

Prof. Dr.-Ing. Matthias L. Hemmje

01877

Dokumenten- und Wissensmanagement im Internet

Kurseinheit 5:

Semantik und Semantisches Web

Fakultät für
Mathematik und
Informatik

Der Inhalt dieses Dokumentes darf ohne vorherige schriftliche Erlaubnis durch die FernUniversität in Hagen nicht (ganz oder teilweise) reproduziert, benutzt oder veröffentlicht werden. Das Copyright gilt für alle Formen der Speicherung und Reproduktion, in denen die vorliegenden Informationen eingeflossen sind, einschließlich und zwar ohne Begrenzung Magnetspeicher, Computerausdrucke und visuelle Anzeigen. Alle in diesem Dokument genannten Gebrauchsnamen, Handelsnamen und Warenbezeichnungen sind zumeist eingetragene Warenzeichen und urheberrechtlich geschützt. Warenzeichen, Patente oder Copyrights gelten gleich ohne ausdrückliche Nennung. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Inhaltsverzeichnis

| | |
|--|-----|
| Inhaltsverzeichnis | III |
| 1. Einleitung | 4 |
| 2. Motivation und Einführung | 4 |
| 3. Einführung zu Semantik und RDF | 8 |
| 4. RDF Schemata | 23 |
| 5. Erzeugung von Ontologien mittels OWL..... | 35 |
| 6. Metadaten | 47 |
| 7. Anhang | 55 |
| 8. Literaturverzeichnis | 57 |
| 9. Abbildungsverzeichnis | 59 |
| 10. Lösungen der Selbsttestaufgaben | 60 |

1. Einleitung

Dies ist die fünfte Kurseinheit des Kurses 1877 „Dokumenten- und Wissensmanagement im Internet“.

Die Teilnehmer des Kurses setzen sich hier mit dem semantischen Web als Fortentwicklung des bestehenden WWWs auseinander. Betrachtet werden die Bereitstellung von Informationsquellen im Hinblick auf menschliche und auch maschinenlesbare Nutzung sowie Probleme und Lösungsansätze bei der Etablierung eines semantischen Netzes.

Die Studierenden analysieren das Ressource Description Framework (RDF) als ein Modell der Wissensrepräsentation und der Darstellung von Metadaten sowie, darauf aufbauend, RDF-Schema (RDFS-S), das Vokabular, welches in RDF-Datenmodellen Verwendung findet, und spezifiziert, welche Merkmale zu welchen Objekten gehören.

Im Folgenden erarbeiten sich die Kursteilnehmenden die Grundlagen der Web Ontology Language OWL, mittels derer Ontologien erzeugt werden können, um formal die in Web-Dokumenten verwendete Terminologie zu beschreiben. Die Verwendung von z. B. Beschreibungen, Klassen, Eigenschaften, und Restriktionen ist ihnen geläufig.

Die besondere Rolle von Metadaten, etwa in RDF und RDF-Schema oder im Dublin Core, wird anschließend noch einmal herausgestellt.

2. Motivation und Einführung

Das Internet ist eine gigantische Sammlung von Informationen aller Art. Von Kochrezepten bis zu Bauplänen kann man hier fast jede Information finden – wenn man genug Geduld und Zeit aufbringt. Die pure Anzahl von Web-Sites wird für den Nutzenden zu einem Problem. Es kommt zu einem Informationsüberfluss. So meldete z. B. Yahoo im August 2005 eine Indexgröße von 19,2 Milliarden Seiten [RAH03]. Dabei ist ein erheblicher, und meist der qualitativ hochwertigere, Teil der Informationen noch nicht einmal zugänglich, da er auf Anfrage dynamisch aus Datenbeständen zusammengesucht und auf dynamisch generierten Seiten (vgl. Deep Web) angezeigt wird. Darüber hinaus sind vielfältige Informationen derzeit in privaten Datenbanken, etwa von Unternehmen oder Behörden, in digitalen Bibliotheken oder Intranets der breiten Nutzung entzogen. Diese Datenmengen in einem anderen als dem ursprünglich vorgesehenen Kontext oder in Kombination anwenden zu können, böte verschiedenste Möglichkeiten, davon zu profitieren.

Die adäquate Repräsentation von Information und Wissen bildet daher eine wichtige Grundlage für deren effektive Erfassung, Verwaltung, Vermittlung und Nutzung. Die

Verwendung geeigneter Standards erleichtert dabei den Austausch von Information und Wissen zwischen Menschen, zwischen Menschen und Maschinen sowie zwischen Maschinen.

Solche Standards stellen ein gemeinsames Grundverständnis, unabhängig von einer individuellen Aushandlung von Formaten, sicher und ermöglichen damit auch dynamische und wechselnde Kooperationsbeziehungen, die in unserer durch das Internet global vernetzten und sich kontinuierlich verändernden Welt zunehmend an Bedeutung gewinnen.

Das semantische Web als Fortentwicklung des bestehenden WWWs

Die Verfügbarkeit einer kritischen Masse an Information, deren Integration einerseits einen echten Mehrwert darstellt sowie andererseits die Unübersichtlichkeit der vorhandenen Informationsquellen begründet eine zunehmende Bedeutung von Diensten, die eine Art „integrierten Zugriff“ anbieten. Die Bereitstellung eines solchen Zugriffs auf die beschriebene Menge von heterogenen, autonomen und verteilten Informationsquellen und die automatische Verarbeitbarkeit der vorgefundenen Informationen ist somit Voraussetzung für Web-Services aller Art sowie für fortgeschrittene Anwendungen in e-commerce-, e-banking-, e-learning- und anderen Bereichen. Diese Bereitstellung wird allgemein als gemeinsame Informationsnutzung (*information sharing*) bezeichnet.

Quellrepräsentation

information sharing

Derzeit wird der größte Teil des Web-Inhalts im Hinblick auf menschliche Nutzung bereit- und dargestellt. Der Nutzende muss die vorgefundene Information sichten, bewerten und ggf. selbständig mit Informationen aus anderen Quellen verknüpfen. Die Aktivitäten der Benutzenden werden, abgesehen von verschiedenen Suchmaschinen, kaum von Programmen unterstützt. Die Benutzenden stehen daher vor folgenden Problemen [AvH04]:

- Es stellt sich das Problem einer großen Anzahl von Suchergebnissen (Treffer) bei geringer Präzision/Relevanz [HOD01].
- Im Gegensatz dazu kann es vorkommen, dass sich keine oder nur wenige Treffer ergeben.
- Die Suchergebnisse sind stark vom verwendeten Vokabular abhängig. Wenn die Suchbegriffe nicht mit den Schlüsselwörtern der relevanten Dokumente übereinstimmen, werden diese nicht gefunden, obwohl sie in ihrer Bedeutung übereinstimmen.
- Es werden nur einzelne Web-Seiten als Suchergebnis geliefert. Dadurch kann es notwendig werden, weitere Anfragen zu stellen. Danach müssen die Ergebnisse

quasi manuell, und damit mit erheblichem kognitiven Aufwand verknüpft werden.

- Auch bei erfolgreicher Suche müssen die Benutzenden in den gelieferten Dokumenten die interessierende Information manuell, und damit mit erheblichem kognitiven Aufwand, herausfiltern.
- Suchergebnisse sind für andere Programme kaum nutzbar und müssen manuell integriert werden.

maschinelle Verarbeitung

Um diese Probleme bearbeiten zu können, muss grundsätzlich die Möglichkeit bestehen, alle im Web verfügbaren Inhalte maschinell verarbeiten zu können. Dabei muss die dargestellte Information nicht unbedingt immer im Wortsinne vollständig oder auch nur teilweise vom Rechner interpretierbar werden, sondern es reicht in vielen Fällen aus, sie derart maschinell verarbeitbar zu machen, dass mithilfe verschiedener Software-Tools das gewünschte Ergebnis geliefert wird.

semantisches Netz

Dieses Konzept findet als sogenanntes semantisches Netz (engl. *semantic web*) aktuell Verwendung. Computerbasierte Dienste bieten dabei dem Nutzenden bei Suche, Sichtung und Bewertung von Informationen Unterstützung an, indem sie den semantischen Hintergrund der dargestellten Informationen interpretieren und automatisch maschinell weiterverarbeiten. Darüber hinaus sollen Beziehungen zwischen den interpretierten Informationen vom System erkannt werden. Methoden der Wissensrepräsentation und -verarbeitung sollen sowohl eine gemeinsame Nutzung als auch Zugriff auf verteilte Informationsquellen ermöglichen.

Probleme bei der Etablierung eines semantischen Netzes

Als zentrale Probleme erweisen sich im semantischen Netz die syntaktische, strukturelle und semantische Heterogenität der Informationsquellen. Um eine sinnvolle Nutzung derartiger heterogener Informationsquellen gewährleisten zu können, ist die automatische Zusammenführung ihrer Inhalte unumgänglich.

semantische Integration

Dies wird unter dem Begriff der semantischen Integration (engl. *semantic integration*) behandelt. Die technische Umsetzung wird in der Regel durch das Mediato- renmodell (vgl. Kurseinheit 6, Kapitel Lösungsansätze für die semantische Integration) realisiert.

Die eindeutige Darstellung der Semantik der Information in schwach strukturierter Umgebung ist dabei eine wesentliche Voraussetzung, um die mit der Heterogenität der Daten verbundenen Probleme zu lösen [SvH05].

Die Probleme, die sich allgemein aus der Heterogenität der Daten ergeben, sind aus dem Bereich der verteilten Datenbanksysteme bereits gut bekannt. Es lassen sich dabei drei wesentliche Problemfelder benennen:

- die Syntax – z. B. heterogene Datenformate,
- die Struktur – z. B. Homonyme, Synonyme oder unterschiedliche Attributnamen in Datenbanktabellen,
- die Semantik – z. B. die beabsichtigte Bedeutung von Ausdrücken in speziellem Zusammenhang oder Anwendung.

Selbsttestaufgabe 5.1:

Was ist ein semantisches Netz und wozu wird es verwendet?

Lösungsansätze

Zur Lösung der Probleme auf syntaktischer Ebene haben sich mittlerweile zahlreiche Standards herausgebildet. Als Beispiele für solche Standards können Datenbank-Schnittstellen wie Open Database Connectivity (ODBC) oder W3C Web-Technologien wie HTML, XML oder RDF genannt werden, die die Integration unterschiedlicher Informationsquellen erlauben. Daneben existieren auch für Heterogenitätsprobleme auf der strukturellen Ebene bereits ausdifferenzierte Lösungen aus der Datenbank-Forschung. Sowohl die maschinelle Interpretation von Information als auch die Verarbeitung der Semantik befinden sich hingegen noch in einem frühen Forschungsstadium. Deshalb soll im Folgenden vor allem dieser Aspekt besondere Beachtung erfahren. Syntaktische Web-Standards sowie Heterogenitätsprobleme auf der strukturellen Ebene werden jedoch ebenfalls kurz behandelt.

Das semantische Web stellt kein neues System dar, das parallel zum Bestehen des World Wide Web existiert, sondern es soll sich sukzessiv aus diesem herausbilden. Die Entwicklung verläuft dabei auf vier Ebenen:

- die Ebene der technischen Aspekte eines Computernetzwerks,
- die Ebene des Webs als Benutzungsschnittstelle für die Interaktion zwischen Mensch und Internet,
- Wissensebene,
- Anwenderebene eines Wissensnetzwerks im Sinne einer Basis für soziale Netzwerke.

Auf der dritten Ebene kann das semantische Web als verteilte Datenbank bzw. Wissensbasis betrachtet werden, deren Inhalt in maschinenlesbarer und -verarbeitbarer Weise vorliegt und von Softwarewerkzeugen zur Informations-Verarbeitung und zum -

Management automatisch verarbeitet wird. Zudem werden den Benutzenden bestimmte Aufgaben abgenommen:

- Informationssuche,
- Informationssammlung,
- Informationsklassifikation,
- Informationsfilterung,
- Informationsmanagement,
- Information-Mining,
- Informationsentdeckung,
- Informationsbewertung.

Hierzu ist die Ergänzung des aktuellen Inhalts der Web-Dokumente um Daten über deren Semantik, sogenannte Metadaten, notwendig. Dabei ist die fortgeschrittene Standardisierung der Darstellung verschiedener Inhalte auf syntaktischer Ebene von Vorteil. Das Potential von Web-Standards soll daher für die Integration auf semantischer Ebene genutzt werden.

3. Einführung zu Semantik und RDF

Die maschinelle Verarbeitbarkeit von Web-Inhalten setzt voraus, dass diese maschinenlesbar vorliegen und zwischen Software-Tools ausgetauscht werden können. Im Allgemeinen ist dies jedoch nicht der Fall. Stattdessen findet man die Daten im Web in unterschiedlichster Form. Für den Menschen sind diese im Normalfall unmittelbar lesbar und verständlich. Für Maschinen jedoch stellen sie zunächst einmal nur eine willkürliche Ansammlung von Zeichen dar. Erst die Definition einer Syntax gibt den Daten eine bestimmte Struktur, indem diese in formaler Weise erlaubte Struktur-Konstruktionen festlegt und unerlaubte ausschließt. Daten strukturieren zu können ist eine notwendige Bedingung, um das darin codierte Wissen zu erhalten, jedoch fehlt der Zugang zur eigentlichen Bedeutung der Elemente in den erkannten Strukturen, der Semantik.

Das semantische Internet

Web-Inhalte werden zum gegenwärtigen Zeitpunkt vor allem für den Menschen aufbereitet und formatiert. Wissen, welches etwa mittels HTML codiert wird, kann von Benutzenden verstanden werden, obwohl die dafür notwendige Meta-Information, wie etwa Syntax- oder Kontext-Information auf semantischer Ebene, gar nicht explizit vorliegt.

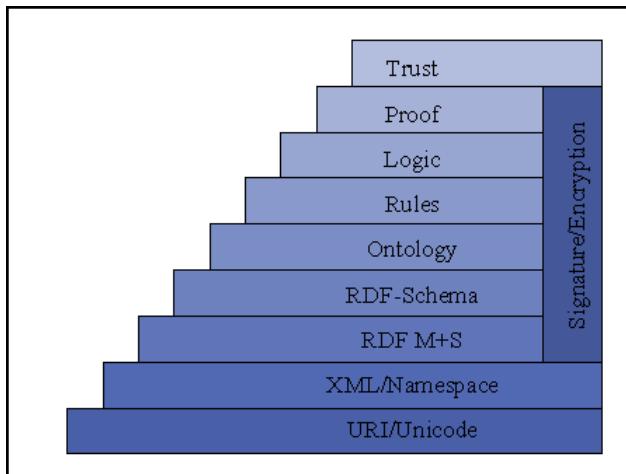


Abbildung 3.1 Semantic Web Layer Cake

Metadaten sind Daten über Daten. Mit Metadaten kann der Sinn von Daten explizit ausgedrückt werden, sie beziehen sich also häufig auf die Semantik. Um der maschinellen Verarbeitung zugänglich zu werden, müssen sie ebenfalls geeignet strukturiert werden. Strukturierte Metadaten erlauben es u. a., in formaler Weise strukturierten Daten eine Bedeutung zuzuweisen und damit Daten auch maschinell hinsichtlich ihrer Semantik interpretierbar zu machen. Erst die Einordnung in einen bestimmten (problem- oder handlungsorientierten) Kontext macht aus diesem explizit semantisch codierten Wissen jedoch Information im eigentlichen, problem- oder handlungsorientierten Sinn. Um Wissen einer bestimmten Domäne formal zu repräsentieren und prinzipiell unabhängig von Programmen wiederverwendbar zu halten, bedarf es einer weiteren Ebene, auf der Instanzen, Konzepte und ihre Beziehungen innerhalb einer Wissensdomäne formal beschrieben und Maschinen dabei unterstützt werden, Inhalte im Web interpretieren zu können, anstatt sie einfach nur darzustellen. Aufbau, Beschreibung und Repräsentation geeigneter Wissensmodelle stellen daher eine weitere wichtige Herausforderung bei der Entwicklung des semantischen Webs dar. Insgesamt wurden vom World Wide Web Consortium (W3C; www.w3c.org) neun (bzw. sieben, je nach Darstellung) aufeinander aufbauende Ebenen des Software-Designs identifiziert, auf denen schrittweise Standards mit wachsender Ausdrucksstärke zur Repräsentation von Daten und Wissen zu entwickeln und durchzusetzen sind, um, ausgehend vom derzeitigen Stand der Entwicklung und auf dessen Grundlage, ein semantisches Web zu installieren [BHL01].

Resource Description Framework

RDF ist unabhängig von einer bestimmten Darstellungsform, weit verbreitet ist jedoch die Repräsentation in XML. Es sind aber auch andere Repräsentationsformen möglich, beispielsweise grafische Darstellungen, welche ebenfalls im Kurstext zu sehen sind.

Die Basis des RDF-Modells ist ein sogenanntes RDF-Tripel. Dieses wird als statement bzw. Aussage bezeichnet, bestehend aus Subjekt bzw. Ressource, Prädikat und Objekt.

Eine Ressource ist beispielsweise eine Web-Site, ein XML-Element oder allgemein ein Objekt.

In einem ersten Schritt kann RDF dazu verwendet werden, einer Ressource Annotationen hinzuzufügen. Auf diese Weise kann die Ressource genauer beschrieben und ihre Beziehung zueinander aufgezeigt werden. Zwei Beispiele können dies verdeutlichen:

Zunächst kann als Ressource die FernUni Hagen betrachtet werden, repräsentiert durch die URI ihrer Webseite: <https://www.fernuni-hagen.de/>. Mittels einer RDF-Annotation kann nun genauer dargestellt werden, was die FernUni Hagen ist (nämlich eine Hochschule):



Abbildung 3.2 RDF-Annotation

Die Notation in XML sähe folgendermaßen aus:

```

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ns1="https://www.fernuni-hagen.de#">
  <rdf:Description rdf:about="https://www.fernuni-
  hagen.de#FernUniHagen">
    <ns1:istEine
      rdf:resource="https://hochschule.art#Hochschule"/>
  </rdf:Description>
</rdf:RDF>
  
```

Sowohl Ressourcen als auch Statements können auf verschiedene Weise dargestellt werden. Die Darstellung als RDF-Graf dient insbesondere dem menschlichen Verständnis. Um Maschinen den Zugang zu RDF-Repräsentationen zu ermöglichen, werden sie in XML notiert. Hierzu dient der Tag `rdf:RDF`, welcher ein RDF-Dokument bezeichnet. Desse[n] Inhalt wird mit verschiedenen `description` Tags (`rdf:description`) beschrieben.

Während im Fall der Ressource „FernUni Hagen“ relativ klar ist, worum es sich dabei handelt – zumindest für einen menschlichen Betrachtenden, bei maschineller Verarbei-

tung ist das nicht unbedingt gegeben – kann ein zweites Beispiel den Sinn von Annotations noch mehr verdeutlichen:

Gegeben ist die Ressource „Bild eines Jaguars“, repräsentiert durch die URI, unter der das Bild zu finden ist: <https://de.wikipedia.org/wiki/Jaguar>

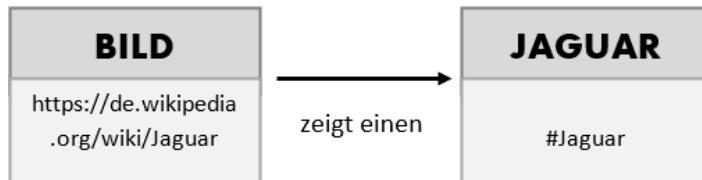


Abbildung 3.3 Semantische Ambiguität

Hier ist nun nicht mehr ohne Weiteres erkennbar, ob es sich bei der Ressource um das Bild eines Tieres oder eines Autos handelt, da die Bildbeschreibung nur textueller Natur ist. Erst bei der Betrachtung des Bildes kann der Begriff, bzw. dessen Bedeutung (Semantik), korrekt zugeordnet werden.

Wollte man diese Semantik auch noch durch eine zusätzliche Annotation explizit machen, dann würde man hierfür ein semantisches Vokabular benötigen. Hiermit ist es möglich, den Begriff „Jaguar“ einzubetten und somit innerhalb einer Taxonomie eindeutig zu machen, auch ohne das eigentliche Bild anschauen zu müssen. In Kapitel 5 wird dieser Aspekt noch einmal aufgegriffen.

Diese Annotation der Ressource „Bild eines Jaguars“ würde demzufolge so aussehen:

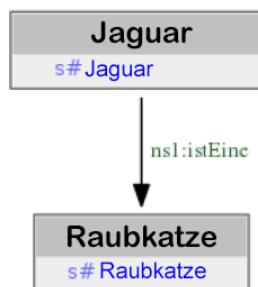


Abbildung 3.4 RDF-Annotation 2

```

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ns1="https://de.wikipedia.org/wiki#">
  <rdf:Description
    rdf:about="https://de.wikipedia.org/wiki#Jaguar">
    <ns1:istEine
      rdf:resource="https://de.wikipedia.org/wiki#Raubkatze"/>
  
```

```
</rdf:Description>
</rdf:RDF>
```

Eine Ressource kann nicht nur annotiert werden, es ist auch möglich, eine Aussage über sie zu treffen. Jede Information über eine Ressource wird dann ebenfalls als RDF-Tripel dargestellt. Wie bereits gezeigt wurde, hat jede Ressource einen eindeutigen Bezeichner in Form von URI oder URL. Das Prädikat ist eine spezielle Ressource, die eine Relation beschreibt, welche das Subjekt mit einem bestimmten Objekt verbindet. Es wird ebenfalls mit einem Uniform Ressource Identifier identifiziert.

