Einführung in XML

Seminar: XML in der Bioinformatik
Frank Schönmann

WS 2002/03

1 Einführung

In der Praxis dient das World Wide Web (WWW) inzwischen für eine Vielzahl von Informationssystemen als Oberfläche. Darunter fallen zum Beispiel Datenbanken mit HTML-Frontend, Warenkataloge, Mail-Archive oder Handbücher.

Diese Dokumente besitzen aber eine reiche innere Struktur, die nicht in HTML ausgedrückt werden kann. Sie geht bei der Darstellung im WWW verloren: es findet ein Informationsverlust statt. So verschwindet bei einer über das Web abfragbaren relationalen Datenbank die auf dem Server vorhandene Strukturierung der Nutzdaten in Felder.

Eine Nutzung der in HTML dargestellten Daten ist anschließend nicht mehr möglich. Lediglich die nun unstrukturierten Einzel-Informationen lassen sich noch durch *Cut & Paste* weiterverwenden.

Dieses Problem verhindert im Internet die Nutzung des WWW als Plattform für einen Informationsaustausch im großen Rahmen. Benötigt würde eine Lösung, die es erlaubt, Daten aus dem Netz ohne Informationsverlust weiterzuverarbeiten, z. B. durch einfaches *Drag & Drop* in lokale Anwendungen.

Aus diesem Grund schuf das W3C[1] die Metasprache XML. XML steht als Abkürzung für *eXtensible Markup Language*, also eine erweiterbare Auszeichnungssprache.

Eine Auszeichnungssprache beschreibt einzelne Teile eines Dokuments als dessen logische Bestandteile. In HTML (*HyperText Markup Language*), einer Auszeichnungssprache, die auf die Strukturierung von Texten abzielt, kann man so beispielsweise Überschriften, Absätze oder Tabellen logisch auszeichnen. MathML (*MATHematic Markup Language*) unterscheidet dagegen Dokumentteile wie Operatoren oder Variablen.

Erweiterbar ist XML deshalb, weil sich damit eigene Auszeichnungssprachen definieren lassen.

XML betrachtet sich selbst als ein Universalkonzept zur Datenspeicherung, da sich damit auf einfache Weise selbst komplexe Daten relativ gut modellieren lassen. Dabei wird auf eine strikte Trennung von Inhalt, Struktur und Layout geachtet: dadurch ist es z. B. möglich, vorhandene XML-Dokumente auf unterschiedliche Arten zu präsentieren, ohne die zugrunde liegenden Daten zu verändern. Wir betrachten an dieser Stelle daher auch nur die Speicherung von Struktur und Inhalt.

2 Entwicklung von XML

2.1 Geschichtliche Entwicklung

1994 Tim Berners Lee, bekannt geworden als „Erfinder“ des World Wide Web durch die Entwicklung von HTML, gründete am MIT (*Massachusetts Institute of Technology*) ein offenes Entwicklerforum, zu dem Vertreter aus Industrie, Forschung, Politik und Medien Zugang hatten.

Ziel war es, eine Definitionssprache für Auszeichnungssprachen zu entwickeln. Es wurde klar, dass HTML den künftigen Anforderungen alleine nicht genügen kann; die Gründe

[1] Das *World Wide Web Consortium* wurde im Oktober 1994 ins Leben gerufen, um das World Wide Web durch die Schaffung von Standardprotokollen und -formaten in seiner Entwicklung zu unterstützen. Das W3C besteht momentan aus ca. 450 Mitglieder, darunter so bekannte Unternehmen wie Microsoft, AOL oder Adobe (`http://www.w3.org/Consortium/`).

dafür sind bereits weiter oben dargelegt. Es sollte aber ein Standard gefunden werden, in den HTML integrierbar war und der den Erfolg von HTML ausnutzen konnte.

Zunächst waren Konzepte im Gespräch, die SGML außer Acht ließen. Doch schließlich setzte sich eine Lösung durch, in der die neue Definitionssprache als Teilmenge von SGML konzipiert und mit Hilfe von SGML beschrieben wurde.

1996 Auf der *SGML 96 Conference* wurde ein erster Entwurf für XML vorgestellt.

1998 Im Februar brachte das W3-Konsortium die erste Empfehlung[2] (*Recommendation*) zu XML heraus. Danach bemühte sich die Organisation, XML zu etablieren und wichtige, bereits bestehende Auszeichnungssprachen wie HTML in XML neu zu definieren.

