit endet nun die Beschreibung der XML-Techniken. Nachfolgend werden noch einige Einsatzbeispiele von XML gegeben, um einen Überblick über das Anwendungsgebiet von XML zu gewinnen.

3. Einsatz von XML

Nachdem im Abschnitt 2. die verschiedenen Techniken von XML beschrieben wurden, werden in den folgenden Abschnitten einige Einsatzgebiete von XML aufgezeigt.

3.1. E-Commerce

E-Commerce ist eines der Schlagworte im Moment. Wie kann also XML E-Commerce unterstützen? Nachdem *EDI (Electronic Data Interchange)* nicht den erwarteten Erfolg brachte, erhofft man sich nun durch das Internet eine breitere Akzeptanz der Unternehmen, vor allem auch der kleinen und mittelständischen Unternehmen. Ein großes Problem bei bisherigen EDI-Anwendungen sind die enormen Kosten für die Einführung des Systems (durchschnittlich mindestens 50.000 US$) und das Fehlen von einheitlichen Standards. Bekommt man einen neuen Geschäftspartner, der einen anderen EDI-Standard verwendet, entstehen weitere Kosten für einen Anpassung. An diesen Punkten setzt XML an. XML ist plattformunabhängig. Die Syntax von XML kann sowohl von Computern als auch von Menschen verstanden werden und ermöglicht das Erstellen von selbst-beschreibenden Daten. Weiterhin läßt sich die Struktur dieser Daten validieren. Durch das XML-Format können Dokumente von verschiedenen Systemen wesentlich leichter ausgetauscht werden als dies vorher der Fall war. Denn das ist eines der Hauptziele im E-Commerce: Daten von einem Unternehmen zu einem

anderen zu schicken, so daß dieses dann die Informationen in jedem seiner Programme nutzen kann, ohne jemals einen Medienbruch, wie z.B. einen Ausdruck, zu machen.
XML stellt also beste Möglichkeiten zur Verfügung, um eigene Formate zu definieren, die auf dem XML-Standard aufbauen. Das Ziel kann aber nicht sein, daß nun jedes Unternehmen eigene Formate für jedes ihrer Dokumente entwirft. Dies wäre dann ein großer Schritt rückwärts. Im Bereich E-Commerce arbeitet *VEO Systems* daran eine Bibliothek, *CBL (Common Business Library)*, zu erstellen, in der standardisierte DTD-Bausteine zur Verfügung gestellt werden, die dann in E-Commerce-Dokumenten Anwendung finden sollen. So könnte man, um ein einfaches Beispiel zu nennen, ein Format für Adressen anbieten, so daß jedes Unternehmen in ihren Dokumenten ihre Adressen in diesem Format angeben könnte und auf diese Weise den oben genannten Anpassungen an andere Formate, anderer Unternehmen, aus dem Wege gehen. Dabei entwirft VEO Systems keine eigenen Formate, sondern übersetzt Standards von z.B. *ISO* oder *ANSI* in XML. Unternehmen habe so die Möglichkeit ihre Dokumentstrukturen sozusagen aus diesen DTD-Bausteinen zusammenzusetzen [MG98, WKL+99].