RDF lässt lediglich zweistellige Relationen zu. Das Objekt ist wiederum eine Ressource oder ein Literal. Die Verwendung von URIs zur Identifizierung ermöglicht ein einheitliches, globales Namensschema ([AvH04], S. 64).

Abbildung 3.5 gibt diesen Sachverhalt anschaulich wieder. Zu beachten ist hierbei, dass in diesem Beispiel bereits zwei Aussagen miteinander verkettet sind:

Aussage 1: Max studiert an der FernuniHagen

Aussage 2: Die FernUni Hagen ist eine Hochschule

Eine solche Verkettung ist abzugrenzen von der sogenannten Reifikation von Aussagen, die im weiteren Verlauf des Kapitels noch ausführlich behandelt wird und es ermöglicht, eine Aussage über eine Aussage zu treffen.

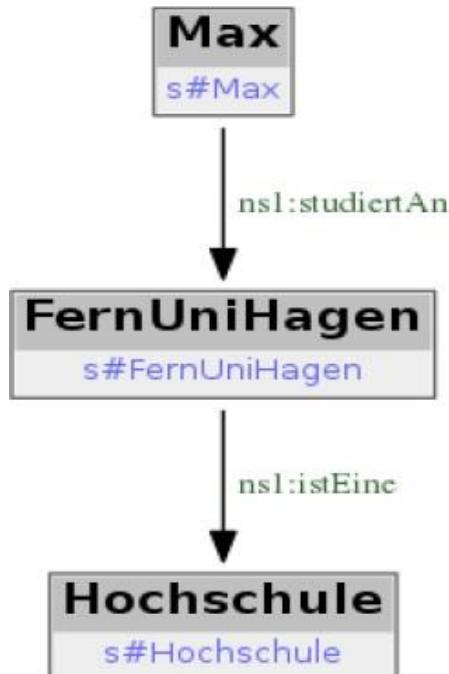


Abbildung 3.5 Beispielhafter RDF-Graph

XML-Notation zu Abbildung 3.5:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ns1="s#"
>
<rdf:Description rdf:about="s#Max">
  <ns1:studiertAn>
    <rdf:Description rdf:about="s#FernUniHagen">
      <ns1:istEine rdf:resource="s#Hochschule"/>
    </rdf:Description>
  </ns1:studiertAn>
</rdf:Description>
</rdf:RDF>
```

RDF bietet einige für die Repräsentation von Metadaten wichtige Eigenschaften. Dazu gehören die sogenannten Reifikations- und Aggregationsmechanismen. Ersterer erlaubt es, Aussagen über Aussagen zu treffen, indem ein eindeutiger Bezeichner, auch ID genannt, für jedes Statement eingeführt wird, wodurch es zu einer Ressource wird.

RDF erlaubt außerdem, dass ein Merkmal mehrere Werte annehmen kann. Hierfür stellt RDF Datentypen für Listen und Mengen von Ressourcen zur Verfügung, die ihrerseits wiederum Ressourcen darstellen.

Als Nachteil gilt, dass nur zweistellige Relationen zugelassen sind, sodass mehrstellige umständlich durch mehrere zweistellige Relationen dargestellt werden müssen. Die ID etwa muss in drei Tripeln, jeweils mit jedem der drei Teile von statement, über die Relationen subject, predicate, object verknüpft werden. Die große Flexibilität und Komplexität von RDF ist auf dieser Ebene der Modellierung von Daten eher unüblich und unnötig. RDF ist insgesamt als Modellierungssprache nicht optimal ([AvH04], S. 69). Trotzdem hat es sich zu einem de facto-Standard zur Modellierung von Information und expliziter Codierung von Wissen entwickelt. Der dezentrale Ansatz von RDF erlaubt es, Wissensbasen schrittweise aufzubauen, wiederzuverwenden und durch verschiedene Anwendungen gemeinsam zu nutzen. RDF ist unabhängig von einer bestimmten Domäne. Daher obliegt es dem Nutzenden, auf einer nächsten Ebene seine eigene Terminologie in einer Schema-Sprache, RDFs genannt, zu definieren.

Selbsttestaufgabe 5.2:

- Welches sind die Bestandteile eines RDF-Tripels?
- Nennen Sie zwei mögliche Darstellungsformen.

Reifikation

Reifikation beschreibt die Fähigkeit, eine Aussage als eigene Ressource zu modellieren, was es erlaubt, Aussagen in anderen Aussagen zu referenzieren. Dies ermöglicht die Verkettung von Aussagen und das Treffen von Aussagen über Aussagen.

Eine reifizierte Aussage stellt eine eigene Ressource dar, auf die in anderen Aussagen Bezug genommen werden kann. Jede Anweisung besteht aus einem Tripel Subjekt, Prädikat, Objekt und kann in ihrer reifizierten Form dargestellt werden.

Danach kann sie als Referenz in anderen Anweisungen verwendet werden. Nimmt man also das folgende N-Triple als Beispiel einer Aussage:

```
<rdf:subject> <rdf:predicate> <rdf:object> .
```

so ist dies das N-Triple der Aussage in reifizierter Form:

```
<statement> <rdf:type> <rdf:Statement> .  
<statement> <rdf:subject> <subject> .  
<statement> <rdf:predicate> <predicate> .  
<statement> <rdf:object> <object> .
```

In der Form als RDF/XML sieht das gleiche Statement folgendermaßen aus:

```
RDF/XML (requires URI identifier for resources):  
<?xml version="1.0" encoding="utf-8" ?>  
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">  
  <rdf:Statement rdf:nodeID="statement">  
    <rdf:subject  
      rdf:resource="https://resource.identifier/subject"/>  
    <rdf:predicate  
      rdf:resource="https://resource.identifier/predicate"/>  
    <rdf:object  
      rdf:resource="https://resource.identifier/object"/>  
  </rdf:Statement>  
</rdf:RDF>
```

Repräsentation in grafischer Form:

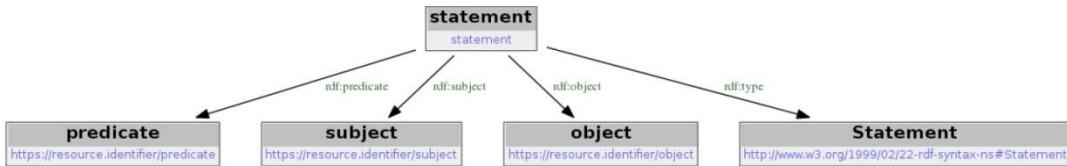


Abbildung 3.6 Aussage über Aussage

Das folgende Beispiel behandelt ebenfalls die Reifikation von Aussagen in RDF. Folgender Sachverhalt soll in RDF dargestellt werden:

[Anton Stadler spielte das Holzblasinstrument Klarinette.](#)

In dem Sachverhalt verbergen sich zwei Aussagen:

[Anton Stadler spielte Klarinette.](#)

[Klarinette ist ein Holzblasinstrument.](#)

Die Aussagen können wie folgt in RDF dargestellt werden:

N-Triple:

```

<musiker:AntonStadler> <musiker:spielt>
<instrument:Klarinette> .
<instrument:Klarinette> <instrument:istEin>
<instrument:Holzblasinstrument> .
  
```

RDF/XML:

```

<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntactic-ns#" xmlns:ns0="musiker:" xmlns:ns1="instrument:">
<rdf:Description rdf:about="musiker:AntonStadler">
<ns0:spielt>
<rdf:Description rdf:about="instrument:Klarinette">
<ns1:istEin rdf:resource="instrument:Holzblasinstrument"/>
</rdf:Description>
<ns0:spielt>
</rdf:Description>
  
```

```
</rdf:RDF>
```



Abbildung 3.7 Reifikation

Um beide Aussagen und ihre Beziehung zueinander auszudrücken, kann auch die Reifikation genutzt werden. Dies stellt den Sachverhalt deutlich klarer dar und vereinfacht das Referenzieren von Aussagen in mehreren, unterschiedlichen Aussagen.

Dafür wird eine der Aussagen (meist die innere Aussage, auf die sich der andere Teil der Aussage bezieht), in ihre reifizierte Form transformiert:

N-Triple:

```

<musiker:AntonStadler>
<musiker:spielt>
<instrument:Statement> .
<instrument:Statement>
<rdf:type> <rdf:Statement> .
<instrument:Statement>
<rdf:subject> <instrument:Klarinette> .
<instrument:Statement> <rdf:predicate>
<instrument:istEin> .
<instrument:Statement>
<rdf:object>
<gattung:Holzblasinstrument> .

```

RDF/XML verschachtelt:

```

<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntacticns#" xmlns:musiker="musiker:">
<rdf:Description rdf:about="musiker:AntonStadler">
<musiker:spielt>
<rdf:Description rdf:about="instrument:Statement">
<rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntacticns# Statement"/>
<rdf:subject rdf:resource="instrument:Klarinette"/>
<rdf:predicate rdf:resource="instrument:istEin"/>
<rdf:object rdf:resource="gattung:Holzblasinstrument"/>
</rdf:Description>

```

```
</musiker:spielt>
</rdf:Description>
</rdf:RDF>
```

RDF/XML vereinfacht:

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:ns0="musiker:">
<rdf:Description rdf:about="musiker:AntonStadler">
<ns0:spielt rdf:resource="instrument:Statement"/>
</rdf:Description>
<rdf:Statement rdf:about="instrument:Statement">
<rdf:subject rdf:resource="instrument:Klarinette"/>
<rdf:predicate rdf:resource="instrument:istEin"/>
<rdf:object rdf:resource="gattung:Holzblasinstrument"/>
</rdf:Statement>
</rdf:RDF>
```

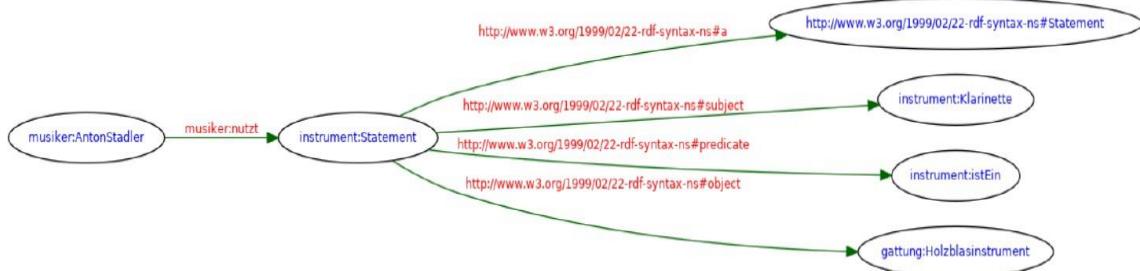


Abbildung 3.8 Reifikation

Selbsttestaufgabe 5.3:

Erläutern Sie das Prinzip der Reifikation anhand des folgenden N-Triples:

N-Triple:

```
<sh:SherlockHolmes> <s:supposes> <s:StatementOfSherlock>

<s:StatementOfSherlock> http://www.w3.org/1999/02/22-rdf-
syntax-ns#

<rdf:Statement> .

<s:StatementOfSherlock> <http://www.w3.org/1999/02/22-rdf-
syntaxns#

subject> <s:Gardener> .

<s:StatementOfSherlock> <http://www.w3.org/1999/02/22-rdf-
syntaxns#

predicate> <s:hasKilled>

<s:StatementOfSherlock> <http://www.w3.org/1999/02/22-rdf-
syntaxns# object> <s:Butler> .
```

Argumente

Der folgende Abschnitt befasst sich mit den sogenannten RDF-Argumenten, einer speziellen Form von Aussagen zur Repräsentation von Logik.

Eine Argumentation ist Teil eines Dialogs, bei dem eine Partei eine andere Partei von einer Behauptung mittels Argumenten überzeugen will. [MM09] Argumente werden konstruiert, ausgetauscht und in Bezug auf andere Argumente und die Hauptbehauptung, auf die sie sich beziehen, bewertet.

Es gibt viele verschiedene Definitionen von Argumenten, wobei oft eine gemeinsame Grundlage ist, dass ein Argument aus Prämissen und einer Schlussfolgerung besteht. Die Menge der Prämissen erlaubt die Ableitung der Schlussfolgerung durch Inferenz¹, daher

¹ Inferenz: Wissen, das aufgrund von logischen Schlussfolgerungen gewonnen wurde

enthält die weit verbreitete Definition eines Arguments einen zusätzlichen Inferenzknoten für die Inferenz von den Prämissen zur Schlussfolgerung.

Diese Definition eines Arguments wird im Argument Interchange Format (AIF) verwendet, bei dem RDF und RDFS verwendet werden, um alle Argumentteile als RDF-Ressourcen darzustellen. AIF verwendet Informationsknoten (I-Knoten) und Schemaknoten (S-Knoten), wobei I-Knoten aussagenbezogene Informationen (Behauptung, Prämissen, Daten, ...) darstellen und S-Knoten die Anwendung von Schemata (Argumentation, Inferenz, ...) erfassen. Ein spezieller S-Knoten ist der RA-Knoten, der eine Regel der Inferenzanwendung darstellt.

Das folgende Argument besteht aus zwei I-Knoten (eine Prämisse, eine Konklusion) und einem S-Knoten (ein RA-Knoten). Die Behauptung in der Prämisse kann vom RA-Knoten zur Inferenz der Konklusion verwendet werden. Dies wird im folgenden Grafen, N-Tripel und RDF/XML dargestellt.

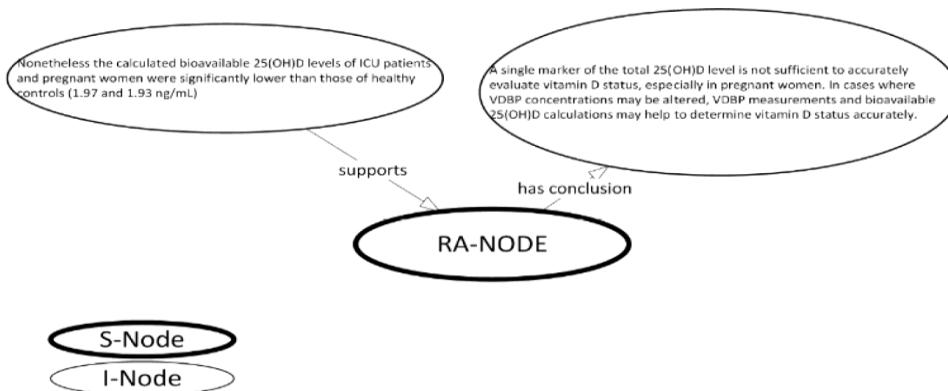


Abbildung 3.9 AIF Triple [NDH21], Text: [Kim17]

N-Triple:

```

<inode-conclusion> <http://www.arg.tech/aif#claimText> "A
single marker of the total 25(OH)D level is not sufficient
to accurately evaluate vitamin D status, especially in
pregnant women. In cases where VDBP concentrations may be
altered, VDBP measurements and bioavailable 25(OH)D calcu-
lations may help to determine vitamin D status accurate-
ly." .
<ra-node> <http://www.arg.tech/aif#Conclusion> <inode-
conclusion> .
<inode-premise> <http://www.arg.tech/aif#Premise> <ra-
node> .
<ra-node> <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>

```

```

<http://www.arg.tech/aif#S-node> .
<inode-conclusion>      <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type>
<http://www.arg.tech/aif#I-node> .
<inode-premise> <http://www.arg.tech/aif#claimText> "None-
theless the calculated bioavailable 25(OH)D levels of ICU
patients and pregnant women were significantly lower than
those of healthy controls (1.97 and 1.93 ng/mL)" .
<inode-premise> <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type>
<http://www.arg.tech/aif#I-node> .

```

RDF/XML:

```

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns="http://www.w3.org/2002/07/owl#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:www="http://www.ArgOWL.org#"
xmlns:aif="http://www.arg.tech/aif#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<aif:I-node rdf:about="inode-conclusion">
<aif:claimText>A single marker of the total 25(OH)D level
is not sufficient to accurately evaluate vitamin D status,
especially in pregnant women. In cases where VDBP concen-
trations may be altered, VDBP measurements and bioavaila-
ble 25(OH)D calculations may help to determine vitamin D
status accurately.</aif:claimText>
</aif:I-node>
<aif:I-node rdf:about="inode-premise">
<aif:Premise rdf:resource="ra-node"/>
<aif:claimText>Nonetheless the calculated bioavailable
25(OH)D
levels of ICU patients and pregnant women were signifi-
cantly lower than those of healthy controls (1.97 and 1.93
ng/mL)</aif:claimText>
</aif:I-node>
<aif:S-node rdf:about="ra-node">
<aif:Conclusion rdf:resource="inode-conclusion"/>
</aif:S-node>
</rdf:RDF>

```

Argumentationsbaum

Argumente bestehen aus Prämissen und Schlussfolgerungen, wobei die Prämisse eines Arguments auch ein anderes Argument sein kann. Dies ermöglicht die Verkettung von Argumenten, was zu einer Argumentationsbaumstruktur führt (siehe Abbildung 3.10).

Die vereinfachte Argumentationsbaumstruktur stellt zwei Argumente dar (inneres und äußeres Argument), wobei das innere Argument das äußere Argument stützt. Das innere Argument besteht aus einer Prämisse und einer abgeleiteten Schlussfolgerung. Das äußere Argument hingegen hat zwei Prämissen (das innere Argument und eine weitere Prämisse) und leitet seine Schlussfolgerung dementsprechend aus zwei Prämissen ab. Abbildung 3.10 zeigt den Argumentationsbaum-Strukturgrafen, basierend auf beiden Argumenten.

Zusätzlich können die N-Tripel und RDF/XML-Darstellungen aus dem Grafen abgeleitet werden, wie unten dargestellt.

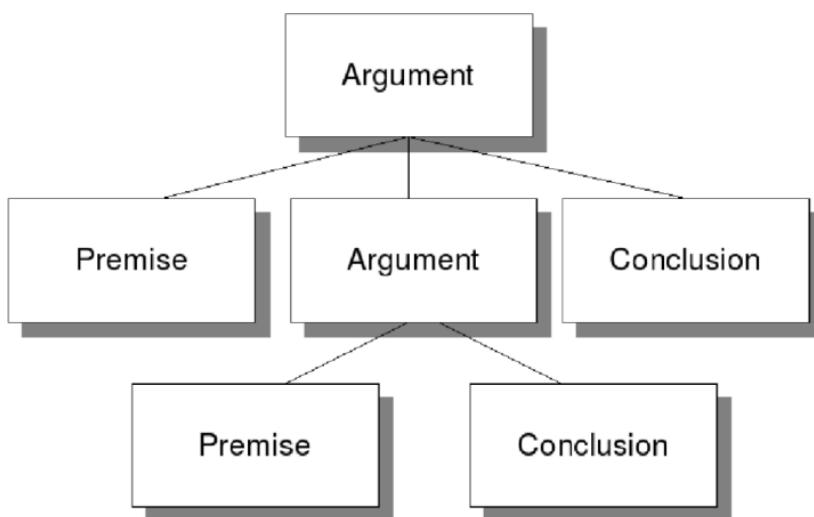


Abbildung 3.10 Einfaches Beispiel einer Argumentationsbaumstruktur [MM09]

N-Triples:

```

<inode-Premise2> <http://www.arg.tech/aif#supports> <outer-argument> .
<inode-Premise1> <http://www.arg.tech/aif#supports> <inner-argument> .
<inner-argument> <http://www.arg.tech/aif#hasConclusion>
<inode- Conclusion1> .
<inode-Premise2> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.arg.tech/aif#Premise> .
  
```

```

<inner-argument> <http://www.arg.tech/aif#supports> <outer-argument> .
<inode-Premise1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.arg.tech/aif#Premise> .
<inode-Conclusion1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.arg.tech/aif#Conclusion> .
<inode-Conclusion2> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.arg.tech/aif#Conclusion> .
<outer-argument> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.arg.tech/aif#Argument> .
<inner-argument> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.arg.tech/aif#Argument> .
<outer-argument> <http://www.arg.tech/aif#hasConclusion>
<inode- Conclusion2> .

```

RDF/XML:

```

<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns="http://www.w3.org/2002/07/owl#"
           xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
           xmlns:www="http://www.ArgOWL.org#"
           xmlns:aif="http://www.arg.tech/aif#"
           xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <aif:Argument rdf:about="inner-argument">
    <aif:hasConclusion rdf:resource="inode-Conclusion1" />
    <aif:supports rdf:resource="outer-argument"/>
  </aif:Argument>
  <aif:Conclusion rdf:about="inode-Conclusion1" />
  <aif:Premise rdf:about="inode-Premise1">
    <aif:supports rdf:resource="inner-argument"/>
  </aif:Premise>
  <aif:Argument rdf:about="outer-argument">
    <aif:hasConclusion rdf:resource="inode-Conclusion2" />
  </aif:Argument>
  <aif:Conclusion rdf:about="inode-Conclusion2" />
  <aif:Premise rdf:about="inode-Premise2">

```

```
<aif:supports rdf:resource="outer-argument"/>
</aif:Premise>
</rdf:RDF>
```

Selbsttestaufgabe 5.4:

Woraus besteht ein Argument im Sinne des Argument Interchange Format (AIF)? Was sind die Aufgaben der sogenannten S-Knoten, bzw. I-Knoten?