2.2 Zur Rolle von SGML

SGML (*Standard Generalized Markup Language*) ist eine „Sprache zur Dokumentenrepräsentation, welche Markup formalisiert und von System- und Verarbeitungsabhängigkeiten löst". So beschreibt es der entsprechende ISO-Standard (ISO-8879).

Es erlaubt den Austausch von komplexen Datenmengen und vereinfacht den Zugriff auf sie. Neben den Möglichkeiten einer beschreibenden Auszeichnung von Dokumenten, erlaubt SGML auch die Überprüfung der Gültigkeit eines Textelements in einem bestimmten Kontext.

XML selbst ist eine Teilmenge von SGML. Dadurch ist es weniger umfangreich, aber in der Praxis ähnlich leistungsfähig. Um dies zu erreichen, wurden vor allem komplexe und selten verwendete Eigenschaften von SGML entfernt. Die abschreckende Syntax von SGML wurde stark vereinfacht.

Zum Vergleich der Komplexität der beiden Sprachen nur folgendes: XML wurde auf etwa 33 Seiten formal definiert; für SGML benötigte man dagegen mehr als 500 Seiten.

2.3 Bekannte XML-Anwendungen

In den folgenden Abschnitten werden einige schone bestehende XML-Sprachen vorgestellt. Die eingefügten Abbildungen zeigen bereits beispielhaft die typische Struktur von XML-Dokumenten.

2.3.1 XHTML

Wie bereits erwähnt entwickelte das W3C nach Schaffung eines XML-Standards zu dessen Etablierung gleich mehrere Auszeichnungssprachen. Besonders auf die Neudefinition der am weitesten verbreiteten, HTML, wurde dabei großen Wert gelegt. Im Januar 2000 veröffentlichte man daher eine Empfehlung für XHTML 1.0, eine Reformulierung von HTML 4.0 mit Hilfe von XML.

Abgesehen von der etwas strengeren Syntax (HTML war vorher in SGML definiert), hat HTML durch diese Neuformulierung nichts von seiner Mächtigkeit verloren. Abbildung 1 zeigt ein Beispiel-XHTML-Dokument.

[2] Diese „Empfehlung" heißt zwar so, ist in Wirklichkeit aber die höchste Ebene, die eine Veröffentlichung des W3C erreichen kann, und steht damit auf gleicher Stufe wie ein Standard.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html><head>
<title>Einfache (X)HTML-Seite</title>
</head><body>
<h1>Überschrift</h1>
<img alt="Bild" src="bild.png" />
<hr />
</body></html>
```

Abb. 1: Ein vollständiges XHTML-1.0-Dokument. Zu sehen sind hier die XML-Deklaration, die *document type declaration* und der eigentliche Inhalt der XML-Datei.

2.3.2 MathML

MathML ist eine der ersten und auch bekanntesten Markup-Sprachen im eher wissenschaftlichen Bereich. Bereits 1998, also noch vor XHTML, war die Entwicklung von MathML 1.0 abgeschlossen, im Jahr 2001 folgte eine W3C-Empfehlung für MathML 2.0. Abbildung 2 zeigt, wie ein typischer Ausschnitt eines MathML-Dokuments aussieht.