3.2. Dokument- und Content-Management

Dokument-Management-Systeme, sind Systeme, die es ermöglichen etwas über Dokumente zu erfahren, ohne sich diese selbst ansehen zu müssen [Fre98]. In einem einfachen Fall könnte man sich dies so verstellen, daß die Benutzer Daten zu ihren Dokumenten angeben, diese dann gespeichert werden und später dann nach diesen Daten gesucht werden kann. Solche Daten sind z.B. der Autor, die Hauptthemen und Beziehungen zu anderen Dokumenten. In einem solchen System kann ein Benutzer nun wesentlich einfacher Dokumente finden, die ihn interessieren, als wenn er das File-System durchsuchen müßte, in der Hoffnung, daß ein Dateiname auch etwas mit dem Inhalt der Datei zu tun hat.
Dies alles wurde schon lange realisiert, bevor man auch nur an XML dachte. Wie kommt nun XML zum Einsatz? XML, wie auch SGML, stellt eine Basis zur Verfügung, mit der es möglich ist, nicht nur die Dokumente selbst zu verwalten, sondern jede einzelne Komponente, aus denen das Dokument besteht. Komponenten müssen nicht unbedingt nur Text sein. Vielmehr können sie auch Graphiken, Audio-Daten oder andere Multimedia-Daten sein.
Dies führt zum sogenannten *Content-Management* [Int98]. Content-Management sollte man sowohl vom Dokument-Management abgrenzen, das sich überwiegend mit dem Speichern und Warten von Dokumenten beschäftigt als auch vom sogenannten *Knowledge-Management*, das hauptsächlich dazu dient, Informationen zur Entscheidungsfindung bereitzustellen. Content- Management stellt vielmehr aus beiden Bereichen Funktionen zur Verfügung. So kann man sowohl seine Dokumente speichern, als auch Informationen darüber bereitstellen.
Da Komponenten, wie oben schon beschrieben wurde, sehr vielseitig sein können, möchte man natürlich für jede Art von Komponenten das bestmögliche Werkzeug zu seiner Erstellung benutzten. Das Problem ist, daß jedes Werkzeug, sprich Programm, sein eigenes Format verwendet und die Wiederverwendung dadurch erschwert wird, daß die kleinste Einheit, die ein Programm liefert, die Datei ist. Das Textverarbeitungssystem MS Word© hat zur Wiederverwendung nur die vollständige *.doc-Datei. Will man aber nicht das gesamte Dokument weiternutzen, sondern z.B. nur einen Absatz, so mußte man die ganze Zeit die Copy-and-Paste-Methode anwenden.
Hier kommt nun XML ins Spiel. Durch die Trennung der physikalischen und der logischen Struktur eines Dokuments sind kleinere Einheiten möglich. Zudem ist das Markup von XML selbsterklärend, was wiederum die Informationsfindung wesentlich erleichtert. Durch das verknüpfen von externen Entitäten (vgl. Abschnitt 2.1.2.) wird zudem die Wartung mehr als erleichtert. Wird eine Komponente verändert die in mehreren Dokumenten eingefügt wurde,

so muß nur diese eine Komponente geändert werden und nicht, wie bei der Copy-and-Paste-Methode, jedes Dokument, in das sie eingefügt wurde.
Hier sollte auch noch einmal angemerkt werden, daß natürlich viele der Konzepte von XML nicht neu sind, sondern in SGML schon seit langer Zeit verankert sind. XML ist eben nur sehr viel leichter zu verstehen, was die Akzeptanz und Verbreitung wesentlich ausmacht.

3.3. Informationssuche im Web

Das Informationsangebot im Web scheint heute kein Ende zu finden. Es gibt (fast) nichts, worüber man nichts im Web finden könnte. Der Wert von vielen dieser einzelnen Quellen könnte aber noch erheblich gesteigert werden, wenn aus den Informationen, aus diesen einzelnen Quellen, neue Angebote kombiniert werden könnten. Zwar ist dies mit *HTML-Frames* in gewisser Weise möglich, doch man möchte eben nicht nur ein Nebeneinander, sondern ein Vermischen von Informationen. Dies nennt man *"zusammengesetzte Informationsdienste"* oder *Mehrwertdienste*. Eine Voraussetzung dafür ist, daß die Rohdaten nicht weiter in HTML-Dokumenten "versteckt" bleiben, sondern strukturierter zur Verfügung gestellt werden. Auf diese Weise kann man die Daten auch durch Datenbanktechniken leichter unterstützen. Hier bietet XML, mit seinen, in den vorangegangenen Abschnitten vorgestellten Möglichkeiten, eine hervorragende Basis. Das folgende kurze Beispiel soll dies anhand zweier unterschiedlicher Darstellungsmöglichkeiten eines Verkaufsangebots verdeutlichen.

Beispiel 3.1. [MG98]

Darstellung in HTML:

```
<TITLE>Laptop Computer</TITLE>
<BODY>
      <UL>
            <LI>IBM Thinkpad 560X
            <LI>233 MHz
            <LI>32 MB
            <LI>4 GB
            <LI>4.1 Pfund
            <LI>3200 $
      </UL>
</BODY>
```

Darstellung in XML:

```
<Computer Klasse="Portabel">
      <Hersteller>IBM</Hersteller>
      <Familie>Laptop</Familie>
      <Linie>Thinkpad</Linie>
      <Modell>560X</Modell>
      <Geschwindigkeit Einheit="MHz">
            233</Geschwindigkeit>
      <Festplatte Einheit="GB">
            4</Festplatte>
      <Gewicht Einheit="Pfund">
            4.1</Gewicht>
      <Preis Waehrung="US$">
            3200</Preis>
</Computer>
```