4. RDF Schemata

Während XML-S die Struktur eines XML-Dokuments festlegt, definiert RDF-S das Vokabular, welches in RDF-Datenmodellen Verwendung findet, und spezifiziert, welche Merkmale zu welchen Objekten gehören. Weiter wird spezifiziert welche Werte die Objekte annehmen können und die Beziehungen zwischen Objekten werden beschrieben.

Indem RDF-S die Bedeutung gewisser Bestandteile wie Class, subClassOf usw. festlegt, ermöglicht es, bestimmte Wissensbereiche zu modellieren, d. h. eine Ontologie im Sinne einer Wissensbasis anzulegen. Der Begriff der Ontologie stammt aus der Philosophie und bedeutet „die Lehre des Seins“. In unserem Fall soll die Ontologie Begriffe und Konzepte für Beschreibungen definieren.

Ontologie und
Wissensbasis

Diese Bedeutung ist festgelegt und unterliegt nicht der Interpretation durch verschiedene Anwendungen. Da jedes RDF-Schema seinen eigenen Namensraum via Präfix verwendet, können Ausdrücke aus verschiedenen Modellen in einem RDF-Dokument ohne Konflikte gemischt werden. RDF definiert eine klare Objektstruktur, sodass es möglich ist, Aussagen über Objekte in einer anderen Sprache zu treffen als in der, in der dieses definiert worden ist [DMHF00].

Die Festschreibung der Bedeutung verschiedener Sprachelemente lässt sich am folgenden Beispiel illustrieren. Ein Ausschnitt aus dem Universitätsgeschehen ließe sich mithilfe von XML-Elementen folgendermaßen darstellen:

Ein weiterer beispielhafter RDF-Graf [SvH05]:

```
<academicStaffMember>
    Grigoris Antoniou
</academicStaffMember >
<professor> Michael Maher </professor >
<course name=" Discrete Mathematics ">
    <isTaughtBy> David Billington </isTaughtBy >
```

```
</course>
```

Um zu überprüfen, ob die Darstellung in XML-Syntax valide ist, wird der [Online Validator des W3C](#)¹ empfohlen.

Will man nur aus dieser Darstellung heraus alle akademischen Mitarbeiter finden, z. B. indem via XPath eine Anfrage formuliert wird: //academicStaffMember, wird als Ergebnis lediglich G.A. geliefert. Das ist zwar aus XML-Sicht korrekt, semantisch jedoch unbefriedigend. M.M. und D.B. wären ebenfalls zu erwarten, da alle Professoren ja auch akademische Mitarbeiter (subclass!) sind. Kurse werden normalerweise nur von akademischen Mitarbeitern gehalten. Erst mit der Formulierung von bestimmtem Hintergrundwissen, wie etwa: „professor is a subclass of academic staff member“ – also durch „Alle Professoren sind auch akademische Mitarbeiter“ – wird es möglich, das Konzept „Unterkasse“ korrekt zu beschreiben und das erwartete Ergebnis der Abfrage zu generieren. Dazu ist ein semantisches Modell des Wissensbereichs nötig, welches nicht in XML oder RDF dargestellt werden kann und daher typisch ist für Wissen, welches in RDF-S beschrieben wird.

RDF-S

RDF-S ist somit bereits eine primitive Ontologie-Sprache. Sie stellt verschiedene Primitive mit festgelegter Bedeutung zur Verfügung. Dazu gehören die Konzepte von class und subclass-Beziehungen, property und subproperty-Beziehungen sowie domain- und range-Einschränkungen. Mithilfe von speziellen Abfragesprachen können Abfragen über RDF bzw. RDF-S Dokumente formuliert werden. Als formale Sprache zur Beschreibung von RDF-S-Dokumenten wird RDF verwendet, d. h. alle Primitive werden als Tripel definiert.

Demzufolge sind RDF-S-Dokumente RDF-Dokumente mit der entsprechenden XML-basierten Syntax. Dadurch ist eine syntaktische Interoperabilität, die den Austausch und die gemeinsame Verwendung von Daten gewährleistet, vorhanden. Auf diese Art macht RDF-S semantische Information maschinell zugänglich, wie es der Vorstellung eines semantischen Netzes entspricht, wobei seine eingeschränkten Möglichkeiten eine komplexere Modellierung der Semantik auf höherer Ebene erforderlich machen.

Klassen

Basis-Klassen in RDF-S sind

- rdfs: Ressource, die Klasse aller Ressourcen,
- rdfs: Class, die Klasse aller Klassen,

¹ <https://www.w3.org/RDF/Validator/>

- rdfs: Literal, die Klasse aller Literale (strings), derzeit der einzige Datentyp von RDF/RDF-S,
- rdf: Property, die Klasse aller Eigenschaften,
- rdf: Statement, die Klasse aller Aussagen.

Die Klasse lecturer kann so z. B. folgendermaßen definiert werden:

```
<rdfs: Class rdf:ID="lecturer">
...
</rdfs: Class >
```

Klassen können als Mengen von Elementen aufgefasst werden. Einzelne Objekte bzw. Ressourcen können über die type-Beziehung einer bestimmten Klasse zugeordnet werden. Klassen können über die subClassOf-Beziehung zu Hierarchien verknüpft werden, wobei Subklassen die Eigenschaften und Beschränkungen der Superklassen erben.

Beziehungen

Beziehungen zwischen Klassen oder Instanzen und Klassen werden über folgende Basis-Eigenschaften definiert:

- rdf: type, verknüpft eine Ressource mit ihrer Klasse, d. h. die Ressource wird zu einer Instanz einer Klasse erklärt,
- rdfs: subClassOf, verknüpft eine Klasse mit einer ihrer Elternklassen, d. h. die Klasse wird zu einer UnterkLASSE erklärt,
- rdfs: subPropertyOf, verknüpft eine Eigenschaft mit einer ihrer Elterneigenschaften.

Ein bekanntes Beispiel ist hier die sogenannte „Taxonomie des Lebens“:

Die Taxonomie des Lebens beschreibt die Lehre von der Benennung, Definition und Klassifizierung von Gruppen biologischer Organismen nach gemeinsamen Kriterien.

Das folgende Beispiel zeigt ein verkürztes Beispiel dieser Taxonomie. Ausgangspunkt und Spitze der Taxonomie ist die Taxonomie selbst, danach wird sie immer spezifischer in der Gruppierung der Merkmale von Lebewesen.

Schauen wir uns die verkürzte Taxonomie des Lebens für das Tier Jaguar an. Er wird aufgrund seiner Eigenschaften als „Katze“ klassifiziert, zusammen mit anderen Tieren wie dem „Löwen“. Katzen sind eine eigene Tiergruppe in der Taxonomie und gehören zu der höheren Gruppe der „Raubtiere“, die andere Lebewesen jagen.

Eine weitere Gruppe von Tieren sind die Nagetiere, denen eine Maus zugeordnet wird. Die Raubtiere wiederum gehören zur Hauptgruppe der Tiere, die in der verkürzten Taxonomie des Lebens direkt dem obersten Knotenpunkt zugeordnet sind.

N-Triple:

```
<Taxonomie-der-Lebewesen:fieldMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:longTailedMice> .
<Taxonomie-der-Lebewesen:animals>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:livingBeings> .
<Taxonomie-der-Lebewesen:plants>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:livingBeings> .
<Taxonomie-der-Lebewesen:cats>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:predators> .
<Taxonomie-der-Lebewesen:lions>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:cats> .
<Taxonomie-der-Lebewesen:houseMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:longTailedMice> .
<Taxonomie-der-Lebewesen:predators>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:animals> .
<Taxonomie-der-Lebewesen:jaguars>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:cats> .
<Taxonomie-der-Lebewesen:rodents>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:animals> .
<Taxonomie-der-Lebewesen:longTailedMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <Taxonomie-der-Lebewesen:rodents> .
```

RDF/XML:

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

```
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:fieldMice">
<rdfs:subClassOf>
<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:longTailedMice">
<rdfs:subClassOf rdf:resource="Taxonomie-der-
Lebewesen:rodents"/>
</rdf:Description>
</rdfs:subClassOf>
</rdf:Description>
<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:plants">
<rdfs:subClassOf rdf:resource="Taxonomie-der-
Lebewesen:livingBeings"/>
</rdf:Description>
<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:lions">
<rdfs:subClassOf>
<rdf:Description rdf:about="Taxonomie-der-Lebewesen:cats">
<rdfs:subClassOf rdf:resource="Taxonomie-der-
Lebewesen:predators"/>
</rdf:Description>
</rdfs:subClassOf>
</rdf:Description>
<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:houseMice">
<rdfs:subClassOf rdf:resource="Taxonomie-der-
Lebewesen:longTailedMice"/>
</rdf:Description>
<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:jaguars">
<rdfs:subClassOf rdf:resource="Taxonomie-der-
Lebewesen:cats"/>
</rdf:Description>
<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:animals">
<rdfs:subClassOf rdf:resource="Taxonomie-der-
Lebewesen:livingBeings"/>
</rdf:Description>
```

```

<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:predators">
<rdfs:subClassOf rdf:resource="Taxonomie-der-
Lebewesen:animals"/>
</rdf:Description>
<rdf:Description rdf:about="Taxonomie-der-
Lebewesen:rodents">
<rdfs:subClassOf rdf:resource="Taxonomie-der-
Lebewesen:animals"/>
</rdf:Description>
</rdf:RDF>

```

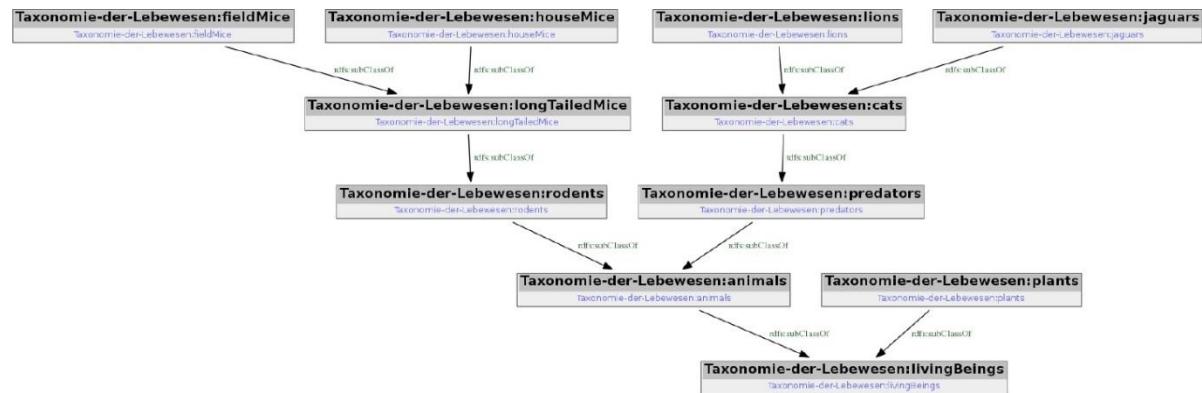


Abbildung 4.1 Taxonomie des Lebens, Ausschnitt

Dass die Taxonomie des Lebens in RDFS-Klassen formalisiert werden kann, erlaubt es uns, Objekte einer bestimmten Klasse zu instanziiieren.

Als Beispiel wird der Jaguar franz vorgestellt. Er hat die annotierte Eigenschaft, schnell zu sein. Es ist nicht klar, ob franz ein Tier oder ein Auto ist, denn das Attribut schnell ist nicht weiter definiert und kann ein Tier oder ein Auto beschreiben. Aufgrund der Instanziierung von franz als Objekt der Taxonomie des Lebens ist es maschinell entscheidbar, dass franz eindeutig ein Tier ist.

RDF / XML Notation:

```
TaxonomieDerLebewesen:fieldMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:longTailedMice> .
<TaxonomieDerLebewesen:animals>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:livingBeings> .
<TaxonomieDerLebewesen:plants>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:livingBeings> .
<TaxonomieDerLebewesen:cats>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:predators> .
<TaxonomieDerLebewesen:lions>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:cats> .
<TaxonomieDerLebewesen:houseMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:longTailedMice> .
<TaxonomieDerLebewesen:predators>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:animals> .
<TaxonomieDerLebewesen:jaguars>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:cats> .
<TaxonomieDerLebewesen:rodents>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:animals> .
<TaxonomieDerLebewesen:longTailedMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:rodents> .
<jaguar:franz> <rdf:type> <TaxonomieDerLebewesen:jaguars>
.
<jaguar:franz> <Eigenschaft:ist> <Eigenschaft:schnell> .
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:ns0="Eigenschaft:">
<rdf:Description
rdf:about="TaxonomieDerLebewesen:fieldMice">
```

```
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:longTailedMice"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:animals">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:livingBeings"/>
</rdf:Description>
<rdf:Description rdf:about="TaxonomieDerLebewesen:plants">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:livingBeings"/>
</rdf:Description>
<rdf:Description rdf:about="TaxonomieDerLebewesen:cats">
<rdfs:subClassOf>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:predators">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:animals"/>
</rdf:Description>
</rdfs:subClassOf>
</rdf:Description>
<rdf:Description rdf:about="TaxonomieDerLebewesen:lions">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:cats"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:houseMice">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:longTailedMice"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:jaguars">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:cats"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:rodents">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:animals"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:longTailedMice">
```

```

<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:rodents"/>
</rdf:Description>
<rdf:Description rdf:about="jaguar:franz">
<rdf:type rdf:resource="TaxonomieDerLebewesen:jaguars"/>
<ns0:ist rdf:resource="Eigenschaft:schnell"/>
</rdf:Description>
</rdf:RDF>

```



Abbildung 4.2 Instanz des Jaguars franz

Um das Beispiel des vorigen Kapitels noch einmal aufzugreifen, wird nun noch das folgende Szenario betrachtet: Zusätzlich existiert ein Foto von franz, aus dem ersichtlich ist, dass es sich bei franz um ein Tier und nicht um ein Auto handelt. Für eine Maschine ist in RDF nicht erkennbar, dass franz ein Tier und kein Auto ist, da das Foto nicht in RDF verarbeitet werden kann.

Durch die Instanziierung von franz als ein Objekt der Taxonomie des Lebens Klasse Jaguar ist es auch maschinell entscheidbar, dass franz eindeutig ein Tier ist, auch ohne das Foto zu betrachten.

RDF / XML Notation:

```

TaxonomieDerLebewesen:fieldMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:longTailedMice> .
<TaxonomieDerLebewesen:animals>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:livingBeings> .
<TaxonomieDerLebewesen:plants>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:livingBeings> .
<TaxonomieDerLebewesen:cats>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:predators> .
<TaxonomieDerLebewesen:lions>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:cats> .

```

```
<TaxonomieDerLebewesen:houseMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:longTailedMice> .
<TaxonomieDerLebewesen:predators>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:animals> .
<TaxonomieDerLebewesen:jaguars>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:cats> .
<TaxonomieDerLebewesen:rodents>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:animals> .
<TaxonomieDerLebewesen:longTailedMice>
<http://www.w3.org/2000/01/rdfschema# subClassOf> <TaxonomieDerLebewesen:rodents> .
<Jaguar:franz> <rdf:type> <TaxonomieDerLebewesen:jaguars>
.
<Jaguar:franz> <Jaguar:fotografiert> <Tierfoto:BildVonFranz> .
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:ns0="Jaguar:">
<rdf:Description
rdf:about="TaxonomieDerLebewesen:fieldMice">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:longTailedMice"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:animals">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:livingBeings"/>
</rdf:Description>
<rdf:Description rdf:about="TaxonomieDerLebewesen:plants">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:livingBeings"/>
</rdf:Description>
<rdf:Description rdf:about="TaxonomieDerLebewesen:cats">
<rdfs:subClassOf>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:predators">
```

```

<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:animals"/>
</rdf:Description>
</rdfs:subClassOf>
</rdf:Description>
<rdf:Description rdf:about="TaxonomieDerLebewesen:lions">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:cats"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:houseMice">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:longTailedMice"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:jaguars">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:cats"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:rodents">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:animals"/>
</rdf:Description>
<rdf:Description
rdf:about="TaxonomieDerLebewesen:longTailedMice">
<rdfs:subClassOf
rdf:resource="TaxonomieDerLebewesen:rodents"/>
</rdf:Description>
<rdf:Description rdf:about="Jaguar:franz">
<rdf:type rdf:resource="TaxonomieDerLebewesen:jaguars"/>
<ns0:fotografiert rdf:resource="Tierfoto:BildVonFranz"/>

```



Abbildung 4.3 Instanz des Jaguars franz (Bild)

Einschränkungen

Die Gültigkeit von Eigenschaften hinsichtlich ihres Wertebereichs kann eingeschränkt werden, sodass unsinnige Aussagen wie „Der Kurs diskrete Mathematik wird von dem Kurs konkrete Mathematik unterrichtet“ ausgeschlossen werden. Hierzu dienen folgende Eigenschaften:

- rdfs: domain, spezifiziert die Klassen, auf deren Elemente eine bestimmte Eigenschaft zutreffen kann, d. h. die Subjekte, die in einem Tripel SPO zu einem Prädikat P gehören können. Ohne solche Einschränkung kann jede Ressource Subjekt sein.
- rdfs: range, spezifiziert die Klassen, deren Elemente Objekt einer bestimmten Eigenschaft sein können.

weitere Primitive

Wie schon erwähnt können mit RDF Aussagen über Aussagen getroffen werden. Hierzu dienen die Primitive rdf: subject, rdf: predicate und rdf: object, die ein Statement mit dessen Subjekt, Prädikat bzw. Objekt verknüpfen. Eine weitere erwähnte Eigenschaft von RDF ist die Möglichkeit, Ressourcen oder Attribute, über die Aussagen getroffen werden sollen, zusammenzufassen. Dieser Vorgang wird Aggregation genannt. Hierzu dienen die Container-Klassen:

- rdf: bag für Menge,
- rdf: Seq für geordnete n-Tupel,
- rdf: Alt für eine Menge von Alternativen jeweils mit der Superklasse rdfs: Container.

Ressourcen können auf verschiedene Art und Weise an unterschiedlichen Stellen im Web definiert und beschrieben sein. Ein entsprechender Bezug kann mittels rdfs: seeAlso bzw. deren subproperty rdfs: isDefinedBy hergestellt werden.

Benutzerfreundliche Kommentare oder Namen von Ressourcen können mithilfe der Properties rdfs: comment bzw. rdfs: label in ein RDF-S Dokument eingefügt werden.

Selbsttestaufgabe 5.5:

Fassen Sie die Klassenhierarchie, die im XML-Dokument „Instrumente“ (auch im Anhang enthalten) repräsentiert ist, in Ihren eigenen Worten zusammen.

5. Erzeugung von Ontologien mittels OWL

Basierend auf RDF und RDF-S besteht der nächste Schritt in Richtung Semantic Web in der Definition einer Ontologie mittels OWL, die formal die in Web-Dokumenten verwendete Terminologie beschreibt. Anhand von Beispielen werden hier die Struktur und die Konzepte von OWL aufgezeigt.

Die Abkürzung OWL steht für Web Ontology Language. OWL wurde 2001 durch die Ontology Working Group vom W3C erstellt, um eine Strategie zu entwickeln, mithilfe derer die Informationen im Internet besser zugänglich und vor allem für Maschinen einheitlicher zu handhaben sein sollten.

Im Jahre 2004 fand die erste Veröffentlichung von OWL statt. Ein Anspruch, der an OWL gestellt wird ist, dass Informationen aus verteilten Quellen bezogen werden müssen, da das Semantic Web ein verteiltes Netz ist. Der Inhalt von Informationen soll so verarbeitet werden, dass diese dem Anwender nicht nur präsentiert, sondern vorher für ihn aufbereitet werden. Dies geschieht durch Ontologien, Klassen und Objekte, die in Beziehung zueinander dargestellt werden. Benötigt wird eine Standard-Syntax, um Metadaten als solche erkennen zu können. Damit Suchmaschinen und Informationsanbieter dieselben Begriffe nutzen, benötigt man ein oder mehrere standardisierte Vocabulare. Klassen, derer man sich in der OWL bedient, können zwar in anderen Ontologien bearbeitet bzw. erweitert werden, es können aber keine Fakten aus der Klasse entfernt werden.

OWL im Detail

Die vom W3C veröffentlichte Version der OWL-Spezifikation umfasst drei verschiedene Varianten mit unterschiedlichen Ausdrucksstärken:

- OWL Lite: ist OWL in seiner einfachsten Ausprägung. Hier werden einfache Restriktionen und die Bildung von Klassenhierarchien unterstützt. Es dient der Darstellung einfacher Taxonomien. Für OWL Lite können einfache Werkzeuge entwickelt werden. Allerdings ist es damit auch am wenigsten ausdrucksstark.
- OWL DL: beinhaltet OWL Lite und bietet eine erweiterte Ausdrucksstärke. Es genügt den Anforderungen der Description Logics, welche die Logik behandelt, die die formelle Basis von OWL begründet. Aktuelle Softwarewerkzeuge unterstützen OWL DL fast vollständig. Diese Ausprägung von OWL ist entscheidbar, sprich die Verarbeitung wird in endlicher Zeit ausgeführt.
- OWL Full: beinhaltet wiederum OWL DL. Es bedient sich der syntaktischen Freiheit von RDFS und hat volle Ausdrucksstärke. OWL Full ist nicht entscheidbar und gibt somit keine Garantie hinsichtlich der Auswertbarkeit. Eine Software, die

OWL Full vollständig unterstützt ist schwer, wenn nicht gar unmöglich zu realisieren.