```
<msup>
    <mfenced>
        <mi>a</mi> <mo>+</mo> <mi>b</mi>
    </mfenced>
    <mn>2</mn>
</msup>
```

Abb. 2: Ausschnitt aus einem MathML-Dokument: die hier dargestellte Formel entspricht dem mathematischen Ausdruck $(a + b)^2$.

2.3.3 Chemische und biologische Auszeichnungssprachen

Auch für andere Naturwissenschaften wurden bereits XML-Formate geschaffen. Beispielhaft erwähnt werden sollen hier nur die CML[3](*Chemical Markup Language*), BioML[4] (*BIOpolymer Markup Language*) zur Speicherung von Bio-Polymeren und die BSML[5] (*Bioinformatic Sequence Markup Language*), die zur Speicherung sämtlicher Arten von Sequenzdaten (z. B. DNA, RNA, Proteine) dient und in Struktur und Informationsgehalt Ähnlichkeiten aufweist zu Formaten der bekannten Sequenzdatenbanken EMBL, GenBank bzw. DDBJ.

[3] `http://www.xml-cml.org/`
[4] `http://www.bioml.com/BIOML/index.html`
[5] `http://www.bsml.org/`

3 Regeln und Eigenschaften von XML-Dokumenten

Kommen wir nun zum wichtigen Abschnitt über den korrekten Aufbau von XML-Dateien. Wie bereits in der Einführung erwähnt, trennt XML zwischen Struktur, Inhalt und Layout: Struktur und Inhalt sind dabei Teile der **Instanz**, aus der jedes XML-Dokument besteht.

Daneben ist es möglich, aber nicht verpflichtend, zusätzlich eine *document type definition* (DTD) anzugeben: dabei handelt es sich um eine Definition, in der die Eigenschaften von Tags und deren erlaubte Schachtelung vereinbart werden. Abschnitt 4 beschreibt Näheres zur Erstellung einer solchen DTD.

3.1 Allgemeine Eigenschaften

Jede XML-Datei beginnt mit der sogenannten **XML-Deklaration**. Dabei handelt es sich um eine *processing instruction* (Verarbeitungsanweisung), die dazu dient, XML- von SGML-Instanzen zu unterscheiden. Da XML eine Teilmenge von SGML ist, wäre sonst kein Unterschied zu erkennen.

Möchte man für seine XML-Datei eine DTD verwenden, benötigt man außerdem noch eine *document type declaration*. Diese gibt an, an welcher DTD man sich orientiert und wo sie zu finden ist. In Abbildung 3 wird der Anfang eines XML-Dokuments exemplarisch dargestellt.

```
<?xml version="1.0" ?>
<!DOCTYPE dtdname SYSTEM "dtdname.dtd">
```

Abb. 3: Beginn eines XML-Dokuments, das außerdem eine DTD verwendet. Zu sehen sind die XML-Deklaration und die Dokumenttyp-Deklaration.

3.2 Tags und Elemente

Um die Struktur der Daten abzubilden, werden sogenannte **Tags** als benannte Klammern verwendet. Man unterscheidet zwischen einem Starttag und einem Endtag; zwischen beiden steht der Inhalt (siehe Abbildung 4). Durch Tags erfolgt die Auszeichnung des Inhalts als logischer Bestandteil des im XML-Format abgelegten Dokuments. Die Kombination von Starttag, Inhalt und Endtag nennen wir **Element**.

```
<name>Hans Müller</name>
```

Abb. 4: Ein Element, aufgebaut aus einem Starttag (`<name>`), dem Inhalt (Hans Müller) und einem Endtag (`</name>`).

Zu beachten ist, dass im Gegensatz zu SGML (z. B. in HTML) jedes öffnende Tag auch wieder geschlossen werden muss. Dies stellt zwar eine zusätzliche Einschränkung von XML dar, ist aber unerheblich für die Modellierung von Daten, da entweder Teile des Dokuments wirklich mit einem bestimmten Tag ausgezeichnet werden sollen oder ein Element ohne Inhalt

durch leere Tags (`<tag></tag>`) nachgebildet werden kann. Für diese leeren Elemente wurde daher eine alternative Syntax geschaffen: `<tag/>` oder `<tag />`[6].

Bei der Wahl eines Tagnamens ist zu beachten, dass er nicht mit der Zeichenfolge `xml` beginnt – diese ist reserviert für kommende Versionen von XML – weder =-Zeichen noch Leerzeichen enthält und keine Ziffern am Anfang stehen.

Schachtelt man Elemente, so ergibt sich (v. a. bei daten-zentrierten XML-Formaten) eine Baumstruktur, die den Aufbau der Daten widerspiegelt. Dabei drücken innere Knoten des Baums diesen Aufbau aus, die Blätter stehen für den Inhalt von Elementen bzw. leere Elemente. Die beiden Abbildungen 5 und 6 zeigen, wie der zugehörige Baum zum Teil eines XML-Dokuments aussieht.