Da in XML die einzelnen Elemente eine Bedeutung haben und nicht nur zur Darstellung dienen, kann in XML-Dokumenten viel besser nach bestimmten Eigenschaften gesucht werden. So könnte z.B. eine Suchanfrage lauten: Finde alle Anbieter, die den Thinkpad 560X von IBM unter 7000 DM anbieten.
Dies war nur ein sehr einfaches Beispiel. Wesentlich komplizierter wird es, wenn die gewünschte Information nicht aus einer einzigen Quelle geholt werden kann, sondern über mehreren Quellen verteilt ist. Ein Problem bleibt aber auch hier gleich: Die Dokumente sind höchstwahrscheinlich in HTML geschrieben.
Da allerdings nicht davon ausgegangen werden kann, daß in kürzester Zeit nun alle Dokumente in Web auf XML umgestellt werden, muß man andere Wege finden um zusammengesetzte Informationsprodukte zu erhalten. Dies geschieht in drei Schritten: vorhandene Quellen erschließen, diese anreichern und dann individualisiert darstellen. Hierzu ein kurzes Beispiel:

Beispiel 3.2.
Gibt es einen Golfplatz, auf dem heute kein Regen zu erwarten ist und den man von Darmstadt aus in weniger als zwei Stunden erreichen kann?

Um diese Frage zu beantworten, müßte man einen Golfführer, einen Reiseplaner und den Wetterdienst zu Hilfe ziehen. Die Frage kann (wohl) kein Webserver direkt beantworten. Ein sogenannter *XML-Broker* müßte aus den einzelnen vorhandenen Informationen ein einziges Angebot generieren und zur Verfügung stellen. Allerdings liegen diese Informationen noch nicht in XML vor. Um Informationen aus verschiedenartigen Quellen in XML zu überführen kann man z.B. *JEDI (Java based Extraction and Dissemination of Information)* [HFA+98] verwenden.
JEDI ist ein Werkzeug um Informationen aus verschieden unabhängigen Quellen zu extrahieren, zu kombinieren und in Einklang zu bringen. Es stellt einen Parser zur Verfügung, um die Daten zu extrahieren und bietet weiterhin ein erweiterbares Objektmodell, das es erlaubt diese Daten einheitlich darzustellen. Als Format zum einheitlichen darstellen wird XML verwendet. Liegen nun die einzelnen Informationen im XML-Format vor, können sie anhand einer Anfragesprache kombiniert werden. Dadurch könnten dann auch solch "komplexe" Anfragen wie die aus Beispiel 3.2. realisiert werde. Momentan wird auch an einer *XML-Anfragesprache* wie z.B. *XQL (XML Query Language)* oder *XML-QL*, gearbeitet.
Auf ein explizites Beispiel wird hier verzichtet, da es den Umfang dieses Dokumentes überschreiten würde. Ein gutes Beispiel findet man in [HFA+98, MFH99].
Es bleibt aber auch hier wieder zu sagen, daß XML zwar eine gute Syntax bereitstellt, um aber bessere Ergebnisse zu erzielen sollte auch der Aufbau (die DTD), zumindest in Teilen, standardisiert werden.

3.4. Web-Automation

Web-Automation ist ein weiteres Schlagwort, daß mit XML in Verbindung steht. Doch was bedeutet Web-Automation eigentlich? Momentan ist es außer Web-Browsern keiner anderen Geschäftsanwendung möglich, direkt auf Webdaten zuzugreifen. Braucht man also bestimmte Daten aus dem Web, muß man momentan seinen Browser aufrufen, auf die benötigte Web-Seite wechseln, eventuell noch einige Formulare ausfüllen und dann die berühmte (besser wohl die berüchtigte) Copy-and-Paste-Methode anwenden und um diese Daten in das firmeneigene Programm zu bekommen. Die momentane Web-Technologie ist also reich an Interaktion aber sehr arm an Automation.
Als eine Anwendung von XML erlaubt *WIDL (Web Interface Definition Language)* Web-Ressourcen als funktionale Schnittstellen zu beschreiben, auf die über Standard-Web-Protokolle zugegriffen werden kann. WIDL versucht also die Interaktion zwischen zwei Systemen herzustellen, ohne das menschliche Unterstützung erforderlich ist. Geschäftsanwendungen sollen somit in das Web integriert werden.
Auch hier kann ohne eine genaue Einführung in die Syntax von WIDL kein Beispiel gegeben werden. Siehe dazu [All97].