Syntax und Semantik

Syntax und Semantik sind notwendig, um eine Ontologie zu schreiben, die nicht mehrdeutig ausgelegt werden kann. Die Semantik ist die Voraussetzung um eine allgemeingültige Verständigung über das Internet zu ermöglichen. OWL ist zunächst Syntax-unabhängig und kann durch unterschiedliche Repräsentationen etwa einer abstrakten Syntax oder auch RDF/XML beschrieben werden. Zur Unterscheidung werden auch in OWL Namespaces genutzt, um Ontologien in XML- Dateien anzulegen.

Selbsttestaufgabe 5.6:

Wofür steht die Abkürzung OWL und worin besteht die generelle Anwendung?

Ontologien

Eine OWL-Ontologie beinhaltet eine Anzahl von Beschreibungen, Klassen, Eigenschaften, Individuen, Axiomen und Fakten. In den Beschreibungen befinden sich z. B. der Name des Autors und Fakten über referenzierte Ontologien. Im Folgenden wird die Syntax für die Ontologie vorgestellt:

```
ontology ::= 'Ontology(' [ ontologyID ] { directive } ')'
directive ::= 'Annotation(' ontologyPropertyID ontologyID
')
| 'Annotation(' annotationPropertyID URIreference ')
| 'Annotation(' annotationPropertyID dataLiteral ')
| 'Annotation(' annotationPropertyID individual ')
| axiom
| fact
```

Beispiel für Annotationen:

```
Ontology(x:meineKleineOntology
Annotation(x:updated 2002-11-25^^xsd:date)
Annotation(owl:incompatibleWith x:meineAndereOntology)
)
```

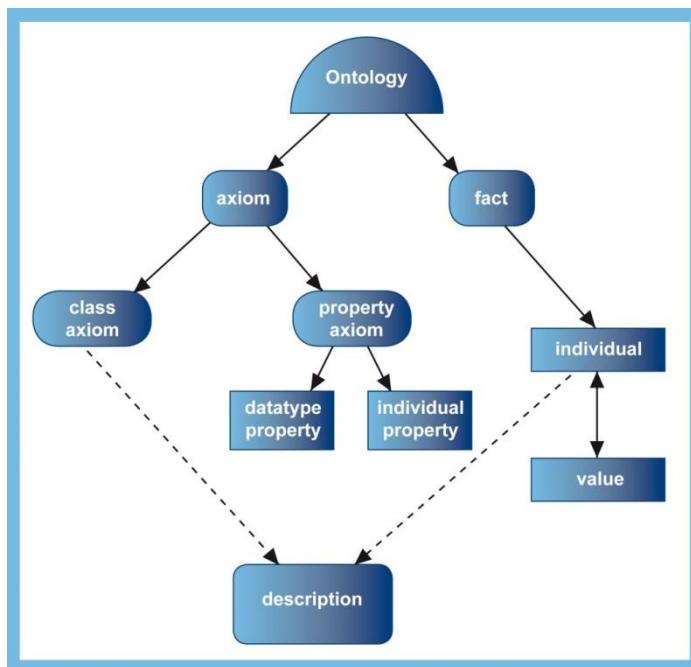


Abbildung 5.1 Die Bestandteile einer Ontologie und deren Relationen zueinander

Die eigentliche Ontologie wird in den Axiomen und Fakten in Form von Klassen, Eigenschaften und Individuen beschrieben:

Klassen bezeichnen eine Menge von Individuen. Vordefinierte Klassen in OWL sind:

- owl: Thing: Oberklasse, beinhaltet alle Klassen,
- owl: Nothing: beinhaltet keine Klassen.

Eigenschaften beziehen sich auf Relationen zu Individuen und anderen Informationen. Individuen sind Instanzen oder Objekte mindestens einer Klasse.

Axiome werden im Allgemeinen benutzt, um Klassen und Eigenschaften Informationen und Charakteristiken zuzuordnen. Unterschieden werden sogenannte Klassenaxiome und Eigenschaftenaxiome.

Axiom

Klassenaxiome

Mit Klassenaxiomen werden Klassen definiert. Hier können verschiedene Relationen festgelegt werden, wie zum Beispiel, dass eine Klasse eine Subklasse einer anderen Klasse ist, dass Klassen zueinander äquivalent sind, dass eine Klasse aus der Zusammensetzung von verschiedenen Superklassen besteht oder durch bestimmte Eigenschaften eingeschränkt wird. Die allgemeine abstrakte Syntax zur Darstellung eines Axioms in OWL Lite ist wie folgt:

```
axiom      ::= 'Class(' classID modality { annotation }
```

```
        { super } ')'
modality ::= 'complete' | 'partial'
super      ::= classID | restriction
```

Die sogenannte Modalität partial wird verwendet, wenn eine Klasse nicht vollständig durch das Axiom beschrieben wird. Ein Beispiel hierfür ist die Klasse Auto, für die nur festgelegt ist, dass sie vier Reifen hat und von einem Motor angetrieben wird. Nicht festgelegt sind hier u. a. die Anzahl der Türen.

Äquivalenz zwischen Klassen wird folgendermaßen definiert:

```
axiom ::= 'EquivalentClasses(' classID classID { classID }
')'
```

Ein Beispiel zur Verwendung sieht folgendermaßen aus:

```
EquivalentClasses (x:PKW x:Auto)
```

Hier wird definiert, dass man mit PKW und Auto die gleichen Mengen von Objekten bezeichnen möchte. Bei OWL DL sieht die Definition von Klassen folgendermaßen aus:

```
axiom      ::= 'Class(' classID  ['Deprecated']
                  modality { annotation } { description } ')'
modality ::= 'complete' | 'partial'
```

In OWL DL werden Klassen durch sogenannte descriptions spezifiziert. Descriptions können wie bei OWL Lite aus Klassen und Restriktionen bestehen. Darüber hinaus lassen sich Descriptions aus einer beliebig verschachtelbaren booleschen Kombination aus mehreren Descriptions gewinnen. Außerdem kann eine Description eine Auswahl von bestimmten Individuen beschreiben.

```
description ::= classID
              | restriction
              | 'unionOf(' { description } ')'
              | 'intersectionOf(' { description } ')'
              | 'complementOf(' description ')'
              | 'oneOf(' { individualID } ')'
```

Hier eine Anwendungsmöglichkeit zur Beschreibung eines Werktages:

```
oneOf(x:Montag x:Dienstag x:Mittwoch x:Donnerstag
      x:Freitag)
```

Im folgenden Beispiel wird die Menger aller weißen Dinge (x:WeißeDinge) mit der Menge aller Weine (x:Wein) geschnitten:

```
intersectionOf(x:WeißeDinge x:Wein)
```

Die Ergebnismenge enthält daher nur Weißweine und keine Rotweine.

```
intersectionOf(complementOf(x:Männer) x:Menschen)
```

Um in diesem Beispiel die Menge aller Frauen zu definieren, wird die Menge aller Menschen mit dem Komplement der Menge aller Männer geschnitten. Die Definition allein über das Komplement der Männer könnte evtl. nicht ausreichen, da diese Menge alle Individuen enthalten würde, welche nicht der Klasse Männer angehören, Männer aber einer weiteren Oberklasse zugehörig sein könnten.

Noch ein Beispiel zur Klassendeklaration in OWL DL:

```
Class(x:Pegasus complete intersectionOf(x:Pferd  
x:Vogel))
```

Die Klasse ‚BremerStadtmusikanten‘ wird im folgenden Beispiel durch die Aufzählung des Esels, des Hundes, der Katze und des Hahns definiert.

```
EnumeratedClass(x:BremerStadtmusikanten x:Esel x:Hund  
x:Katze x:Hahn)
```

Zusätzlich zur Deklaration von äquivalenten Klassen lassen sich hier auch noch disjunkte Klassen deklarieren. Eine Klasse ist dann disjunkt zu einer anderen, wenn sie keine Individuen mit der anderen Klasse gemeinsam hat.

Hier lassen sich statt Klassen auch kompliziertere Konstrukte mithilfe von descriptions in Beziehung setzen. Außerdem kann eine Klasse eine Subklasse einer anderen Klasse sein.

```
axiom ::= 'DisjointClasses(' description description  
{ description } ')'  
| 'EquivalentClasses(' description  
{ description } ')'  
| 'SubClassOf(' description description ')'
```

Ein Beispiel zur Anwendung ist:

```
DisjointClasses(UnionOf(x:GrüneDinge x:Gemüse)  
x:Männer)
```

Mit DisjointClasses werden zwei Klassen definiert, deren Schnittmenge leer ist. Es gibt also keine Dinge in der Ontologie, die gleichzeitig zur Menge der grünen Gemüsesorten gehören und zu der Menge der Männer [RAH03].

```
SubClassOf(x:Hund x:Säugetier)
```

In diesem Beispiel wird eine neue Klasse Hund definiert, die eine Unterklasse der Klasse Säugetier darstellt.

Restriktionen

Eine restriction ist eine Einschränkung einer Eigenschaft. Hierbei wird zwischen den Daten- und Individuen-Eigenschaften unterschieden:

OWL Lite

```
restriction ::= 'restriction(' datavaluedPropertyID data-
taRestrictionComponent ')'
              | 'restriction(' individualvaluedPropertyID
                                individualRestrictionComponent
              ')'
dataRestrictionComponent ::= 'allValuesFrom(' dataRange
')
                           | 'someValuesFrom(' dataRange ')
                           | cardinality
individualRestrictionComponent ::= 'allValuesFrom(' clas-
sID ')
                                    | 'someValuesFrom(' classID
              ')
                           | cardinality
cardinality ::= 'minCardinality(0)' | 'minCardinality(1)'
               | 'maxCardinality(0)' | 'maxCardinality(1)'
               | 'cardinality(0)' | 'cardinality(1)'
dataRange ::= datatypeID | 'rdfs:Literal'
```

dataRanges

Dateneigenschaften werden mithilfe von sogenannten dataRanges eingeschränkt. Diese bestimmen eine Menge von Datenwerten, die sich auf einen bestimmten Daten-Typ beschränken lässt. Als Beispiele lassen sich hier xsd:boolean, xsd:string, xsd:decimal etc. nennen. Das Literal rdfs:Literal kennzeichnet, dass keine Einschränkung erfolgt.

Individuen-Eigenschaften werden mithilfe von Klassen definiert. Die beiden Gruppen der Daten- und Individuen-Eigenschaften lassen sich über folgende Regeln einschränken:


```

        { individualRestrictionCompo-
        nent } ')'
dataRestrictionComponent ::= 'allValuesFrom(' dataRange
')'
| 'someValuesFrom(' dataRange ')'
| 'value(' dataLiteral ')'
| cardinality
individualRestrictionComponent ::= 'allValuesFrom('description ')
| 'someValuesFrom(' descrip-
tion ')
| 'value(' individualID ')'
| cardinality
cardinality ::= 'minCardinality(' non-negative-integer ')'
| 'maxCardinality(' non-negative-integer ')'
| 'cardinality(' non-negative-integer ')'
dataRange ::= datatypeID
| 'rdfs:Literal'
| 'oneOf(' { dataLiteral } ')'

```

Die wesentlichen Unterschiede sind sogenannte IndividualRestriktions, die mithilfe von descriptions miteinander boolsch kombiniert werden sowie einzelne Individuen annehmen. Zusätzlich ist die Kardinalität nicht auf 0 und 1 beschränkt, sondern kann jede nicht negative Integer annehmen.

Eigenschaftsaxiome

Es gibt grundsätzlich zwei verschiedene Arten von Eigenschaften. Die einen spezifizieren Eigenschaften mittels Datenwerten wie z. B. Integer. Die andere Art definiert Eigenschaften in Abhängigkeit von anderen Individuen.

```

Eigenschaften in Abhängigkeit von anderen Individuen.
axiom ::= 'DatatypeProperty(' datavaluedPropertyID
['Depre-cated']
{ annotation }
{ 'super(' datavaluedPropertyID ')'} ['Func-
tional']
{ 'domain(' classID ')'}
{ 'range(' dataRange ')'} ')
| 'ObjectProperty(' individualvaluedPropertyID
['Deprecated'] { annotation }
{ 'super(' individualvaluedPropertyID ')'} 
```

```

        [ 'inverseOf(' individualvaluedPropertyID ')'
]
[ 'Symmetric' ]
[ 'Functional' | 'InverseFunctional' | 
'Functional'
    'InverseFunctional' | 'Transitive' ]
{ 'domain(' classID ')' }
{ 'range(' classID ')' } ')
| 'AnnotationProperty(' annotationPropertyID
    { annotation } ')
| 'OntologyProperty(' ontologyPropertyID { annotation } ')
dataRange ::= datatypeID | 'rdfs:Literal'

```

Mithilfe von Datatype-Properties lassen sich neue Datentypen definieren.

```

DatatypeProperty( x:istPrimzahl super( x:istPositiv )
                    domain( xsd:integer )
                    range ( xsd:boolean ))

```

Die DatatypeProperty istPrimzahlen ist hier definiert als eine Subproperty der Property istPositiv. Des Weiteren besteht der Definitionsbereich nur aus Integern. Eine Domäne gibt an, welche Individuen potentielle Kandidaten für hier definierte Property mit eben dieser Eigenschaft sind.

In OWL-Lite sind Property-Domänen Klassen.

Eine Range definiert, welche Individuen oder Datenwerte Objekte der spezifizierten Property sein können.

Range

Die Eigenschaft der Primzahl ist eigentlich noch gar nicht definiert, so wie wir sie verstehen. Solch zusätzliche semantische Information kann über die Annotation-Properties hinzugefügt werden. Ein Beispiel für eine Object-Property ist im Folgenden gegeben:

```
ObjectProperty(x:istElternteil inverseOf(x:istKind))
```

Hier wird die Eigenschaft istElternteil definiert. Sie ist invers zu der Eigenschaft istKind. Das bedeutet, wenn Klara istKind von Tom gegeben ist, dann folgt für Tom umgekehrt istElternteil von Klara.

```

axiom ::= 'EquivalentProperties(' datavaluedPropertyID
                    datavaluedPropertyID
                    {datavaluedPropertyID} ')

```

```

| 'SubPropertyOf(' datavaluedPropertyID
    datavaluedPropertyID ')'
| 'EquivalentProperties(' individualvaluedPropertyID
    individualvaluedPropertyID
    { individualvaluedPropertyID } ')'
| 'SubPropertyOf(' individualvaluedPropertyID
    individualvaluedPropertyID ')'

```

Durch die Definition von Subproperties besteht die Möglichkeit Properties hierarchisch anzugeordnen, wie das folgende Beispiel verdeutlicht:

```

SubPropertyOf (x:istSohnVon x:istKindVon)
,istSohnVon' wird definiert als eine Untereigenschaft ist
von ,istKindVon'.

```

Des Weiteren kann eine Property auch als äquivalent zu einer anderen Property definiert werden:

```
EquivalentProperties(x:istOmaVon x:istGroßmutterVon)
```

istOmaVon und istGroßmutterVon beinhalten die gleiche Menge an Objekt-Tupeln.

Im Vergleich zu den bisher behandelten Eigenschaftsaxiomen von OWL Lite werden die Eigenschaftsaxiome bei OWL DL wie folgt spezifiziert:

```

axiom    ::=    'DatatypeProperty('      datavaluedPropertyID
['Depre-cated']
            { annotation }
            { 'super(' datavaluedPropertyID ')'} ['Func-
tional']
            { 'domain(' description ')'}
            { 'range(' dataRange ')'} ')
| 'ObjectProperty(' individualvaluedPropertyID
['Deprecated'] { annotation }
            { 'super(' individualvaluedPropertyID ')'}
            [ 'inverseOf(' individualvaluedPropertyID ')'
]
            [ 'Symmetric' ]
            [ 'Functional' | 'InverseFunctional' | 'Func-
tional'
            'InverseFunctional' | 'Transitive' ]
            { 'domain(' description ')'}

```

```

        { 'range(' description ')' } ')'
| 'AnnotationProperty(' annotationPropertyID
    { annotation } ')
| 'OntologyProperty(' ontologyPropertyID { annota-
    tion } ')

```

Die Eigenschaften sind fast genau wie OWL Lite Property Axiome, nur erlauben sie zusätzlich `description` an der Stelle von Klassen und zusätzlich Aufzählungen zu den Datentypen. Subproperties und Äquivalenzbedingungen sind wie in OWL-Lite definiert.

```

ObjectProperty( x:istEhefrauVon
    super(x:istEhepartnerVon)
    inverseOf(x:istEhemannVon)
    Functional InverseFunctional
    domain(x:Frau)
    domain(x:NichtLedigerMensch)
    domain(x:MonogamerMensch)
    range(x:Mann)
    range(x:NichtLedigerMensch)
    range(z:MonogamerMensch)
)

```

Hier wird die ObjectProperty mit `istEhefrauVon` definiert. Diese ist abgeleitet von der Eigenschaft `istEhepartnerVon`. Außerdem ist sie invers zur Eigenschaft `istEhemannVon`. Diese beiden sind funktional, d. h. eindeutig bzw. invers funktional. Die hier verwendete `domain` ist eine Kombination aus `Frau` und `NichtLedigerMensch`. Die Menge, auf die diese Eigenschaft abgebildet wird, ist durch eine konjunktive Kombination aus mehreren Werten durch `range` gegeben.

Eindeutigkeit und inverse Funktionalität

OWL und Facts

Wir unterscheiden bei der abstrakten OWL-Syntax zwei verschiedene Arten von Facts: Die erste Art sagt etwas über ein bestimmtes Individuum aus, zu welcher Klasse es gehört und welche Eigenschaften und Werte es besitzt:

```

fact ::= individual
individual ::= 'Individual(' [ individualID ] { annotation
}
        { 'type(' type ')' } { value } ')
value ::= 'value(' individualvaluedPropertyID individualID
')
        | 'value(' individualvaluedPropertyID individual
')
        | 'value(' datavaluedPropertyID dataLiteral ')

```

Zunächst einige Beispiele zu value:

```
value( x:Alter "30"^^xsd:integer )
```

Hier wurde ein Wert mithilfe eines Datenliterals definiert.

```
value( x:verheiratetMit x:Klara )
```

Definition eines Wertes anhand eines Individuums und zu guter Letzt eine Wertdefinition unter Verwendung einer Klasse:

```
value( x:magGerne x:Apfel )
Mithilfe dieser Möglichkeiten wollen wir nun ein Individuum definieren:
Individual(x:Hans type(x:Mann)
            value( x:Alter "30"^^xsd:integer)
            value( x:verheiratetMit x:Klara )
            value( x:magGerne x:Apfel )
)
```

Definiert wird hier das Individuum Hans, das vom Typ Mann und 30 Jahre alt ist. Er ist mit Klara verheiratet und mag gerne Äpfel. Dies funktioniert, da verschiedene values miteinander kombiniert werden können.

IndividualID

Die *IndividualID* gibt dem Individuum einen Namen. Das Individuum muss aber keinen Namen erhalten und kann auch anonym bezeichnet werden. Auf anonyme Individuen kann in der Ontologie nicht referenziert werden.

Facts sind in der abstrakten Syntax von OWL-DL und OWL-Lite gleich angelegt. Der einzige Unterschied besteht darin, dass in OWL-Lite KlassenIDs und Beschränkungen Typen sein können. In OWL-DL hingegen sind es Beschreibungen, in welchen die Typen von OWL-Lite neben anderen enthalten sind.

OWL-Lite:

```
type ::= classID
       | restriction
```

OWL-DL:

```
type ::= description
```

Die zweite Art fact orientiert sich an schon vorhandenen Individuen:

```
fact ::= 'SameIndividual(' individualID individualID  
           {individualID} ')'  
       | 'DifferentIndividuals(' individualID individualID  
           {individualID} ')'
```

Hier ein Beispiel:

```
SameIndividual(Klara KlaraMüller Klärchen)
```

Klara ist ein Individuum, das auch unter dem Namen KlaraMüller bekannt ist und bei guten Freunden auch Klärchen genannt wird. Klara, KlaraMüller und Klärchen bezeichnen also ein und dasselbe Individuum. Anders verhält es sich bei

```
DifferentIndividuals(Klara Fritz)
```

Mit Klara und Fritz werden zwei verschiedene Individuen bezeichnet.

Datenliterale bestehen in der abstrakten Syntax entweder aus Unicode Strings in der normalen C Form mit plain literals oder einer URI-Referenz mit typed literals.

```
dataLiteral ::= typedLiteral | plainLiteral  
typedLiteral ::= lexicalForm^^URIreference  
plainLiteral ::= lexicalForm | lexicalForm@languageTag  
lexicalForm ::= as in RDF, a unicode string in normal  
               form C  
languageTag ::= as in RDF, an XML language tag
```

6. Metadaten

Der Begriff Metadaten ist relativ neu, wobei das Prinzip an sich u. a. bereits eine jahrhundertelange bibliothekarische Praxis darstellt. Laut ISO 11179 handelt es sich bei Metadaten um eine Dokumentation von Daten, die es ermöglicht, Wissen aus Daten zu verstehen und die es erlaubt, diese mit anderen Benutzern auszutauschen. HTML bietet zur Notation von Metadaten das Sprachkonstrukt der Meta-Tags an. Diese können verschiedene Anmerkungen aufnehmen, welche beispielsweise die Suche durch eine Suchmaschine oder die Interpretation durch einen Web-Browser erleichtert.