```
<person>
    <name>Hans Müller</name>
    <email>hans@mueller.de</email>
    <birthday> <day>21</day><month>03</month><year>1970</year> </birthday>
</person>
```

Abb. 5: Mehrfach verschachtelte XML-Tags. Die Struktur des XML-Dokuments steckt in den Tags, der Inhalt dazwischen.

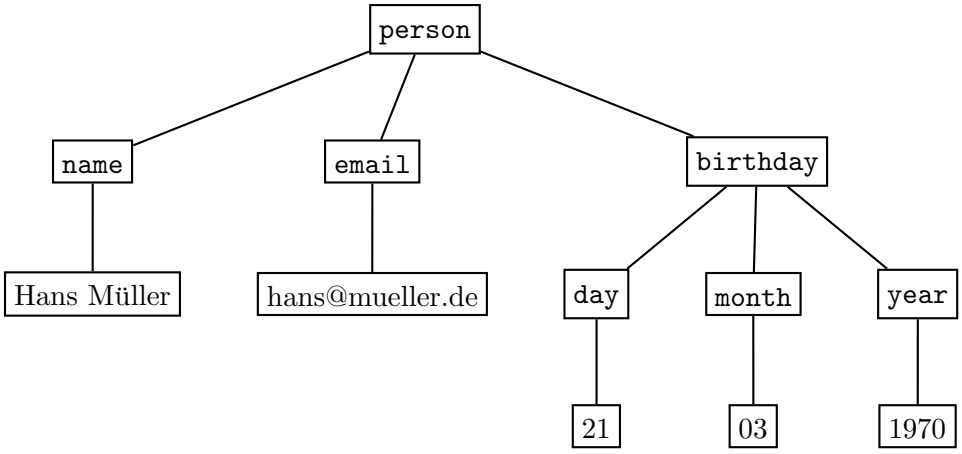

Abb. 6: Der Baum zum Ausschnitt in Abbildung 5. Zur Unterscheidung von Struktur und Inhalt des XML-Dokuments sind diese in verschiedenen Schrifttypen dargestellt.

3.3 Kommentare

Um z. B. seinen Daten eine versteckte Beschreibung mitzugeben oder sonstige Informationen anzumerken, kann man in XML Kommentare einbetten, die beliebigen Text enthalten dürfen. Sie beginnen mit der Zeichenfolge `<!--` und enden mit `-->`.

[6] Das Leerzeichen ist hier laut XML nicht notwendig, aber im Hinblick auf z. B. ältere Browser, die XHTML als (altes) HTML interpretieren sollen, kann das zusätzliche Leerzeichen dazu dienen, dass die Tagnamen korrekt erkannt werden.

Wichtig: Kommentare dürfen nicht ineinander geschachtelt werden und die Zeichenkette `--` sollte darin nicht vorkommen. Zwei kleine Beispiele sind in Abbildung 7 zu sehen.

```
<!-- Kommentar -->

<!-- mehrzeiliger
     Kommentar... -->
```

Abb. 7: Ein einzeiliger und ein mehrzeiliger Kommentar.

3.4 Attribute

Tags können mit Attributen versehen werden, durch die sich die Funktion des Tags in der Dokumentinstanz weiter spezialisieren lässt. Die verwendeten Attribute sind Name-Wert-Paare, wobei die Werte – im Gegensatz zu SGML – in (einfachen oder doppelten) Anführungszeichen stehen müssen. In Abbildung 8 wurde das bisherige Beispiel erweitert um einige Attribute.