4. Zusammenfassung

In den ersten Abschnitten dieses Dokuments wurden Techniken der Metasprache *XML (Extensible Markup Language)* erklärt, eine Teilmenge von *SGML (Standard Generalized Markup Language)*. Dabei wurde sowohl auf die logische Dokumentstruktur, als auch die physikalische Dokumentstruktur eingegangen. Des weiteren wurde aufgezeigt, wie man ein XML-Dokument anhand seiner *DTD (Dokumenttypdefinition)* strukturell validieren kann und mit Hilfe von *XLink* (*Extensible Linking Language)* Verweise einfügt. Mit *XSL (Extensible Stylesheet Language)* wurde ein Mittel zur Präsentation von XML-Dokumenten erläutert, wobei insbesondere auf *XSLT (XSL Transformation Language)* eingegangen wurde. In diesem Zusammenhang wurde auch die Funktionsweise von *XPath (XML Path Language)* und *XPointer (XML Pointer Language)* beschrieben. Als letzte Technik wurde die Beschreibung von Metadaten mit Hilfe von *RDF (Resource Description Framework)* vorgestellt.
In den späteren Abschnitten wurden verschiedene Einsatzgebiete von XML diskutiert. Dazu gehören Dokument- und Content-Management, Web-Automation, Informationssuche im Internet und der Einsatz von XML im E-Commerce.

Literatur

[All97] Allen, C.: WIDL: *Application Integration with XML*, World Wide Web Journal, XML: Principles, Tools, and Techniques, Vol. 2, Issue 4, pp. 229-248, Herbst 1997.

[BPS98] Bray, T., Paoli, J., Sperberg-McQueen, C.M.: *Extensible Markup Language (XML) 1.0, W3C Recommendation 10-February-1998*, W3C, http://w3.org/TR/ REC-xml, Februar 1998.

[CD99] Clark, J., De Rose, S.: *XML Path Language (XPath), W3C Working Draft, 9-July-1999*, W3C, http://www.w3.org/TR/xpath, Juli 1999

[Cha99] Chawathe, S. S.: *Describing and Manipulating XML Data*, Data Engineering, Vol. 22, No. 3, pp. 3-9, IEEE Computer Society, September 1999.

[DD99] De Rose, S., Daniel Jr., R.: *XML Pointer Language (XPointer), W3C Working Draft 9-July-1999*, W3C, http://www.w3.org/TR/xlink, Juli 1999.

[Dea99] Deach, S.: *Extensible Stylesheet Language (XSL) Spezifikation, W3C Working Draft 21-April-1999*, W3C, http://www.w3.org/TR/WD-xsl/, April 1999.

[DOT00] De Rose, S. Orchard, D. Trafford, B.: *XML Linking Language (XLink), W3C Working Draft 19-January-2000*, W3C, http://www.w3.org/TR/WD-xptr, Januar 2000.

[Fre98] Freter, T.: XML: *Document and Information Management*, Technology & Research, Sun Microsystems, http://www.sun.com/980908/xml/, September 1998.

[GP98] Goldfarb C.F., Prescod, P.: *The XML-Handbook*, Prentice Hall, 1998.

[HFA+98] Huck, G., Fankhauser, P., Aberer K., Neuhold, E. J.: *Jedi: Extracting and Synthesizing Information from the Web*, Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS'98), pp. 32-43, New York City, August 1998.

[Int98] Interleaf: *Putting XML to Work: Business-to-Business Advantages Through Enterprise Content Management*, White Paper, Interleaf, Inc., July 1998.

[MFH99] Macherius, I., Fankhauser, P., Huck, G.: *Zwischenhändler: Verknüpfung von Webdaten mit XML*, iX - Magazin für professionelle Informationstechnik, Volume 4/99, pp. 90-95, April 1999.

[MG98] Meltzer, B., Glushko, R.: *XML and Electronic Commerce: Enabling the Network Economy*, SIGMOD Record, Vol. 27, No. 4, pp. 21-24, December 1998.

[WKL+99] Weitzel, T., Kronenberg, R., Ladner, F., Buxman, P.: *Die Rückkehr der EDI-Ritter: XML als Alternative zu traditionellem EDI*, iX-Magazin für professionelle Informationstechnik, Volume 7/99, pp. 127-129, July 1999.