Rolle der Metadaten

In Anbetracht der unterschiedlichen Notationen von Metadaten trafen Kashyap und Sheth [BBB97] folgende Unterscheidung:

- Inhaltsunabhängige Metadaten sind Daten, die nicht direkt zum informationellen Inhalt von Daten in Beziehung stehen. Sie beschreiben vielmehr den Kontext und die Umgebung, in der diese eingebettet sind wie Autor des Dokuments und Erzeugungsdatum.
- Inhaltsabhängige Metadaten sind Daten über Informationen, die direkt vom Inhalt der Daten abgeleitet sind, jedoch nicht den Inhalt beschreiben, wie Größe des Dokuments oder Anzahl der Wörter.
- Auf dem Inhalt basierende Metadaten geben direkt den Inhalt von Daten wieder, fügen jedoch Information, Wissen oder sonstige Strukturen an, die die Verarbeitung der Originaldaten effizienter gestalten sollen wie z. B. Dokumentvektoren oder Volltextindizes.
- Inhaltsbeschreibende Metadaten sind Daten, die auf dem Inhalt der ursprünglichen Daten beruhen, die aber nicht direkt aus ihnen hervorgehen. Sie geben eine Zusammenfassung des Inhalts und helfen z. B. Entscheidungen zu treffen, ob eine Informationsquelle passend für eine bestimmte Aufgabe ist wie eine Schlüsselwortliste oder ein Glossar.

Die folgende Abbildung veranschaulicht den Zusammenhang:

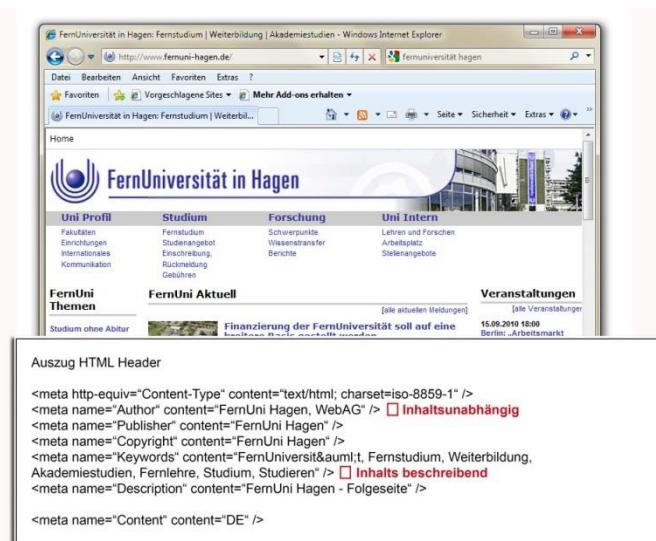


Abbildung 6.1 Auszug der Metainformation der Webseiten der FernUniversität in Hagen

Die unterschiedlichen Arten von Metadaten decken mehrere Aspekte von Informations- und Wissensbeschreibungen ab. Es handelt sich um technische Daten wie etwa Daten

über Speicherung und Zugangsmethoden, um Inhaltsbeschreibungen, Gebrauch, Eignung und Datenqualität. Bezuglich des Auffindens und des Zugangs zu Information und Wissen stellt sich die Rolle der Metadaten wie folgt dar:

- Wissensanbietern ermöglichen sie die Organisation, Instandhaltung und Katalogisierung ihrer Daten.
- Informationsnutzern helfen sie beim Auffinden, Zugang und der Interpretation von Wissen.

Gebrauch

Metadaten können in den verschiedensten Gebieten zur Anwendung kommen: etwa bei der Suche von Daten, beim Zugriff auf Daten sowie bei der Interpretation von Daten. Diese drei Gebiete werden im Folgenden erläutert.

- **Suche:** Durch den Gebrauch von Themenbeschreibungen, Schlüsselwortlisten und Inhaltszusammenfassungen können Metadaten dazu dienen, Information und Wissen im Web zu lokalisieren, ohne den Inhalt der entsprechenden Seite genauer zu untersuchen.
- **Zugriff:** Metadaten bezüglich technischer Eigenschaften einer Informationsquelle wie etwa Formatierung und Codierung können den Aufwand zur Verarbeitung erheblich reduzieren.
- **Interpretation:** Eine Applikation benötigt nicht nur den Zugriff auf Information und Wissen, sondern muss auch in der Lage sein, diese zu interpretieren. Informationen über die enthaltene Terminologie und erforderliches Wissen, um den Inhalt zu interpretieren, ermöglichen es sowohl Benutzenden als auch intelligenten Systemen den eigentlichen Inhalt der Informationsquelle besser zu verstehen.

Offensichtlich ist der Gebrauch von Metadaten enorm wichtig, um den Umgang mit Information und Wissen in heterogenen Umgebungen wie dem WWW zu unterstützen.

Information und Wissen in heterogenen Umgebungen

Management

Oben getroffene Überlegungen klären die Notwendigkeit von Metadaten speziell für Web-basierte Informationssysteme. Mit der Absicht unterschiedliche Aspekte von Metadaten anzusprechen wurden Standards eingerichtet; beispielsweise über die Syntax der Kodierung, der Modell-Struktur und des Inhalts eines Metadaten-Modells. Einige dieser Standards sind:

- **Syntaktische Standards:** Eigenschaften zum Codieren von Metadaten sind

RDF

schon im allgemeinen HTML enthalten. Meta-Tags können genutzt werden um Attribute und korrespondierende Werte eines HTML-Dokuments zu spezifizieren. Mittels RDF wird die kodierte Syntax definiert, um einen Web-Browser zu befähigen, die Metadaten zu verarbeiten.

- Topic Maps
- Strukturelle Standards: Um die Entwicklung von Metadatenmodellen zu unterstützen, stellt die Standardisierung von Modellen einen wichtigen Aspekt dar. Strukturelle Standards wurden oberhalb existierender syntaktischer Standards definiert. RDF-Schema beispielsweise definiert eine Modellstruktur wie sie auch in frame-basierten Wissensrepräsentationssystemen zur Anwendung kommt. Des Weiteren stellen Topic Maps eine zusätzliche Methode zur Beschreibung von Ressourcen dar.
- Inhaltsbezogene Standards: Während RDF-Schema und Topic Maps auf struktureller Ebene Elemente strukturieren um Metadaten zu repräsentieren, ermöglichen sie es nicht, Information bezüglich der Datenhaltung, Organisation und Nutzung zu beschreiben. Einer der wichtigsten inhalts-bezogenen Standards ist der sogenannte Dublin Core-Standard, welcher einen Satz von Metadatenelementen für Dokumente vorgibt.

Dublin Core

Die oben angeführten Standards bieten eine gute Anleitung zum Design und zur Kodierung von Metadaten für Informations- und Wissensressourcen im WWW. Darüber hinaus treten jedoch noch weitere Herausforderungen auf, die weder strukturelle noch inhaltliche Aspekte der Kodierung von Wissen und Information betreffen. Diese Herausforderungen betreffen die Beziehungen zwischen Information, Wissen und deren Metadaten. Einige der wichtigsten sind:

- Vollständigkeit: Um einen vollen Zugriff gewährleisten zu können, muss gesichert werden, dass die Wissensressource mit allen informationell relevanten Metadaten annotiert ist. Andernfalls geht Suchmaschinen, die Metadaten orientiert vorgehen, wichtige Information verloren.
- Konsistenz: Metadaten sind nur brauchbar, wenn sie den Inhalt der Wissensressource korrekt beschreiben. Tatsächlich sind inkonsistente Metadaten ein viel größeres Problem als fehlende Metadaten. Eine typische Inkonsistenz ist z. B. der Fall, bei dem innerhalb der Metadaten ein bestimmtes Schlüsselwort erwähnt wird (z. B. Wasserspiegel), sich aber der Inhalt der Wissensressource um etwas anderes dreht (z. B. Energieschutz).

- Erreichbarkeit: Damit Metadaten wirklich nutzbar sind, müssen sie nicht nur für den Wissensanbieter erreichbar sein, sondern auch für die Benutzer, die diese als Information anwenden wollen.

Metadaten in RDF und RDF-Schema

Das Resource Description Framework ist ein Rahmenwerk, das es ermöglicht, strukturierte Metadaten zu kodieren und auszutauschen. Es basiert auf XML und erweitert es um die Fähigkeit semantische Aussagen über eine Ressource auszudrücken. RDF basiert auf einem Grafenmodell, das die Semantik der Metadaten ausdrückt.

Das Basiskonzept von RDF besteht darin, dass eine Ressource durch eine Menge von RDF-Eigenschaften, die RDF- Beschreibung genannt werden, beschrieben wird. Jede dieser RDF-properties hat dabei einen eindeutigen Typ und einen Wert. Im Gegensatz zu Metadatenbeschreibungen, die nur auf XML basieren, können durch dieses Modell Aussagen über andere Ressourcen getroffen werden. Der Grundgedanke von RDF ist also ein Modell zur Darstellung von Eigenschaften und Eigenschaftswerten. Das RDF-Modell stützt sich auf Prinzipien zur Darstellung von Ähnlichkeiten und Beziehungen. RDF-Eigenschaften können als Attribute von Ressourcen betrachtet werden und entsprechen den traditionellen Attributpaaren [BCV99]. RDF repräsentiert Verbindungen und Beziehungen zwischen Ressourcen, weshalb es auch einem Verbindungsdiagramm ähnelt. Im objektorientierten Design würden Ressourcen den Objektklassen und die Ressourceneigenschaften den Instanzen entsprechen. Das RDF-Datenmodell ist ein syntaxneutraler Weg zur Darstellung von RDF-Ausdrücken und -Aussagen. Im Gegensatz zu XML, dass nur die Struktur der Metadaten beschreibt, können durch die Kodierung mit RDF Anfragen der Art: „Über was wird eine Aussage getroffen?“ und „Wer trifft eine Aussage“ gestellt werden. Es ist mit RDF aber ebenfalls nicht möglich, Fragen der Art „Was wird gesagt?“ auszudrücken. Zur Beantwortung dieser Frage ist auch bei RDF die Einigung auf ein Vokabular und auf die Bedeutung der einzelnen Elemente notwendig. Aus diesem Grund wurden vom W3C RDF-Schema entwickelt [SSR94]. Durch RDF-Schema wird ein minimales Basisvokabular für die Bedeutung einer Aussage und für die Bedeutung von Beziehungen zwischen Metadatenbeschreibungen eingeführt.

Metadaten in Dublin Core

Das Metadatenschema von Dublin Core ist ein weit verbreitetes Schema zur Beschreibung Web-basierter Ressourcen. Es wurde im Jahre 1997 von einer Gruppe von Informationswissenschaftlern, Bibliothekaren und Dokumentären in Dublin (Ohio, USA) erarbeitet [BBB97]. Der Dublin Core ist vollständig textorientiert und daher sowohl von Menschen, als auch von automatisierten Systemen interpretierbar. Der Dublin Core ist ein besonders im Bibliothekswesen verbreitetes Schema. Folgende Eigenschaften charakterisieren den Dublin Core und zeichnen ihn als zukünftigen Standard für die Beschreibung elektronischer Ressourcen aus:

- Einfachheit: Dieses Schema ist sowohl für Anfänger als auch für Experten leicht zu benutzen, da die Syntax in fast allen Elementen allgemein verständlich und überschaubar ist.
- Semantische Interoperabilität: Um eine qualifizierte Suche im Internet zu ermöglichen, sollten sich Metadaten zwischen verschiedenen Fachgebieten semantisch entsprechen.
- Erweiterbarkeit: Dublin Core kann mithilfe von verschiedenen detaillierten Beschreibungsmodellen erweitert werden.
- Modularität im Web: Dublin Core unterstützt eine Architektur von Metadaten wie das bereits genannte RDF, das im Rahmen des W3Cs (World Wide Web Consortium) entwickelt wurde.

Bei Dublin Core sind 15 Elemente vorgesehen, die als allgemein verständliche Bezeichner von Meta-Tags verwendet werden können ([LWGG06], [B06]):

- DC.title (Titel) gibt den Namen der WWW-Seite an, wobei DC.title nicht mit der Angabe z. B. im Title-Tag einer HTML-Seite übereinstimmen muss.
- DC.creator (Urheber) gibt den Autor der Seite an, also die Person, die für den Inhalt verantwortlich ist.
- DC.subject (Schlüsselwörter) nennt alle Themen der Publikation. Hierbei sollten Schlagwörter verwendet werden, die den Inhalt am besten wiedergeben.
- DC.description (Beschreibung) dient zur Beschreibung bzw. als Zusammenfassung des Inhalts.
- DC.publisher (Verlag) bezeichnet die Person oder Organisation, die für die Publikation in der vorliegenden Form verantwortlich ist (Web-Administrator, Abteilung einer Firma oder ein Verlag).
- DC.contributor (Mitwirkende) zählt die Personen oder Organisationen auf, die einen nennenswerten Beitrag zu der Veröffentlichung geleistet haben, jedoch unter DC.creator bzw. DC.publisher nicht erwähnt wurden.
- DC.date (Datum) gibt das Datum der Publikation in der vorliegenden Form an. Dabei empfiehlt sich die normierte Darstellung JJJJ-MM-TT, also zunächst die vierstellige Jahreszahl, gefolgt von den zweistelligen Zahlen für Monat und Tag.
- DC.type (Art der Quelle) beinhaltet Terme, die die generellen Kategorien, Funktionen und Genres beschreiben.

- DC.format (Format) enthält normalerweise den Medientyp. Format kann benutzt werden, um die Software, Hardware oder andere Ausrüstung zu beschreiben, die notwendig ist, um die Ressource darzustellen oder zu verarbeiten.
- DC.identifier (Bezeichnung) kennzeichnet das Dokument eindeutig beispielsweise mittels einer ISBN, wenn es sich um die elektronische Variante eines Buches handelt oder die ISSN einer digitalen Zeitschrift.
- DC.source (Quelle) dient zur Angabe der Quelle des Werkes, also z. B. Titel, Verlag und ISBN der entsprechenden gedruckten Version.
- DC.language (Sprache) bezeichnet die verwendete natürliche Sprache.
- DC.relation (Verhältnis) kennzeichnet das Verhältnis des Dokumentes zu anderen Ressourcen (z. B. Bilder in einem Dokument, Buch etc.).
- DC.coverage (erfasste Menge) wird typischerweise den Aufenthaltsort, die Zeitperiode oder die Jurisdiktion beinhalten.
- DC.rights (Rechte) soll Informationen über Rechte in und über die Ressource bereitstellen oder auf einen Dienst verweisen, der solche Informationen zur Verfügung stellt. Informationen über Rechte beinhalten oft Urheberrechte und das Copyright. Fehlt das DC.right-Element, können keine Aussagen über den Status von Rechten in Bezug auf das Dokument gemacht werden.

Da die 15 Elemente bei vielen Anwendungen nicht zur Beschreibung ausreichen, gibt es die Möglichkeit, das Schema zu erweitern, um die Struktur und ausführlichere Semantik zu kodieren.

Im folgenden Listing wird die Darstellung aus der vorhergehenden mithilfe des Dublin Core-Standards erweitert.

```
<link rel="schema.DC"
      href="http://purl.org/dc/elements/1.1/">
<meta name="DC.format" content="text/html" charset="iso-8859-1" >
<meta name="DC.Titel" content ="FernUniversit&auml;t in Ha-gen: Fernstudium | Weiterbildung | Akademiestudien" />
<meta name="DC.Creator" content="FernUni Hagen, WebAG" />
<meta name="DC.Publisher" content="FernUni Hagen" />
<meta name="DC.Copyright" content="FernUni Hagen" />
```

```
<meta name="DC.Subject" content="FernUniversit&auml;t,  
Fern-studium, Weiterbildung, Akademiestudien, Fernlehre,  
Studium, Studieren" />  
<meta name="DC.Description" content="FernUni Hagen - Fol-  
geseite" />  
<meta name="DC.Language" content="de" />
```

fully qualified Dublin Core

Das Basis-Schema wurde weiterhin um eine Variante erweitert, die fully qualified Dublin Core genannt wird. In diesem Schema werden alle 15 Elemente weiter verfeinert und zusätzlich wird für bestimmte Elemente ein festes Vokabular als möglicher Wert vorgegeben.

Selbsttestaufgabe 5.7:

Was ist die Rolle von Metadaten in (beispielsweise) einem HTML Dokument?

Selbsttestaufgabe 5.8:

Was ist der sogenannte Dublin Core?

7. Anhang

Instrumente.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"      xmlns:rdfs="http://www.w3.org/2000/01/rdf-
schema#"          xmlns:dc="http://purl.org/dc/terms/"
xmlns:ns0="musik:instrumente;">
<rdf:Description rdf:about="musik:instrumente">
<rdfs:type
rdf:resource="http://www.w3.org/2002/07/owl#owl"/>
<rdfs:type
rdf:resource="http://www.w3.org/2000/01/rdfschema#      Re-
source"/>
<rdfs:type      rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
<dc:title>Instrumente</dc:title>
<rdfs:label>Instrumente (Resource)</rdfs:label>
<rdfs:label>Instrumente (Class)</rdfs:label>
<rdfs:comment>Alle Arten von Instrumenten (Re-
source)</rdfs:comment>
<rdfs:comment>Alle Arten von Instrumenten (Class -- All
classes are resources, too)</rdfs:comment>
<rdfs:isDefinedBy rdf:resource="musik:instrumente"/>
</rdf:Description>
<rdf:Description rdf:about="musik:instrumente:name">
<rdfs:type
rdf:resource="http://www.w3.org/2000/01/rdfschema# Property"/>
<rdfs:domain rdf:resource="musik:instrumente"/>
<rdfs:range rdf:resource="musik:instrumente"/>
<rdfs:subPropertyOf rdf:resource="musik:instrumente"/>
</rdf:Description>
<rdf:Description
rdf:about="musik:instrumente:holzblasinstrumente">
<rdfs:type      rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
<rdfs:isDefinedBy
rdf:resource="musik:instrumente:holzblasinstrumente"/>
```

```
<rdfs:label>Holzblasinstrumente</rdfs:label>
<rdfs:comment>Gattung Holzblasinstrumente</rdfs:comment>
<rdfs:subClassOf rdf:resource="musik:instrumente"/>
</rdf:Description>
<rdf:Description
rdf:about="musik:instrumente:holzblasinstrumente:klarinette">
<rdfs:type    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<rdfs:isDefinedBy
rdf:resource="musik:instrumente:holzblasinstrumente:klarinette"/>
<rdfs:label>Klarinette (label)</rdfs:label>
<rdfs:subClassOf
rdf:resource="musik:instrumente:holzblasinstrumente"/>
<ns0:name>Klarinette (nested property)</ns0:name>
</rdf:Description>
<rdf:Description
rdf:about="musik:instrumente:blechblasinstrumente">
<rdfs:type    rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
<rdfs:isDefinedBy
rdf:resource="musik:instrumente:blechblasinstrumente"/>
<rdfs:label>Blechblasinstrumente</rdfs:label>
<rdfs:comment>Gattung Blechblasinstrumente</rdfs:comment>
<rdfs:subClassOf rdf:resource="musik:instrumente"/>
</rdf:Description>
</rdf:RDF>
```

8. Literaturverzeichnis

- [B06] Petrie, Bussler-Industrial Semantics and Magic.In IEEE INTERNET COMPUTING: 96-98, 07/08 2006
- [BHL01] Berners-Lee, Hendler, Lassila The Semantic Web. Scientific American Magazine: 35-44, May 2001
- [BBB97] Bayardo ,. Bohrer, Brice, Cichocki, Fowler, Helal, Kashyap,. Ksiezyk, Martin, Nodine, Rashid, Rusinkiewicz, Shea, Unnikrishnan, Unruh, and WoelkInfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments.In ACM 0-89791-911 - 4/97 /0005: 195-206, 1997
- [BCV99] Bergamaschi, Castano, Vincini Semantic Integration of Semistructured and Structured Data Sources. In SIGMOD Record, 28(1): 54-59, March 1999
- [AvH04] Antoniou,G. van Harmelen,F. *A Semantic Web Primer*. The MIT Press, 2004
- [DMHF00] Decker, Melnik, van Harmelen, Fensel,. Klein, Broekstra, Erdmann, Horrocks The Semantic Web: The Roles of XML and RDF In IEEE INTERNET COMPUTING: 63-74, 09/10 2000.
- [LWGG06] Lemmens,. Wytzisk, de By, Granell, Gould, van OosteromIntegrating Semantic and Syntactic Descriptions to Chain Geographic Services. In IEEE INTERNET COMPUTING: 42-52, 09/10 2006
- [M01] Harold., Means. XML in a Nutshell, Dt. Ausgabe. O'Reilly Verlag, Köln, 2001.
- [SvH05] Stuckenschmidt, van Harmelen Information Sharing on the Semantic Web Springer-Verlag, Berlin Heidelberg New York, 2005
- [SSR94] Sciore, M. Siegel, A. Rosenthal Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems.In ACM Transactions on Database Systems, 19(2): 254-290, June 1994
- [W3C1] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/xpath-datatype/>.
- [W3C2] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Functions and Operators W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/xpath-functions/>.
- [W3C4] World Wide Web Consortium: XML Path Language XPath 1.0. W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/1999/REC-xpath-10>

19991116.

- [W3C5] World Wide Web Consortium: XML Path Language (XPath) 2.0. W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/2005/CR-xpath20-20051103/>
- [W3C6] World Wide Web Consortium: XSL Transformations (XSLT) Version 2.0. W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/2005/CR-xslt20-20051103/>
- [W3C7] World Wide Web Consortium: XQuery 1.0: An XML Query Language. W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/2005/CR-xquery-20051103/>
- [MM09] Mochales, Raquel & Moens, Marie-Francine. (2009). Argumentation mining: The detection, classification and structure of arguments in text. Proceedings of the International Conference on Artificial Intelligence and Law. 98-107. 10.1145/1568234.1568246.
- [Kim17] Hyun Jeong Kim et al. "Clinical Utility of Measurement of Vitamin D-Binding Protein and Calculation of Bioavailable Vitamin D in Assessment of Vitamin D Status". eng. In: Annals of Laboratory Medicine 37.1 (Jan. 2017), pp. 34–38. ISSN: 2234-3814. DOI: 10.3343/alm.2017.37.1.34.
- [Naw21] Christian Nawroth, Supporting Information Retrieval of Emerging Knowledge and Argumentation, Dissertation, FernUniversität in Hagen, eingereicht zur Begutachtung und Veröffentlichung, 2021
- [MVH04] Deborah L. McGuinness and Frank Van Harmelen. "OWL web ontology language overview". In: (2004).
- [NDH21] Christian Nawroth, Alexander Duttenhöfer, and Matthias Hemmje. "Argumentationsunterstützung durch emergentes Wissen in der Medizin". In: Wilhelm Bauer, Joachim Warschat, Innovation durch Natural Language Processing - Mit Künstlicher Intelligenz die Wettbewerbsfähigkeit verbessern. Carl Hanser Verlag GmbH, 2021. ISBN: 978-3446462625.
- [RAH03] E. Rahm, G. Vossen (Hrsg.) Web & Datenbanken. dpunkt.verlag, 2003
- [HOD01] Hodgson, J.: *Do HTML Tags Flag Semantic Content*. In: IEEE INTERNET COMPUTING: 20-25, 01/02 2001.

9. Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 3.1 Semantic Web Layer Cake..... | 9 |
| Abbildung 3.2 RDF Annotation | 10 |
| Abbildung 3.3 Semantische Ambiguität | 11 |
| Abbildung 3.4 RDF Annotation 2 | 11 |
| Abbildung 3.5 Beispielhafter RDF-Graf und XML-Notation..... | 12 |
| Abbildung 3.6 Aussage über Aussage..... | 15 |
| Abbildung 3.7 Reifikation | 16 |
| Abbildung 3.8 Reifikation | 17 |
| Abbildung 3.9 AIF Triple [NDH21], Text: [Kim17] | 19 |
| Abbildung 3.10 Einfaches Beispiel einer Argumentationsbaumstruktur [MM09]..... | 21 |
| Abbildung 4.1 Taxonomie des Lebens, Ausschnitt | 28 |
| Abbildung 4.2 Instanz des Jaguars franz..... | 31 |
| Abbildung 4.3 Instanz des Jaguars franz (Bild) | 33 |
| Abbildung 5.1 Die Bestandteile einer Ontologie und deren Relationen zueinander..... | 37 |
| Abbildung 6.1 Auszug der Metainformation der Webseiten der FernUniversität in Hagen..... | 48 |

10. Lösungen der Selbsttestaufgaben

Selbsttestaufgabe 5.1

Was ist ein semantisches Netz und wozu wird es verwendet?

Ein Semantisches Netz (semantic web) hilft dabei, die im Web verfügbaren Inhalte maschinell verarbeiten zu können. Dabei wird der semantische Hintergrund der dargestellten Informationen interpretiert und Beziehungen zwischen den interpretierten Informationen vom System erkannt und automatisch maschinell verarbeitet. Dies soll den Benutzern helfen, die richtigen bzw. relevanten Informationen zu finden.

Selbsttestaufgabe 5.2

Welches sind die Bestandteile eines RDF-Tripels? Nennen Sie zwei mögliche Darstellungsformen

Ressource/Subjekt, Prädikat, Objekt. Als Graf oder als XML-Notation

Selbsttestaufgabe 5.3

Erläutern Sie das Prinzip der Reifikation anhand des folgenden N-Triples:

N-Triple:

```
<sh:SherlockHolmes> <s:supposes>
<s:StatementOfSherlock> .
<s:StatementOfSherlock> <http://www.w3.org/1999/02/22-rdf-syntax-ns#a>
<rdf:Statement> .
<s:StatementOfSherlock>
<http://www.w3.org/1999/02/22-rdf-syntaxns# subject> <s:Gardener> .
<s:StatementOfSherlock>
<http://www.w3.org/1999/02/22-rdf-syntaxns# predicate> <s:hasKilled> .
<s:StatementOfSherlock>
<http://www.w3.org/1999/02/22-rdf-syntaxns# object>
<s:Butler> .
```

Das Prinzip der Reifikation ermöglicht es, in RDF Aussagen über Aussagen zu treffen.

Im Beispiel ist die „innere“ Aussage, dass der Gärtner den Butler getötet hat. Ergänzt wird dies durch die zusätzliche Aussage, dass Sherlock Holmes diesen Sachverhalt (lediglich) vermutet.

Selbsttestaufgabe 5.4

Woraus besteht ein Argument im Sinne des Argument Interchange Format (AIF)? Was sind die Aufgaben der sogenannten S-Knoten, bzw. I-Knoten?

Ein Argument besteht aus einer oder mehreren Prämissen und einer Schlussfolgerung.

Die I-Knoten enthalten dabei aussagenbezogene Informationen wie eine Behauptung oder ermittelte Daten, die S-Knoten erfassen die Anwendung von Schemata (Argumentation, Inferenz, ...).

Ein Spezialfall sind dabei die RA-Knoten, die eine Regel der Inferenzanwendung darstellen.

Selbsttestaufgabe 5.5

Aufgabe zu Klassen / Klassenhierarchie:

Fassen Sie die Klassenhierarchie, die im XML-Dokument „Instrumente“ repräsentiert ist, in Ihren eigenen Worten zusammen.

Das Dokument bildet die Beziehung zwischen verschiedenen Musikinstrumenten ab. Hierbei ist die Klasse Instrument übergeordnet, Holzblasinstrumente und Blechblasinstrumente sind jeweils untergeordnete Klassen davon (rdfs: subClassOf). Als ein Beispiel der Klasse Holzblasinstrumente ist das Instrument Klarinette modelliert.

Selbsttestaufgabe 5.6

Wofür steht die Abkürzung OWL und worin besteht die generelle Anwendung?

Web Ontology Language, Zweck: Semantische Informationen im Internet besser zugänglich und vor allem für Maschinen verarbeitbar / interpretierbar zu machen.

Selbsttestaufgabe 5.7

Was ist die Rolle von Metadaten in (beispielsweise) einem HTML Dokument?

Suche durch eine Suchmaschine oder die Interpretation durch einen Web-Browser werden erleichtert. Beispiele:

- Kontext und die Umgebung, in der diese eingebettet sind wie Autor des Dokuments und Erzeugungsdatum.
- Größe des Dokuments oder Anzahl der Wörter.
- Zusammenfassung des Inhalts hilft z. B. Entscheidungen zu treffen, ob eine Informationsquelle passend für eine bestimmte Aufgabe ist

Selbsttestaufgabe 5.8

Was ist der sogenannte Dublin Core?

Das Metadatenschema von Dublin Core ist ein weit verbreitetes standardisiertes Schema zur Beschreibung Web- basierter Ressourcen.

Ziel ist die leichte Auffindbarkeit von Objekten durch Metainformationen

- 15 Kernfelder (Core Elements) sind vorgesehen, die als allgemein verständliche Bezeichner von Meta-Tags verwendet werden
- Bsp.: date, type, format, subject
- Die Felder sind optional, können mehrmals und in belieber Reihenfolge auftauchen
- Felder (Simple Dublin Core): identifier, format, type, language, title, subject, coverage, description, creator, publisher, contributor, rights, source, relation, date

000 000 000 (00/23)

00000-0-00-S 1

Prof. Dr.-Ing. Matthias L. Hemmje

01877

Dokumenten- und Wissensmanagement im Internet

Kurseinheit 4:
Einführung in XSLT und XQuery

Fakultät für
Mathematik und
Informatik

Der Inhalt dieses Dokumentes darf ohne vorherige schriftliche Erlaubnis durch die FernUniversität in Hagen nicht (ganz oder teilweise) reproduziert, benutzt oder veröffentlicht werden. Das Copyright gilt für alle Formen der Speicherung und Reproduktion, in denen die vorliegenden Informationen eingeflossen sind, einschließlich und zwar ohne Begrenzung Magnetspeicher, Computerausdrucke und visuelle Anzeigen. Alle in diesem Dokument genannten Gebrauchsnamen, Handelsnamen und Warenbezeichnungen sind zumeist eingetragene Warenzeichen und urheberrechtlich geschützt. Warenzeichen, Patente oder Copyrights gelten gleich ohne ausdrückliche Nennung. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Inhaltsverzeichnis

| | |
|---|-----|
| Inhaltsverzeichnis | III |
| 1. Einleitung..... | 4 |
| 2. XSL-Transformationen | 4 |
| 3. XQuery..... | 20 |
| 4. Anhang..... | 36 |
| 5. Literatur | 38 |
| 6. Abbildungsverzeichnis | 39 |
| 7. Tabellenverzeichnis..... | 39 |
| 8. Lösungen der Selbsttestaufgaben..... | 40 |

1. Einleitung

Dies ist die vierte Kurseinheit des Kurses 1877 „Dokumenten- und Wissensmanagement im Internet“.

Anknüpfend an die Grundlagen von XML werden die Anwendungen XSLT und XQuery erarbeitet.

Die Absolventen des Kurses können die Anwendungsgebiete von XSLT bezüglich Darstellung und Datenaustausch erklären und eigene XSLT-Templates, beispielsweise mit dem Ziel der selektiven Datenausgabe, erzeugen.

Darauffolgend wird XQuery als Abfragesprache für XML-Dokumente eingeführt, mittels derer Suchanfragen auf XML-Datenquellen gestellt werden können. Die Studierenden erarbeiten sich die Prinzipien der Abfrageverarbeitung, des Typsystems und Sprachenkonzepts, sodass sie schließlich in der Lage sind, selbständig XQuery-Abfragen zu erstellen.

2. XSL-Transformationen

XSLT

XSL-Transformationen (XSLT) ist ein Teil der extensiblen Stylesheet Language, obwohl die Sprache auch unabhängig von XSL existieren kann. XSLT ist eine XML-Anwendung, d. h. ein XSLT-Dokument ist auch ein XML-Dokument. Ein XSLT-Dokument beinhaltet Regeln, nach denen ein anderes XML-Dokument transformiert werden kann. Dabei ist das Zielformat prinzipiell frei und nur von dem XSLT-Dokument abhängig. Diese Freiheit gilt, solange sich die Struktur des Dokuments mit XSLT-Regeln beschreiben lässt. So ist es zum Beispiel möglich, ein XML-Dokument zu einem HTML-Dokument zu transformieren, welches das XML-Dokument in Browsern in bestimmter Weise darstellt. Es ist z. B. auch möglich, ein XML-Dokument zu einem LATEX-Dokument zu konvertieren. Weiterhin kann ein XML-Dokument z. B. in ein anderes XML-Dokument transformiert werden. Es kann aber auch nur darum gehen, bestimmte Quelldaten neu zu ordnen, zu erweitern oder zu filtern.

Stylesheet

Ein XSLT-Dokument wird u. a. auch Stylesheet genannt. Ein Stylesheet enthält Vorlagen oder Muster (engl. *templates*), die jeweils aus einem Muster und einem Inhalt bestehen. Ein XSLT-Prozessor, d. h. eine Anwendung, die ein XML-Dokument nach Maßgabe eines XSLT-Stylesheets transformiert, sucht im XML-Dokument nach Vorkommen der Muster dieser Templates. Findet er ein passendes Template-Muster, gibt er den Inhalt des Templates aus. Das Stylesheet kann auch nur einen Teil des Inhalts aus dem Quelldokument beachten und das Ergebnisdokument um Informationen erweitern, welche im Quelltext gar nicht vorhanden sind.

XSLT-Stylesheets sind selbst wohlgeformte XML-Dokumente. XSLT benutzt bei der Transformation nur das im XML-Dokument vorhandene Markup. Inhaltsmodelle wie DTDs oder XML-Schemata werden nicht einbezogen. Es erfolgt somit keine Prüfung auf Gültigkeit, aber das Quelldokument muss wohlgeformt sein. Genau aus diesem Grund können XSLT-Stylesheets auch nicht auf HTML-Dateien angewendet werden, sehr wohl aber auf XHTML-Dokumente, weil diese ja die XML-Syntax berücksichtigen müssen.

XSLT ist somit eine Programmiersprache und wird benutzt, um Inhalte eines XML-Dokumentes in ein Ergebnisdokument zu transformieren und für die Ausgabe in einem bestimmten Medium zu formatieren. Eine reine Transformation ist von Formatierungsaspekten völlig unabhängig. Die Transformation eines XML-Dokumentes wird mittels Umwandlungsregeln in XSLT definiert. Die Umwandlung selbst wird mit XSLT-Prozessoren vorgenommen, die diese Syntax verstehen und umsetzen können. XSLT-Prozessoren sind bereits in vielen Web-Browsern integriert.

Das Anwendungsgebiet von XSLT ist zum einen die Transformation zum Zweck der Darstellung, auch Präsentationsorientiertes Publizieren (engl. *presentation oriented publishing, POP*) genannt. Mit unterschiedlichen Stylesheets können die Daten unter anderem in XHTML, SVG und viele andere Formate umgewandelt werden.

Weiter wird die Transformation zum Zwecke des Datenaustausches, gerne mit dem englischen Begriff *message oriented middleware (MOM)* bezeichnet, benutzt. Da XML lediglich ein Sprachkonzept zur Entwicklung von Sprachen bildet, genügt es nicht, dass zwei Programme XML beherrschen. Zum gegenseitigen Datenaustausch müssen sie die gleiche XML-basierte Sprache verwenden. Falls sie nicht die gleiche Sprache verwenden, wird gerne XSLT eingesetzt, um mithilfe von Transformationen Übersetzer von der einen Sprache in die andere zu entwickeln.

Es gibt einige frei verfügbare XSLT-Prozessoren wie [xalan-java](#)¹. Dieser ist in Java geschrieben und damit plattformunabhängig. Entwicklungsumgebungen wie z. B. XML-Spy von Altova arbeiten mit eigenen integrierten XSLT-Prozessoren. Seit Windows 7 ist der Microsoft Core XML Services MSXML bereits integriert und muss nicht separat installiert werden. Der Aufruf in einem Shell-Script sieht aus wie folgt, wobei eventuell noch die Pfade zu den jar-Dateien ergänzt werden müssen:

```
java -cp xalan.jar:xml-apis.jar:xercesImpl.jar  
org.apache.xalan.xslt.Process -IN $1 -XSL $2 -OUT $3
```

Hierbei stehen \$1, \$2 und \$3 für die Dateinamen des zu transformierenden XML-Dokuments, des XSLT-Stylesheets und des Ausgabedokuments. Die drei Dateinamen

¹ <http://www.apache.org/dyn/closer.cgi/xalan/xalan-j>

müssen beim Aufruf des Shell-Scripts als Kommandozeilen-Parameter übergeben werden.

Dies gilt allerdings nur für alte Java Versionen. Bei aktuelleren Versionen lässt sich im die jar-Datei direkt ausführen (hier als Beispiel befinden wir uns am Speicherort der Java-Datei und die IN und XSL Datei befinden sich ebenfalls dort, dann kann man die Ausführung starten durch:

```
java -jar xalan.jar -IN $1 -XSL $2 -OUT $3
```

Hier lassen sich auch absolute Pfade verwenden damit die Dateiverwaltung einfacher ist.

Xalan-J kann unter <http://www.apache.org/dyn/closer.cgi/xalan/xalan-j> heruntergeladen werden.

XSLT ist sehr umfangreich und verwendet teilweise die Syntax von XPath (siehe Kurseinheit 3). In unserer einführenden Darstellung werden wir allerdings zunächst keine XPath-Ausdrücke benutzen.

Selbsttestaufgabe 4.1:

Beschreiben Sie kurz, was XSLT-Stylesheets sind, und warum man sie nicht auf HTML-Dateien anwenden kann.

Entwicklung und Standards von XSLT

Die Empfehlung für XSL-Transformation Version 1.0 wurde im November 1999 verabschiedet. Hier geht es vor allem um die benötigten Sprachelemente zur Transformation von Quell- in Zieldokumente. Die Arbeiten am Vokabular für mögliche Formatierungsoptionen unter der Überschrift XSL Formatting Objects (XSL-FO) haben dagegen längere Zeit in Anspruch genommen. Im Oktober 2001 wurde die Empfehlung für XSL gegeben, welche XSLT und XSL-FO einschließt. Im Dezember 2006 folgte XSL 1.1.

Das W3C hat schließlich im Januar 2007 eine weitere Empfehlung zu XSL-Transformation 2.0 herausgegeben, in der es hauptsächlich um die Erweiterung der Gestaltungsmöglichkeiten durch Stylesheets geht. Neben neuen Funktionen und Elementen betrifft ein großer Teil die Formatierung von Datums- und Zeitwerten.

Im Wesentlichen geht es bei XSLT 3.0 um Transformationen im Streaming-Modus, wobei Quell- und Zieldokument nicht mehr komplett in den Hauptspeicher geladen werden; Stylesheets lassen sich aus Modulen zusammensetzen, und XSLT 3.0 wird die Funktionalitäten und das Datenmodell von XPath 3.0 unterstützen

Funktionsweise

Eine Transformation besteht aus einer Reihe von einzelnen *Transformationsregeln*, die *Templates* genannt werden. Ein Template beinhaltet ein auf XPath basierendes Muster, das beschreibt, für welche Knoten es gilt. Auf diese Weise wird die darzustellende Gesamtmenge bereits reduziert. Ein Beispiel:

In der XML Datei „[Datei „Batman“](#)² (auch im Anhang enthalten) gibt es den Wurzelknoten „root“. Die Child Knoten sind *Batman* und *BatmanBegins*. Beide haben jeweils einen Kindknoten *actors* sowie *filmcrew:members*.

Mit / kann man auf die Wurzel zugreifen. Durch die Angabe des Pfades ist es möglich, einzelne Einträge aufzurufen. Mit */root/Batman/actors/actor[1]/text()* wird im Knoten Batman der Knoten *actors* durchsucht nach dem ersten actor Eintrag und dort der Texteintrag aufgerufen.

Durch das Verwenden des // kann der absolute Pfad auch vernachlässigt werden. *//actor[@id='Joker']* durchsucht alle Knoten nach einem actor Knoten mit der ID „Joker“ und gibt

```
Element='<actor id="Joker">Jack Nicholson</actor>'
```

zurück, während

```
//actor[@id='Batman']
```

die beiden Einträge:

```
Element='<actor id="Batman">Michael Keaton</actor>'  
Element='<actor id="Batman">Christian Bale</actor>'
```

zurückgibt.

Der Inhalt bestimmt, **wie** das Template seinen Teil des Zielbaums erzeugt. Templates können selbst in XSLT definiert werden. Die Transformation beginnt am Anfangsknoten des XML-Dokumentes. Für jedes *<xsl:apply-template ...>* wird geprüft, ob der gerade selektierte Knoten passt. Gibt es kein passendes Template, wird das Standard-Template verwendet.

Selbsttestaufgabe 2.4:

Wie muss der Pfadausdruck aussehen zur Ausgabe aller Knoten (die nach Filmen benannt wurden), in denen ein *actor* mit der ID Batman vorkommt?

²<https://moodle-wrm.fernuni-hagen.de/mod/resource/view.php?id=152580>

Verwenden Sie auch den xpath-Tester auf der Seite
<https://www.freematter.com/xpath-tester.html>³.

Aufbau eines XSLT-Stylesheets

XSLT benutzt den Namensraum mit der URI <https://www.freeformatter.com/xpath-tester.html>⁴.

Im Stylesheet können auch Elemente und Attribute stehen, die nicht zu dem oben genannten Namensraum gehören und werden in der Regel direkt zum Ergebnisbaum hinzugefügt.

Als Präfix wird üblicherweise xsl benutzt. Das Wurzelement eines XSLT-Stylesheets heißt xsl:stylesheet, synonym kann auch xsl:transform benutzt werden. Das Element xsl:stylesheet hat eine verpflichtende Attribut version, z. B. gilt für XSLT 1.0 der Wert 1.0. Die Zuordnung des Namensraumes ist notwendig. Es gibt weitere, optionale Attribute, die bei Interesse [hier](#)⁵ recherchiert werden können. Ein XSLT-Stylesheet hat damit mindestens den folgenden Umfang:

```
<?xml version = "1.0" ?>
<xsl:transform version = "1.0"
  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
</xsl:transform>
```

Die Wirkung der Anwendung eines XSLT-Prozessors über das Stylesheet auf ein XML-Dokument ist die, dass alle Auszeichnungen entfernt und die Inhalte der Elemente ausgegeben werden. Zum Beispiel sei folgendes XML-Dokument gegeben:

```
<?xml version = "1.0" ?>
<buch>
<titel>Dies ist der Buchtitel</titel>
<autor name = 'Nachname' name = "Vorname" / >
<Inhalt>
<Kapitel Nummer = '1'>
<Unterkapitel Nummer = "1.3">Der Name des Unterkapitels
1.3</Unterkapitel>
</Kapitel>
<Kapitel Nummer = '2'>Der Name des zweiten
Kapitels</Kapitel>
</Inhalt>
```

³ <https://www.freematter.com/xpath-tester.html>

⁴ <https://www.freeformatter.com/xpath-tester.html>

⁵ <http://www.w3.org/TR/xslt#stylesheet-element>