```
<?xml version="1.0"?>
<!DOCTYPE contacts SYSTEM "contacts.dtd">

<contacts type="private" desc="Persönliches Addressbuch">
    <person id="42">
        <name>Hans Müller</name>
        <email>hans@mueller.de</email>
        <birthday>
            <day>21</day><month>03</month><year>1970</year>
        </birthday>
    </person>
</contacts>
```

Abb. 8: Das in Abbildung 5 dargestellte Beispiel wurde hier mit einigen Attributen versehen und in ein vollständiges XML-Dokument umgewandelt.

3.5 Zeichensätze

In XML wird zwischen Groß- und Kleinschreibung unterschieden: das Tag `<xml>` kann also nicht synonym verwendet werden mit `<Xml>` oder `<XML>`. Man sollte aber darauf verzichten, Namen durch unterschiedliche Großschreibung mehrfach zu belegen, damit auch menschliche Betrachter die Übersicht behalten.

Auf die Internationalisierung ist XML bereits gut vorbereitet: erlaubt ist die Verwendung aller Zeichen des ISO-10646-Standards. Unicode[7] ist die offizielle Implementierung dieses

[7] `http://www.unicode.org/`

Standards. Damit ist XML auch geeignet für exotische Zeichen aus asiatischen oder altindianischen Sprachen. Vorschläge für zukünftige Versionen von XML sehen vor, auch den Zeichenraum für Tagnamen auf den gesamten Unicode-Zeichensatz auszudehnen.

Lediglich einige Sonderzeichen, die in XML eine spezielle Bedeutung haben, müssen im normalen Text maskiert werden (siehe Tabelle 1).

<	<	>	>	&	&
"	"	'	'		

Tab. 1: XML-Sonderzeichen und ihre Maskierung im Text.

3.6 Bezeichnungen

Zum Abschluss dieses Abschnitts sollen noch einige Bezeichnungen eingeführt werden, die im Zusammenhang mit XML auftauchen.

Ein XML-Dokument heißt **wohlgeformt** (*well formed*), wenn alle Regeln für XML eingehalten werden. Zu beachten ist:

- Das Dokument muss mit einer XML-Deklaration beginnen.
- Es muss genau ein äußerstes Element existieren; insbesondere darf das XML-Dokument also nicht aus mehr als einem äußeren Datenelement bestehen.

Darüber hinaus müssen natürlich weiterhin die bisher genannten Regeln, z. B. zur Verschachtelung von Elementen beachtet werden.

Ein XML-Dokument heißt **valid** (gültig), wenn es wohlgeformt ist und zusätzlich den Regeln einer DTD genügt. Die Unterscheidung zwischen valid und wohlgeformt, was es bei SGML noch nicht gab, wurde eingeführt, da XML-Dokumente nicht zwingend auf einer DTD basieren müssen.

4 Document Type Definition (DTD)

DTD steht als Abkürzung für *document type definition* (im Gegensatz zur *document type declaration*, die am Beginn eines XML-Dokuments steht und auf die Anwendung einer DTD hinweist). Definiert werden die Namen der in Dokumentinstanzen erlaubten Attribute, Entitäten sowie Tags und ihre mögliche Schachtelung, das sogenannte **Inhaltsmodell**.

Man kann eine DTD als formale Grammatik betrachten, die eine bestimmte XML-Sprache definiert. Wie bei einer Grammatik üblich, wird nur die erlaubte Syntax angegeben und keine semantische Aussage gemacht.

Die Regeln der Grammatik werden im Folgenden beschrieben. Als Startsymbol der Grammatik dient ein besonderes Tag, das auch als äußerstes Element in der XML-Datei steht: der **Dokumenttyp**. Dieser trägt den selben Namen wie die definierte XML-Sprache.

4.1 Inhaltsmodell

Das Inhaltsmodell eines Tags gibt an, woraus dessen Inhalt bestehen darf. Dies entspricht etwa den Ableitungsregeln einer formalen Grammatik: ein Nichtterminalsymbol wird allerdings

`,`	notwendige Elemente in fester Reihenfolge
`|`	alternative Elemente
`?`	optionale Elemente
`+`	1 . . . n Wiederholungen
`*`	0 . . . n Wiederholungen
`()`	Klammerung von Ausdrücken

Tab. 2: Diese Operatoren sind in einer DTD zur Definition des Inhaltsmodells eines Elements erlaubt. Damit lassen sich die in der Aufzählung genannten Bestandteile zu komplexeren Ausdrücken verknüpfen.

nicht ersetzt, sondern anhand der Regeln mit Inhalt gefüllt und bleibt selbst bestehen (man kann also die Ableitungskette später nachvollziehen).

Aus folgenden Bestandteilen wird das Inhaltsmodell zusammengesetzt:

- andere Tags: erlaubt man als Inhalt eines Elements weitere Tags, entsteht erst die bereits erwähnte Baumstruktur der Daten.
- `#PCDATA` (*parseable character data*): an dieser Stelle ist beliebiger Text aus den gültigen Zeichen des verwendeten Alphabets erlaubt.
- `EMPTY`: das Inhaltsmodell ist leer, also darf dieses Tag nur als leeres Element im XML-Dokument vorkommen.
- `ANY`: ein beliebiges Inhaltsmodell, d. h. sämtliche andere Elemente sowieo normaler Text können beliebig vermischt an dieser Stelle als Inhalt dienen.

Diese einzelnen Elemente können weiter verknüpft werden durch die in Tabelle 2 genannten Operatoren. Abbildung 9 zeigt in einem einfachen Beispiel einige Möglichkeiten.

```
<!ELEMENT contacts (person*)>

<!ELEMENT person (name, email*, birthday?)>

<!ELEMENT name  (#PCDATA)>
<!ELEMENT email (#PCDATA)>

<!ELEMENT birthday (year, month, day)>

<!ELEMENT year  (#PCDATA)>
<!ELEMENT month (#PCDATA)>
<!ELEMENT day   (#PCDATA)>
```

Abb. 9: Eine einfache DTD für das Beispiel der Kontaktdatenbank: `<contacts>` darf beliebig viele `<person>`-Elemente enthalten. `<person>` muss aus einem `<name>`-Element bestehen, kann dann beliebig viele `<email>`-Tags enthalten und möglicherweise ein `<birthday>`-Tag, etc.

4.2 Attribute

XML-Tags dürfen Attribute enthalten; auch diese werden in der DTD definiert. Jedem Attribut wird dazu ein Typ zugeordnet. Gültige Typen sind: `CDATA` (*character data*) für beliebigen Text, `ID` für eine dokumentenweit eindeutige Identifikationszeichenkette, `IDREF` bzw. `IDREFS` für ein Attribut, das auf eine oder mehrere vorhandene IDs verweist, sowie eine Liste von erlaubten Werten[8].

Attribute sind entweder optional, obligatorisch oder konstant. Dies wird durch jeweils eine der folgenden Alternativen gekennzeichnet: `#IMPLIED` markiert ein optionales, `#REQUIRED` ein obligatorisches Attribut; mit `#FIXED` gibt man einen konstanten Wert an. Außerdem lässt sich ein Standardwert angeben, der verwendet wird, falls das Attribut im XML-Dokument fehlt.

Abbildung 10 erweitert das letzte DTD-Beispiel um einige Attribute.

```
<!ELEMENT contacts  (person*)>
<!ATTLIST contacts
    type    (private|work)  #IMPLIED
    desc    CDATA           #IMPLIED
>

<!ELEMENT person    (name, email*, birthday?)>
<!ATTLIST person
    id      ID              #REQUIRED>

<!ELEMENT name      (#PCDATA)>
<!ELEMENT email     (#PCDATA)>

<!ELEMENT birthday  (year, month, day)>

<!ELEMENT year      (#PCDATA)>
<!ELEMENT month     (#PCDATA)>
<!ELEMENT day       (#PCDATA)>
```

Abb. 10: In dieser DTD wurden einige erlaubte Attribute definiert: das `<contacts>`-Tag darf eine beliebige Beschreibung haben und lässt sich durch einen Typ als private oder geschäftliche Kontaktliste definieren. Außerdem wird das `<person>`-Tag um ein obligatorisches ID-Attribut ergänzt.

4.3 Entitäten

Mit Entitäten (*entities*) lassen sich Kürzel definieren, die später in der DTD selbst oder in der XML-Instanz verwendet werden können. Dabei betrachten wir v. a. die beiden wichtigsten Typen von Entitäten.

Zum einen kann man Entitäten als Textbausteine verwenden: sie dienen dann im XML-Dokument als Makros. Eingebaut werden sie dort mit der speziellen Syntax `&entityname;`.

[8] Tatsächlich existieren noch weitere mögliche Typen für Attribute (z. B. `NMTOKEN` bzw. `NMTOKENS` und `ENTITY`), die aber seltener benötigt und daher hier nicht näher beschrieben werden.

Beispielsweise die einfache Einfügung von Umlauten und Sonderzeichen in XHTML wurde durch die Verwendung von Entitäten realisiert.

Andererseits existieren sogenannte *parameter entities*. Diese wiederum dienen als Makros in der DTD: mehrfach verwendete Inhaltsmodelle oder Attributlisten, die häufiger vorkommen, kann man zum Beispiel auf diese Weise verwirklichen. Einige kleine Beispiele finden sich in Abbildung 11.

```
<!ENTITY   xml       "Extensible Markup Language">
<!ENTITY   auml      "&#228;">
<!ENTITY % bool      "(true|false)">

<!ELEMENT  person    (name, email*, birthday?)>
<!ATTLIST  person
    id               ID      #REQUIRED
    premiumcustomer  %bool;  "false"
>