```
</buch>
```

Die Ausgabe der Transformation des XML-Dokuments mit obigem Stylesheet sieht dann aus wie folgt:

```
<?xml version="1.0" encoding="UTF-8"?>
Dies ist der Buchtitel
Der Name des Unterkapitels 1.3
Der Name des zweiten Kapitels
```

Der XSLT-Prozessor hat eine Textdeklaration vorangestellt, damit die Ausgabe ein XML-Fragment bildet. Die Leerzeilen entsprechen den Zeilen des XML-Dokuments, in denen kein Inhalt steht. Der Grund für das beschriebene Verhalten ist, dass in XSLT sieben Arten von sogenannten Knoten eines XML-Dokuments unterschieden werden.

- Wurzel
- Element
- Attribut
- Text
- Kommentar
- Steueranweisung
- Namensraum

7 XML-Knoten

Für jede Art Knoten gibt es eine vorgegebene Regel, die greift, wenn im Stylesheet keine Regel für diese Art Knoten definiert ist. Diese Regeln lauten:

- Für Element- und Wurzelknoten werden die Kindelemente untersucht.
- Für Text- und Attributknoten wird der Text bzw. der Attributwert ausgegeben.
- Für Kommentar- und Steueranweisungsknoten wird nichts ausgegeben.
- Für Namensraumknoten werden alle Namensraum-Deklarationen automatisch in das Ausgabedokument eingefügt.

Die Regeln bewirken in einem angenommenen minimalen Stylesheet, dass nacheinander alle Elemente betrachtet werden und dass bei jedem betrachteten Element der Inhalt ausgegeben wird.

Grundlegende XSLT-Templates

XSLT ist sehr umfangreich. Es gibt sogenannte Top-Level-Elemente als direkte Kinder des Wurzelementes und zusätzlich Anweisungselemente, die überall in einem

Template auftreten können. Darüber hinaus können durch die Nutzung von XPath und durch XSLT-Prozessor-spezifische APIs beliebig komplexe Transformationen realisiert werden. Die vorliegende Kurseinheit soll sich im Folgenden auf die Darstellung der Grundprinzipien beschränken. Eine vollständige Darstellung findet man zum Beispiel in Kapitel 20 von [M01] oder im Web unter dem Suchbegriff *XSLTreference*.

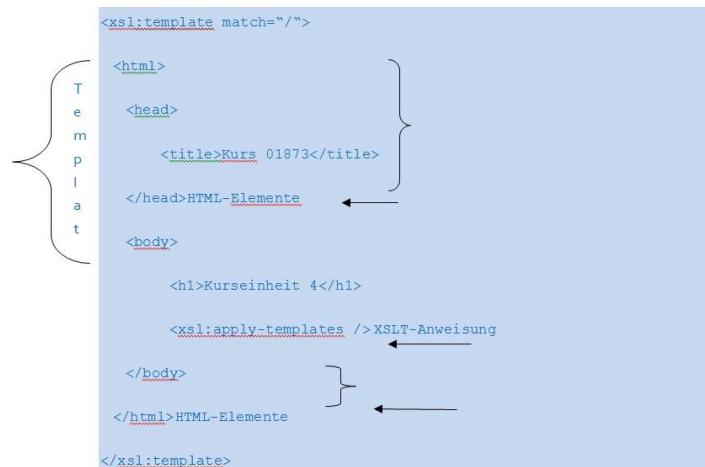
Die Top-Level-Elemente von <stylesheet> sind:

```
<xsl:import/>
<xsl:include/>
<xsl:namespace-alias/>
<xsl:strip-space/>
<xsl:preserve-space/>
<xsl:output/>
<xsl:key/>
<xsl:decimal-format/>
<xsl:attribute-set/>
<xsl:variable/>
<xsl:param/>
<xsl:template/>
```

In welcher Reihenfolge die Elemente im Stylesheet platziert werden, ist nicht relevant, außer <xsl:import>-Elemente, diese müssen am Anfang stehen. Es können auch eigene Erweiterungselemente im Stylesheet verwendet werden, wenn für diese ein Namensraum definiert wurde, aber der XSLT-Prozessor kann diese auch ignorieren.

Das grundlegende Element zur Verarbeitung ist xsl:template. Zum einen können im Template XSLT-Elemente mit Anweisungen an den XSLT-Prozessor vorkommen, welche dasselbe Namensraumpräfix haben, z. B. <xsl:apply-templates/> um die Regeln auszuführen. Zum anderen sind das Elemente, die nicht zum Namensraum von XSLT gehören, z. B. HTML-Tags wie <html>, <head> und <body> und unverändert in das Zieldokument übernommen werden.

Möglicher Aufbau einer Template-Regel



Das wichtigste Attribut ist `match`, um die Suchmuster anzugeben. `match` enthält im einfachen Fall den Namen eines Elements oder den Namen eines Attributs. Zur Unterscheidung wird Attributen in XSLT das at-Zeichen `@` vorangestellt. Der Inhalt von `xsl:template` gibt an, was ausgegeben werden soll. Damit ändert dieses Template die zweite vorgegebene Regel, falls der Inhalt von `xsl:template` nicht leer ist.

Um zu beeinflussen, was ausgegeben werden soll, gibt es das Element `xsl:value-of`. Sein Attribut `select` hat als Wert im einfachsten Fall den Namen eines Elements oder eines Attributs mit führendem `@`. Die Wirkung ist die Ausgabe des Elementinhalts bzw. des Attributwerts.

Das bereits oben verwendete XML-Dokument wird mit folgendem Stylesheet exemplarisch transformiert:

```

<?xml version = "1.0" ?>
<xsl:transform version = "1.0"
  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match = "buch">
    <xsl:value-of select = "Inhalt" />
  </xsl:template>
</xsl:transform>

```

Es ergibt sich dann als Ausgabe:

```

<?xml version="1.0" encoding="UTF-8"?>
Der Name des Unterkapitels 1.3
Der Name des zweiten Kapitels

```

Die Wirkung ist, dass für jedes Element `buch` der Inhalt des Kindelements `Inhalt` ausgegeben wird. Hierzu zählt auch der Inhalt der Kindelemente von `Inhalt`.

selektive Ausgabe

Manchmal ist eine Einschränkung bezüglich der Ausführung eines Templates sinnvoll. Zum Beispiel: Es gibt zwar ein Template für das Element `titel`, das an anderer Stelle benutzt wird, aber in der gegenwärtigen Anwendung sollen lediglich die Unterkapitel ausgegeben werden. Hierzu dient das Element `xsl:apply-templates`, welches das Attribut `select` hat. Der Attributwert ist im einfachsten Fall der Name des Kindelements, auf das die Anwendung von `xsl:template`-Elementen beschränkt werden soll. Die Anwendung des Stylesheets

```
<?xml version = "1.0" ?>
<xsl:transform version = "1.0"
  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match = "buch">
    <xsl:apply-templates select = "Inhalt" />
  </xsl:template>
  <xsl:template match = "Inhalt">
    <xsl:apply-templates select = "Kapitel" />
  </xsl:template>
  <xsl:template match = "Kapitel">
    <xsl:apply-templates select = "Unterkapitel" />
  </xsl:template>
  <xsl:template match = "titel">
  </xsl:template>
</xsl:transform>
```

führt zur Ausgabe

```
<?xml version="1.0" encoding="UTF-8"?>
Entwicklung und Standards von XSLT
Die Empfehlung für XSL-Transformation Version 1.0 wurde im
November 1999 verabschiedet.
Hier geht es vor allem um die benötigten Sprachelemente
zur Transformation von Quell- in Zieldokumente.
Die Arbeiten am Vokabular für mögliche Formatierungsoptionen
unter der Überschrift XSL Formatting Objects (XSL-FO) haben dagegen längere Zeit in Anspruch genommen.
Im Oktober 2001 wurde die Empfehlung für XSL gegeben, welche XSLT und XSL-FO einschließt.
Im Dezember 2006 folgte XSL 1.1.
```

Obwohl es also ein Template für das Element `titel` gibt, kommt dieses nicht zur Anwendung, weil im Template für das übergeordnete Element `buch` bestimmt wird, dass nur Templates für das Kindelement `Inhalt` ausgeführt werden sollen.

Im Template für das Element Inhalt kann übrigens beim Element `xsl:apply-templates` das Attribut `select` weggelassen werden, da als Kindelemente nur Kapitel vorkommen können. Wird hingegen im Template für das Element Kapitel das Attribut `select` von `xsl:value-of` weggelassen, dann wird auch der Textinhalt des zweiten Kapitel-Elements mit ausgegeben.

Der Inhalt eines Templates kann natürlich auch sonstigen Text beinhalten, solange dieser wohlgeformt ist. So erzeugt das folgende Template ein HTML-Dokument mit dem Titel des Buchs:

```
<?xml version = "1.0" ?>
<xsl:transform version = "1.0"
  xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
<xsl:template match = "buch">
<html>
<head><title>Buchtitel</title></head>
<body>
<em>
<xsl:apply-templates select = "titel" />
</em>
</body>
</html>
</xsl:template>
</xsl:transform>
```

Das erzeugte HTML-Dokument sieht wie folgt aus:

```
<html>
<head>
  <META http-equiv="Content-Type"
    content="text/html; charset=UTF-8">
  <title>Buchtitel</title>
</head>
<body>
<em>
  Dies ist der Buchtitel
</em>
</body>
</html>
```

In diesem Fall hat der XSLT-Prozessor selbsttätig auf das Voranstellen einer Text-Deklaration verzichtet, weil erkannt wurde, dass ein HTML-Dokument erzeugt wird. Stattdessen hat er ein META-Element eingefügt, das über den verwendeten Zeichensatz informiert. Würde der XSLT-Prozessor ein `
`-Element entdecken, würde er vermutlich `
` ausgeben, da dies in HTML erlaubt und üblich ist. Allerdings würde

diese Schreibweise im Template selbst die Wohlgeformtheit verletzen und damit einen Fehler hervorrufen.

An dieser Stelle sei nochmal gesagt, dass es für die Entwicklung von XSLT-Anwendungen von essentieller Bedeutung ist, zu wissen, wie ein XSLT-Prozessor ein spezielles Stylesheet auf ein XML-Dokument abarbeitet:

- Zuerst liest und parst der Prozessor beide Dokumente, also das spezielle Stylesheet und XML-Dokument.
- Im Speicher wird für jedes Dokument eine interne Baumstruktur erstellt, wie bei XPath auch, denn das Stylesheet verwendet ja XPath-Ausdrücke, um die Knoten im Baum des Quelldokumentes zu suchen.
- Anschließend beginnt der eigentliche Prozess der Transformation. Zuerst wird immer die Template-Regel für die Wurzel des XML-Baums verarbeitet. Dabei werden immer genauso viele Quelldaten abgegriffen, wie es die Anweisungen der Template-Regeln erfordern.
- Mit diesen Quelldaten und Elementen, welche nicht zum XSLT-Namensraum gehören, wird der Baum für das Zieldokument aufgebaut.
- Anschließend werden durch eine Serialisierung die Zieldokumente erzeugt, je nachdem was im Stylesheet angegeben wurde, entweder eine XML-Datei, HTML-Datei oder Textdatei.

Nun folgen noch kurz die Prioritätsregeln, wenn es zu Template-Konflikten kommen sollte, also wenn mehrere Templates auf ein und denselben Knoten betreffen:

1. Regeln mit spezifischen Informationen kommen immer vor Regeln mit allgemeinen Informationen, z. B. kommt die spezifische Information.
2. `match="/fernuni/lehrgebiet/multimedia/professor/name"` vor `match="//name"`. Suchmuster mit Wildcards sind allgemeiner als Suchmuster ohne Wildcards.
3. Wenn mit den Optionen in Punkt 1 und 2 nicht priorisiert werden kann, dann bekommt die zuletzt stehende Regel den Vorrang.
4. Mit dem Attribut `priority` kann die Priorität gezielt beeinflusst werden, wobei Ganzzahlen als auch reelle Zahlen verwendet werden können, je höher die Zahl, umso höher die Priorität.

Es existieren in XSLT auch Anweisungen für Verzweigungen und Wiederholungen, mit welchen der Ablauf der Operationen zusätzlich beeinflusst werden kann die da lauten:

<xsl:if>, <xsl:choose> und <xsl:when>. Die <xsl:if>-Anweisung knüpft die Ausführung des Templates an eine Bedingung. Das geklammerte Template wird nur ausgeführt, wenn die Bedingung, welche mit dem Attribut test formuliert wird, erfüllt ist. Mit der <xsl:choose>-Anweisung gibt es weitere Wahlmöglichkeiten. Zusätzlich kann mit <xsl:otherwise> eine Default-Verzweigung festgelegt werden. In <xsl:when>-Elementen können gleichzeitig mehrere Ausdrücke wahr sein, aber es wird nur das Template des ersten Elements ausgeführt, das den Test besteht.

Eine else-Verzweigung gibt es mit <xsl:if> nicht. In XSLT gibt es auch keine Schleifen wie in anderen Programmiersprachen, sondern lediglich Iterationen mit der Anweisung

```
<xsl:for-each>.
```

Falls im Ergebnis, also im Zieldokument eine andere Anordnung benötigt wird als das Quelldokument vorgibt, gibt es die <xsl:sort>-Elemente, welche als Kindelemente in <xsl:apply-templates> oder <xsl:for-each> eingebunden werden können.

Parameter können an eine Template-Regel übergeben werden, wenn diese mit <xsl:apply-templates> oder <xsl:call-template> aufgerufen wird. Sofern der Prozessor es unterstützt, können dem Stylesheet auch globale Parameter übergeben werden. Lokale und globale Variablen betreffend ist XSLT im Vergleich zu anderen Programmiersprachen eher spärlich ausgerüstet. Mit dem Kindelement <xsl:variable> von <stylesheet> ist es möglich, globale Konstanten zu definieren oder lokal innerhalb eines Templates. Die so deklarierte Variable ist nicht mehr änderbar. Ein Template kann somit keine Variable an ein anderes Template übergeben, was dafür aber die Nutzung der verschiedenen Templates innerhalb des Stylesheets in verschiedenen Reihenfolgen vereinfacht. Variablen- und Parameternamen müssen in einem Template eindeutig sein. Es ist aber möglich, globale Variablen durch lokale Variablen zu überdecken, indem man denselben Bezeichner vergibt.

Nun wurde auch deutlich, dass XSLT zwar einige Möglichkeiten bietet in Bezug auf Verzweigungen oder Schleifen wie in üblichen Programmiersprachen wie Java oder C# auch, aber doch sehr eingeschränkt. Aber genau das verdeutlicht gut, dass es sich bei XSLT im Wesentlichen um eine beschreibende Sprache handelt.

Selbsttestaufgabe 4.3:

Schreiben Sie ein XSLT-Stylesheet, das aus dem hier verlinkten [XML-Dokument „Buch“⁶](#) (auch im Anhang enthalten) mit Element `buch` ein HTML-Dokument erzeugt, in dem die Nummern aller Kapitel und Unterkapitel ausgegeben werden.

⁶<https://moodle-wrm.fernuni-hagen.de/mod/resource/view.php?id=152581>

Selbsttestaufgabe 4.4:

Wiederholen Sie die sieben möglichen XML-Knotentypen und das Standard-Verhalten, falls im Stylesheet keine Regel für den jeweiligen Knoten vorhanden ist.

Beispiel Buchdatenbank

Im nachfolgenden, etwas umfangreicherem Beispiel einer Buchdatenbank wird exemplarisch dargestellt, wie aus einem XML-Dokument, das die Inhalte einer Buchdatenbank repräsentiert, über XLST die Transformation in eine neue Strukturierung des gleichen Datenbankinhalts erreicht werden kann:

XML-Dokument

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xmlstylesheet type="text/xsl" href="datenbank.xslt"?>
<resources           xmlns:xsi="http://w3.org/2001/XMLSchema-
instance"
  xsi:noNamespaceSchemaLocation="datenbank.xsd">
  <buecher>
    <buch isbn="8534">
      <verlag>Bücherdruck</verlag>
      <autor>Roman Herzig</autor>
      <title>Das verlorene Schatzi</title>
    </buch>
    <buch isbn="0567">
      <verlag>Die Binder</verlag>
      <autor>Sven Bach</autor>
      <title>Der Dunkle Mond</title>
    </buch>
    <buch isbn="9643">
      <verlag>Die Binder</verlag>
      <autor>Rene Westrupp</autor>
      <title>Das Ende des Buches</title>
    </buch>
    <buch isbn="3267">
      <verlag>Bücherdruck</verlag>
      <autor>Sven Bach</autor>
      <title>Informatik verstehen</title>
    </buch>
  </buecher>
</resources>
```

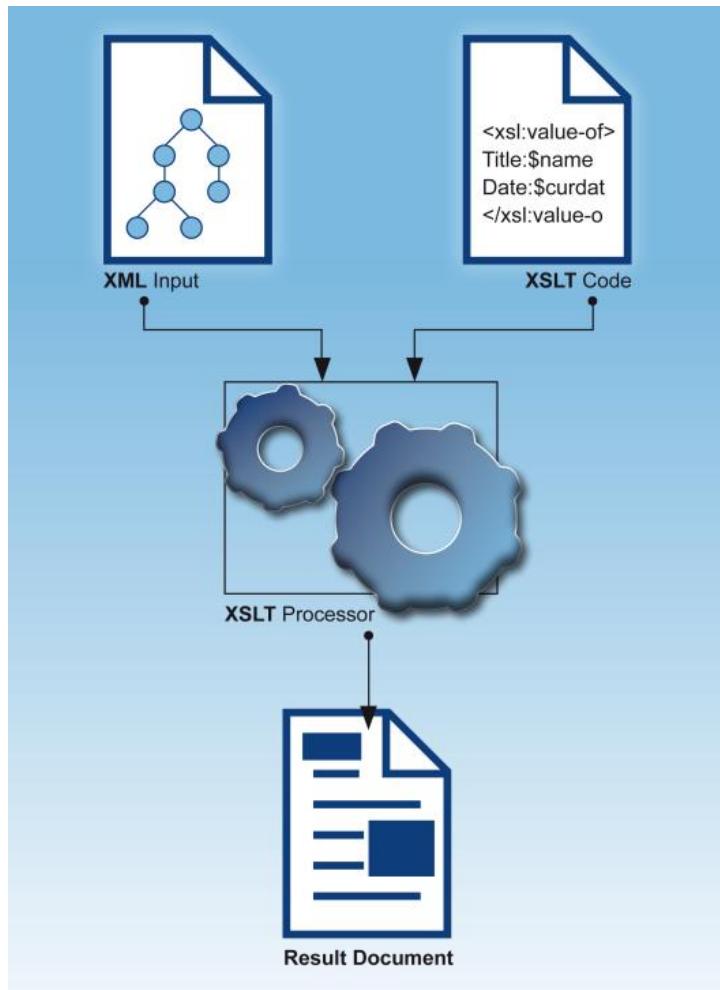


Abbildung 2.1 XSLT

XSLT-Template XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" encoding="ISO-8859-1"
  indent="yes"/>
<xsl:template match="/resources/buecher">
  <buecher>
    <xsl:for-each select="buch">
      <xsl:element name="buch">
        <xsl:attribute name="isbn"><xsl:value-of
          select="@isbn"/></xsl:attribute>
        <xsl:attribute name="titel"><xsl:value-of
          select="title"/></xsl:attribute>
        <xsl:attribute name="autor"><xsl:value-of
          select="autor"/></xsl:attribute>
        <xsl:attribute name="verlag"><xsl:value-of
```

```

        select="verlag"/></xsl:attribute>
    </xsl:element>
</xsl:for-each>
</buecher>
</xsl:template>
</xsl:stylesheet>
```

Transformation in XML (MOM)

```

<?xml version="1.0" encoding="UTF-8"?>
<buecher>
    <buch isbn="8534" titel="Das verlorene Schatzi"
        autor="Roman Herzig" verlag="Bücherdruck" />
    <buch isbn="0567" titel="Der dunkle Mond"
        autor="Sven Bach" verlag="Die Binder" />
    <buch isbn="9643" titel="Das Ende des Buches"
        autor="Rene Westrupp" verlag="Die Binder" />
    <buch isbn="3267" titel="Informatik verstehen"
        autor="Sven Bach" verlag="Bücherdruck" />
</buecher>
```

Anwendung von XPath in XSL-Templates

Abschließend in diesem Kapitel betrachten wir nochmals die XML-Datei „[Batman](#)⁷“ bzw. die hier abgebildete Datei [batman_rules.xsl](#)⁸. Diese verwendet xsl-Templates, also einzelne Transformationsregeln, in Form von XPath-Ausdrücken, um aus der Gesamtmenge der Informationen eine bestimmte Untermenge herauszufiltern (Übersicht über die Rollenbesetzungen).

⁷<https://moodle-wrm.fernuni-hagen.de/mod/resource/view.php?id=152580>

⁸https://moodle-wrm.fernuni-hagen.de/pluginfile.php/451106/mod_book/chapter/10649/batman_rules.xsl