<!ELEMENT  birthday  (year, month, day)>
<!ATTLIST  birthday
    sendgreeting     %bool;  #IMPLIED
>
```

Abb. 11: Entitäten wurden hier verwendet, um zwei Textmakros zu definieren: `&xml;` und `&auml;` (kopiert aus der offiziellen XHTML-1.0-DTD) lassen sich nun in der XML-Instanz verwenden. Eine *parameter entity* (zu erkennen am %-Zeichen) wurde verwendet, um einen boolschen Datentyp zu entwerfen, der weiter unten für zwei Attributen als Typ dient.

5 XML in der Bioinformatik

Im Folgenden werden einige Vor- und Nachteile von XML genannt. Anschließend soll diskutiert werden, ob XML für den Einsatz in der Bioinformatik geeignet scheint.

5.1 Vorteile

Standardisierung Ein wichtiger Schritt wurde hier bereits durch die Standardisierung von XML selbst getan. Bisherige Datenformate in der Bioinformatik (und auch anderen Bereichen) sind proprietär. Möglicherweise entsteht also ein Anreiz, XML einzusetzen, um selbst standardisierte Formate zu entwickeln, die von anderen verwendet werden können.

Offenheit Dies ergibt sich bereits relativ zwingend daraus, dass XML ein Textformat ist. Ein Anwender kann jede XML-Datei mit einem einfachen Texteditor verändern oder auslesen. Werden für Elemente auch noch aussagekräftige Namen verwendet, entsteht eine leicht durchschaubare Auszeichnungssprache, die weitläufig genutzt werden kann. Die Syntax wird in einer DTD definiert und ist daher für alle direkt einsehbar und muss nicht erraten werden.

Einfachheit Entscheidend ist hier, dass einfache Änderungen an bestehenden Daten möglich sind, da XML ein lesbares und editierbares Textformat ist (*content scalability*) – man benötigt im Gegensatz zu Binärformaten keine speziellen Programme zur Bearbeitung der Daten.

Ein weiterer Aspekt ist, dass durch die strenge Syntax eine schnelle Validierung möglich ist, das Dokument also einfach auf Gültigkeit überprüft werden kann. Ähnlich einfach ist eine Transformation in ein anderes Zielformat, da hierfür bereits Standards existieren (XSL-FO, XSLT[9]).

Standardisierte Verarbeitungsmodelle Benötigt man einen Parser, so stehen bereits zwei standardisierte Verfahren zur Verfügung, für die es außerdem bereits fertige Programme gibt. SAX[10] (*Simple API for XML*) ist ein ereignisbasierter Parser, d. h. er erhält das XML-Dokument als Datenstrom und löst in bestimmten Situationen (z. B. dem Beginn eines neuen Elements) Ereignisse aus, die durch Funktionen abgefangen werden können. Die Alternative sind DOM-Parser[11] (*Document Object Model*), die das XML-Dokument bereits beim Start parsen und danach Operationen auf dem XML-Baum erlauben.

Internet-Orientierung Wie aus der Motivation bereits hervorgeht, wurde XML speziell für die Verwendung im Internet entwickelt. Darüber hinaus erlaubt XML die Einbettung von Daten an anderen Adressen (über spezielle Entitäten in DTDs). Daneben sind noch reichhaltige Fähigkeiten zur Verlinkung von Daten vorhanden, die auf XML aufsetzen. Erwähnt sei hier nur ohne weitere Erklärungen XLink[12] (*XML Linking Language*).

5.2 Nachteile

Daten-Overhead Durch das textbasierte Format und die Syntax-Regeln benötigt XML einen größeren Speicherbedarf als andere Formate. Beispielsweise erhält man in einer der bekannten Sequenzdatenbanken XML-Dateien als Ausgabeformat, die 25 bis 75 Prozent größer sind als die Daten im EMBL-Format. Neben dem Speicherbedarf schlägt sich das natürlich auch in höheren Übertragungsmengen nieder.

Fehlende Skalierbarkeit Ebenfalls mit dem Textformat hängt dieses Problem zusammen: die Dateien lassen sich nicht für schnellere Zugriffe indizieren, auch *clustering* ist kaum möglich. Folge ist eine schlechte Performance bei der Datenhaltung. Allerdings könnte eine Kombination mit echten Datenbanken hier Abhilfe schaffen, die XML nur noch als Ein- und Ausgabe verwenden, intern aber ein eigenes Format zur Speicherung der Daten nutzen.

Limitierte Modellierungsfähigkeiten XML ist in der Datentypisierung sehr beschränkt: es stehen dafür fast nur Zeichenketten zur Verfügung, aber keine Typen wie Zahlen, Tabellen oder Matrizen. Weiterhin gibt es keinen Mechanismus zur Beschreibung von Beziehungen zwischen Elementen; dafür ist nur eine in diese Richtung gehende Nachbildung mit Hilfe von

[9] XSL-FO (*eXtensible Stylesheet Language Formatting Objects*) dient dazu, XML-Dokumente in gedruckter Form zu repräsentieren, mit XSLT (*eXtensible Stylesheet Language Transformations*) lassen sich XML-Dokumente in eine andere XML-Sprache transformieren. Nähere Informationen hierzu unter `http://www.w3.org/Style/XSL/`

[10] `http://www.saxproject.org/`

[11] `http://www.w3.org/DOM/`

[12] `http://www.w3.org/XML/Linking`

`ID` und `IDREF` möglich. Die Lösung für dieses Problem könnte XML Schema[13] heißen, auf das hier jedoch nicht näher eingegangen wird.

5.3 Einsatz in der Bioinformatik

Für den Einsatz von Datenformaten in der Biologie und Bioinformatik gibt es mehrere Aspekte, die hier kurz stichpunktartig genannt werden sollen:

- Daten benötigen eine komplexe Modellierung, es gibt viele verschiedene Datentypen und durch neue Entdeckungen in der Forschung können sich bestehende Typen häufig leicht ändern.
- Es entstehen oft neue Arten von Daten, für die geeignete Formate entwickelt werden müssen, die sich mit bereits bestehenden kombinieren lassen sollten.
- Bestehende Daten werden häufig aktualisiert und dann von Wissenschaftlern stark übers Internet verbreitet.
- Verschiedene Typen von Benutzern brauchen Zugriff auf die Daten; darunter fallen Biologen, Programmierer und andere. Nicht alle potenziellen Anwender haben Erfahrung mit Datenbanken. Ein möglichst einfacher Umgang mit den Daten ist daher wünschenswert.

Momentan gibt es in der Bioinformatik eine große Anzahl bestehender Datenbanken mit uneinheitlichem Format. Viele Daten sind mehrfach vorhanden und gar nicht oder nur schwach miteinander verknüpft (z. B. durch einfache Hyperlinks, aber nicht durch eine beschreibende Beziehung).

Vergleicht man diese Aspekte mit den oben genannten Vorteilen, so kann man erkennen, dass XML schon viele dieser Punkte abdeckt. Die meisten genannten Nachteile lassen sich umgehen oder durch zusätzliche Techniken lösen. Tabelle 3 vergleicht XML mit zwei anderen Lösungen, die bereits in der Bioinformatik genutzt wurden.

	XML-Format	Feld/Wert	(OO)DBMS
Ausdruckskraft	**	*	****
Modellierungsfähigkeit	**	*	****
Selbstbeschreibbarkeit	ja	nein	ja
Abfragesprache	ja	nein	ja
Flexibilität	****	*	****
Einfachheit	****	****	**
Skalierbarkeit	**	*	****
Kompatibilität	****	*	***

Tab. 3: Vergleich einiger Lösungen zur Datenhaltung.

6 Zusammenfassung

Betrachtet man die oben genannten Nachteile von XML, so lassen sich wohl die meisten der dort erwähnten Aspekte umgehen: Probleme mit der schlechten Performance bei der

[13] `http://www.w3.org/XML/Schema`

Datenverwaltung kann man beseitigen, indem man die Daten weiterhin in einer (relationalen oder objektorientierten) Datenbank speichert und XML nur als Ein- und Ausgabeformat nutzt.

Übrig bleibt noch das Problem der Modellierungsfähigkeit. Dieses Problem lässt sich aber möglicherweise durch XML Schema lösen.

Natürlich ist es danach noch nötig, einheitliche XML-Formate für biologische Zwecke zu entwickeln, die eine Art Standard in diesem Bereich bilden. Falls es darüber zu einer Einigung käme, könnte XML bisherige Textformate, die in einiger Hinsicht unterlegen sind (siehe Tabelle 3) ablösen.

7 Quellen

- Achard, Vaysseix, Barillot: XML, bioinformatics and data integration, *Bioinformatics Review Vol 17 no. 2 2001*
- `http://www.w3.org/XML/`
- Macherius: Revolution der Experten, *iX 6/1997, S. 106*
  `http://www.heise.de/ix/artikel/1997/06/106/`
- `http://selfhtml.teamone.de/xml/`