```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsl:transform version = "1.0" xmlns:xsl = "http://www.w3.org/1999/XSL/Transform" xmlns:filmcrew = "https://personodata.toolforge.org/p/">
3  <xsl:output method="text"/>
4  <xsl:variable name="roles" select="//actor//@id[not(.=preceding::actor//@id)]"/>
5  <xsl:template name="describe-roles">
6    <xsl:param name="roles"/>
7    <xsl:for-each select="$roles">
8      <xsl:variable name="thisrole" select=". />
9      <xsl:value-of select="" />
10     <xsl:text> wurde von </xsl:text>
11     <xsl:for-each select="//actor[@id=$thisrole]">
12       <xsl:choose>
13         <xsl:when test="position()=1">
14           <xsl:otherwise>
15             <xsl:text> und </xsl:text>
16           </xsl:otherwise>
17         </xsl:choose>
18         <xsl:value-of select=".//text()" />
19       </xsl:for-each>
20     <xsl:text> gespielt. &#xa;</xsl:text>
21   </xsl:for-each>
22 </xsl:template>
23 <xsl:template match ="root">
24   <xsl:text>Hier eine Übersicht über die Rollenbesetzungen in Batman sowie Batman Begins: &#xa;</xsl:text>
25   <xsl:call-template name="describe-roles">
26     <xsl:with-param name="roles" select="$roles"/>
27   </xsl:call-template>
28 </xsl:template>
29 </xsl:transform>
30

```

Abbildung 2.2 Xpath innerhalb von XSLT

Zur Erläuterung:

In Zeile 4 wird eine Variable mit Namen roles definiert. ([xsl:variable](#)). Dieser wird der Inhalt aller Knoten, die durch den XPath-Ausdruck

//actor//@id[not(.=preceding::actor//@id)] ausgewählt werden. Dabei wird der Variable der Inhalt von actor//@id hinzugefügt.

Gleichzeitig werden durch den Ausdruck in den eckigen Klammern alle doppelten Einträge herausgefiltert: Der Zeiger . steht dabei für die jetzige ID, die verglichen wird mit den IDs vorrangegangener actor Knoten (Ausdruck: preceding::actor//@ID).

Dem Template [describe-roles](#) wird die Variable roles dann in Zeile 6 als Parameter übergeben.

Mit einer For Each Schleife wird jede Rolle ausgewertet (Zeile 7ff). Dabei wird die aktuelle Rolle als Variable gespeichert ([thisrole](#)), anhand derer in der inneren For Each Schleife (Zeile 11ff) die actor Einträge ausgewählt werden.

Der [xsl:choose](#) Block von Zeile 12 bis 17 dient als Abfrage-Block der dafür sorgt, dass in jeder Iteration außer der Ersten ein „ „ und „ „ Textbaustein vor dem nächsten Schauspielernamen hinzugefügt wird.

Durch [select=".//text\(\)"](#) in Zeile 18 wird der Inhalt des actor Elementes, auf das der Zeiger zeigt (bestimmt durch die Iteration der inneren For Each Schleife), ausgegeben.

Zeile 23: mit diesem Template match wird die Bearbeitung des Hauptelements gestartet, wobei mit [call-template](#) das oben definierte Template aufgerufen wird.

Hier muss auch noch einmal ausdrücklich der roles-Parameter übergeben werden.

Die Anwendung des XSL-Templates auf die XML-Datei ergibt schließlich folgende Ausgabe:

Hier eine Übersicht über die Rollenbesetzungen in ausgewählten Batmanfilmen:

- Batman wurde von Michael Keaton und Christian Bale gespielt.
- Joker wurde von Jack Nicholson gespielt.
- Vicki Vale wurde von Kim Basinger gespielt.
- Alfred wurde von Micheal Gough und Michael Caine gespielt.
- Ra's al Ghul wurde von Liam Neeson gespielt.
- Rachel Dawes wurde von Katie Holmes gespielt.

Selbsttestaufgabe 4.5:

Schreiben Sie eine XSLT-Datei, welche die Mitglieder der Filmcrew sowie ihren Tätigkeitsbereich auflistet (ggf. auch mehrere).

Verwenden Sie dafür eine Variable *crewmembers*, die durch folgende Codezeile deklariert wird:

```
<xsl:variable name="crewmembers"
  select="//filmcrew:member//text() [not(.=..
  /preceding-sibling::filmcrew:member/text())]"/>
```

3. XQuery

Zunehmend werden mehr oder weniger komplex strukturierte Informationskollektionen in XML-Dokumenten dargestellt, gespeichert oder ausgetauscht. Dies gilt für Daten

- zur Kommunikation zwischen Anwendungsprogrammen,
- Logs,
- Blogs und somit RSS,
- Metadaten wie bei Schema, WSDL, XMP,
- und Präsentation von Daten wie in XHTML,
- Dokumente wie Word-Dateien,
- Sichten auf andere Datenquellen,

- in relationalen Datenbanken, wo über entsprechende Erweiterungen und spezielle Datentypen XML-Dokumente in großem Umfang bereitgestellt werden z. B. Oracle XML DB, IBM DB2 pureXML, der xml-Datentyp des MS SQL Servers.

Damit entsteht der Wunsch nach Software, die informationelle Zugriffe und damit quasi Suchanfragen auf XML-Datenquellen durchführen kann. Eine solche Abfrage-sprache sollte Funktionen zum Aufspüren und Interpretieren von Information aus verschiedenen Quellen zur Verfügung stellen. Des Weiteren soll das Selektieren von Elementen und Attributen, mehrere Dokumente zu einem neuen Dokument zusammenfügen, Berechnen neuer Daten, Hinzufügen von Elementen und Attributen zum Ergebnis sowie das Sortieren der Ergebnisse möglich sein.

Suchanfragen sollen darin präzise und leicht verständlich sein sowie genug Flexibilität besitzen, um ein breites Spektrum an XML-Informationsquellen, einschließlich Datenbanken und Dokumenten abzufragen. XQuery ist also eine Abfragesprache für XML-Dokumente, die ähnlich wie SQL für relationale Datenbanken funktioniert und als Ergebnisse Datensätze zurückliefert. Für XQuery gibt es eine non-XML-Syntax und eine XML-Syntax.

XQuery benutzt eine an XSLT und SQL angelehnte Syntax und verwendet XPath sowie das XML-Schema für sein Datenmodell und seine Funktionsbibliothek; als Typkonzept wird von XQuery ebenfalls das Typkonzept von XML-Schema übernommen.

Bei XQuery wurde außerdem die Spezifikation von XSLT 2.0 [W3C6] einbezogen. Diese enthält Konzepte für Stylesheets und Templates und umfasst die Spezifikationen für XPath 1.0 und 2.0 [W3C4, W3C5].

Für XQuery und XPath gilt, dass sie selbst keine XML-Anwendungen sind, sondern eine eigene Syntax zur Bildung von Zeichenketten enthalten, vor allem zu Adressierung von Untermengen eines Dokumentes (vgl. Kapitel XSL-Transformationen/Funktionsweise).

In den XPath-Dokumenten sind unter anderem Pfad- und Vergleichsausdrücke, bedingte und arithmetische Ausdrücke sowie einige Funktionen beschrieben. Weitere Spezifikationen, die neben der eigentlichen XQuery-Spezifikation eine Rolle spielen, sind im Literaturverzeichnis aufgeführt. Namensraumpräfixe haben in XQuery eine festgelegte Bedeutung im Prolog einer Abfrage. Vorgegebene Namensraumpräfixe in XQuery sind:

- `xml = http://www.w3.org/XML/1998/namespace` Festlegung von Sprache und Leerraumbehandlung
- `xs = http://www.w3.org/2001/XMLSchema` XML-Schema
- `xsi = http://www.w3.org/2001/XMLSchema-instance` XML-Schemainstanz

- fn = http://www.w3.org/2005/xpath-functions XPath-Funktionen
- xdt = http://www.w3.org/2005/xpath-datatypes XPath-Datentypen
- local = http://www.w3.org/2005/xquery-local-functions Lokale XQuery-Funktionen
- err = http://www.w3.org/2005/xqt-errors XQueryFehlermeldungen

Die Empfehlungen des W3C zu XQuery 3.0 liegen seit April 2014 vor und reihen sich in die Empfehlungen der 3.0-Versionen wie XPath 3.0 oder XSLT 3.0 ein. Aus diesem Grund wurde die Versionsnummer 2.0 übergangen.

Selbsttestaufgabe 4.6:

Ergänzen Sie die fehlenden Begriffe im Text:

XQuery ist eine Abfragesprache für XML-Dokumente, die ähnlich wie SQL für relationale _____ funktioniert und als Ergebnisse Datensätze zurückliefert. Für XQuery gibt es eine

_____ und eine XML-Syntax. Für XQuery und XPath gilt, dass sie selbst keine XML-_____ sind, sondern eine eigene Syntax zur Bildung von Zeichenketten enthalten, vor allem zu Adressierung von _____ eines Dokumentes.

Abfrageverarbeitung

Die XQuery-Abfrageverarbeitung ist in [W3C7] dargestellt. XML-Quelldokumente werden dabei in einem ersten Schritt extern geparsert und eventuell zusätzlich validiert und dann vom Query-Prozessor in eine Instanz des XQuery/XPath-Datenmodells [W3C1] überführt.

Die lokal gültigen Schemadefinitionen des statischen Kontextes können von den zugehörigen XML-Schemata extrahiert oder durch irgendeinen anderen Mechanismus erzeugt werden, wobei in beiden Fällen die Konsistenzbedingungen erfüllt werden müssen. XQuery definiert zwei Phasen der Verarbeitung: statische Analysephase und dynamische Auswertungsphase. Dabei ist die statische Überprüfung bei der Implementierung optional, wenn Typfehler zur Laufzeit erkannt werden.

| statische Analysephase

| Die statische Analysephase hängt vom Abfrageausdruck selbst und vom statischen Kontext ab. Während dieser Phase wird die Abfrage in eine interne Darstellung, den sogenannten Operationsbaum überführt. Der statische Kontext wird durch die Auswertungsumgebung initialisiert. Dieser wird dann um Informationen aus dem Prolog erweitert oder verändert. Beim Schemaimport werden die lokal gültigen Schemadefinitionen mit Informationen von importierten Schemata überschrieben. Der statische Kontext wird verwendet, um Schematypnamen, Funktionsnamen,

Namensraumpräfixe und Variablennamen zu erhalten. Wenn im statischen Kontext ein Name von einem dieser Typen im Operationsbaum nicht gefunden wird, wird ein statischer Fehler gemeldet. Der Operationsbaum wird dann durch Anwendung der impliziten Operationen wie Atomisierung und Extraktion der gültigen booleschen Werte normalisiert. Jedem Ausdruck wird dann ein statischer Typ zugewiesen. Der Zweck des statischen Typisierungsfeatures ist es, eine frühe Entdeckung von Typfehlern zu ermöglichen und Typinformationen abzuleiten, die nützlich sind, wenn die Auswertung eines Ausdrückes optimiert wird.

Die dynamische Auswertungsphase ist die Phase, während der der Wert eines Ausdrückes berechnet wird. Sie beginnt nach Vollendung der statischen Analysephase und kann nur starten, wenn keine Fehler während der statischen Analysephase ermittelt wurden. Die dynamische Auswertungsphase ist abhängig vom Operationsbaum, dem Ausdruck, der gerade ausgewertet wird, vom Dateninput und vom dynamischen Kontext, der der Reihe nach Informationen von der externen Umgebung und vom statischen Kontext bezieht. Die dynamische Auswertungsphase kann neue Datenmodellwerte erzeugen und sie kann den dynamischen Kontext erweitern, zum Beispiel indem sie Werte an Variablen bindet. Die Serialisierung ist der abschließende Prozess des Umwandelns einer XDM-Instanz in eine Sequenz von Achtbitzeichen z. B. ein XML-Dokument.

dynamische Auswertungsphase

Selbsttestaufgabe 4.7:

Worin unterscheiden sich die statische Analysephase und die dynamische Auswertungsphase bei XQuery-Abfragen?

Datenmodell

Bei [W3C7] wird zwischen dem Datenmodell als allgemeinem Konzept und den spezifischen Items wie Dokumente, Elemente, atomare Werte etc. unterschieden, die konkreten Ausformungen des Datenmodells sind. Quelldokumente werden in eine Instanz des Datenmodells überführt. Eine Operation eines Ausdrucks erzeugt auf dieser Datenmodellinstanz wieder eine Instanz des Datenmodells unter Berücksichtigung der Abgeschlossenheit des Modells.

Sequenz

Grundlegend ist, dass jede Instanz des Datenmodells eine Sequenz ist, wobei eine Sequenz eine geordnete Sammlung von null oder Item mehr Einträgen ist.

Sie kann nicht in einer Sequenz verschachtelt sein. Ein einzelner Wert wird als Sequenz modelliert, die ein Item – auch Eintrag genannt – enthält. Ein Item ist entweder ein Knoten oder ein atomarer Wert. Jeder Knoten ist einer der sieben Knotentypen, die weiter unten beschrieben sind. Notiert werden Sequenzen in Klammerbeschreibweise, wobei die einzelnen Items durch Kommata getrennt werden.

Item

Atomare Werte

Atomare Werte sind die primitiven Schematypen mit dem Präfix `xs` und benutzerdefinierte Typen. Sie werden durch Konstruktoren erzeugt. Dabei wird dem Wert ein Typ zugeordnet.

Bei atomaren Typen kann eine Typumwandlung durchgeführt werden mit z. B. "10" `cast as xs:integer`. XQuery hat eigene Typen für atomare Werte. Das Präfix dafür ist `xdt` und steht für XPath-Datentyp. Die XQuery-Typen sind zunächst `xdt:untyped` für Elementknoten, die nicht validiert wurden und `xdt:untypedAtomic` für Attributwerte mit unbekanntem Typ. Weitere XQuery-Typen sind:

`xdt:yearMonthDuration` und `xdt:dayTimeDuration`; diese werden von

`xs:duration` abgeleitet und sind im Gegensatz zum `xs`-Typ vollständig geordnet.

Knoten

Knoten bilden einen Baum, der aus einem Wurzelknoten plus allen Knoten besteht, die direkt oder indirekt vom Wurzelknoten erreichbar sind. Jeder Knoten gehört genau einem Baum, und jeder Baum hat genau einen Wurzelknoten.

Bei XQuery sind die sieben Arten von Knoten von XPath übernommen:

- Dokumentknoten: Ein Baum, dessen Wurzelknoten ein Dokumentknoten ist, kennzeichnet ein Dokument. Wenn der Wurzelknoten des Baumes kein Dokumentknoten ist, dann handelt es sich um ein Fragment.
- Elementknoten: entspricht einem XML-Element. Der Wert eines Elementknotens ist der textuelle Wert des Knotens und seiner Nachfolger bzw. der typisierte Wert aus der Schemavalidierung (oder `xs:untypedAny` ohne Schema: nicht typisiert und `xdt:untypedAtomic`: typisiert).
- Textknoten: haben reinen Text als Inhalt, der nicht weiter ausgewertet wird
Attributknoten: entsprechen den Attributen von XML-Elementen
Namensraumknoten: stehen für den im aktuellen Kontext gültigen Namensraum
- Verarbeitungsanweisungs-Knoten: nehmen eine Verarbeitungsanweisung (Ziel und Inhalt) auf
- Kommentarknoten: Knoten haben einen textuellen und einen typisierten Wert

Typsystem und Sprachkonzept

XQuery hat ein mächtiges und komplexes Typsystem. Die Typen werden von XML-Schema importiert. Jeder Ausdruck in XQuery hat einen statischen Typ und jede XDM-Instanz hat einen dynamischen Typ. Mithilfe des Typsystems ist es möglich, statische Fehler in XQuery-Abfragen zu entdecken, den Ergebnistyp gültiger Suchen

festzulegen und statisch sicherzustellen, dass das Resultat eine Suche von einem erwarteten Typ ist (abhängig vom Typ der Inputdatenmenge). Das Typsystem beinhaltet die Typen `xdt:untypedAtomic`, alle 19 primitiven XML-Schema-Typen, alle Benutzer-definierten atomaren Typen, `Empty` und `None`.

Die Grundstruktur einer Abfrage in XQuery besteht aus einem Prolog, gefolgt von einem oder mehreren Ausdrücken. Dabei gibt der Prolog den Kontext an, in dem der Ausdruck kompiliert und ausgewertet wird. Er beinhaltet: Namensraumdefinitionen, Schemaimporte, Vorgabe von Element- und Funktionsnamensraum, Funktionsdefinitionen, Importe von Funktionsbibliotheken, globale und externe Variablen-deklarationen usw. Neben der Anwendungsumgebung erweitert und verändert die Prologdeklaration den statischen Kontext. Im Folgenden werden die Ausdrücke von XQuery kurz im Überblick vorgestellt. Eine genaue Beschreibung der Ausdrücke findet sich in [W3C7].

Elementare Ausdrücke

Elementare Ausdrücke sind Literale, Variablenreferenzen, Ausdrücke für Kontextknoten, Konstruktoren und Funktionsaufrufe.

Literale

Ein Literal repräsentiert einen atomaren Wert. XQuery unterstützt Literale des Typs String in Anführungszeichen sowie im Wesentlichen die Zeichen des Zeichensatzes und Entityreferenzen und verschiedene numerische Literale wie Integer, Decimal und Double. Diese Literale werden anhand ihrer Schreibweise erkannt. So werden Zeichen, die zwischen Anführungszeichen stehen, als String interpretiert, reine Ziffern als Integer und Ziffern mit einem Punkt als Decimal. Die atomaren Werte von primitiven XML-Schema-Typen können mit Konstruktoren wie bei [W3C2] beschrieben wie folgt erzeugt werden: Der Ausdruck beginnt mit `xs:` es folgt als Konstruktorname der Name des Typs und in Klammern ein oder kein Argument des Typ `sxdt:anyAtomicType`. Der Konstruktor liefert einen Ausdruck des gewünschten Typs. Ein Beispiel dafür ist der Ausdruck `xs:integer(“4”)`, er gibt den ganzzahligen Wert 4 zurück.

atomarer Wert

Variablenreferenzen

Variablenreferenzen bestehen aus einem Dollarzeichen gefolgt von einem qualifizierten Variablenreferenzen binden Namen. Zum Beispiel `$x`. Variablen werden im Prolog oder in Modulen deklariert. In FLWOR- und Typeswitch-Ausdrücken können Variablenreferenzen an diese Ausdrücke gebunden werden.

Variablenreferenzen binden

Geklammerte Ausdrücke

Das Setzen von Klammern in Ausdrücken erlaubt die Behandlung von unterschiedlichen Operatoren nach ihrer jeweiligen Priorität.

Ausdrücke für Kontextknoten

| context items

- | Ausdrücke für Kontextknoten erlauben es, zum Auswertungszeitpunkt einen Bezug auf den aktuellen Kontextknoten herzustellen. Der Bezug wird durch einen Punkt ausgedrückt.

Funktionsaufruf

Ein Funktionsaufruf besteht aus dem Funktionsnamen, also ein qualifizierter Name vom Typ QName ; dahinter in Klammern keine oder bis zu mehreren Argumenten.

Pfadausdrücke

Ein Pfadausdruck dient dazu, Knoten in Bäumen zu lokalisieren. Diese Ausdrücke wurden von der XPath 1.0 Spezifikation [W3C4] in XQuery übernommen.

Lokalisierungsschritte

| steps

- | Ein Pfadausdruck beginnt mit Schrägstrichen / oder //, besteht aus einem oder mehreren Lokalisierungsschritten, die wiederum von Schrägstrichen / oder // getrennt werden. Hierbei steht der doppelte Schrägstrich als Trennzeichen zwischen den Schritten für /descendant-or-self::node() /, dies bedeutet, dass // unter Umständen für mehrere Hierarchiestufen im Baum steht. Der doppelte Schrägstrich zu Beginn des Ausdruckes adressiert alle Knoten des aktuellen XML-Dokuments. Bei den Lokalisierungsschritten wird zwischen Filter- und Achsen-schritten unterschieden.

Achsen

| axes

- | Die Achsen in XPath legen ausgehend vom aktuellen Standort self:: im XML-Dokument eine Richtung für die Suche fest. Dabei wird, wie die Grafik zeigt, nach vorwärts und rückwärts unterschieden.

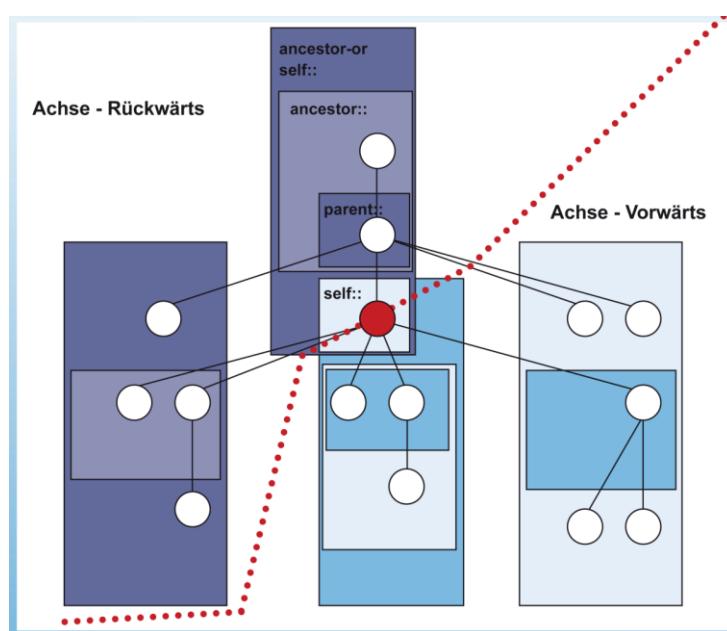


Abbildung 3.1 Achsen in XPath

Die Achsen `parent::`, `child::`, `preceding-sibling::` und `following-sibling::` adressieren dabei die direkt benachbarten Knoten. Ein Pfadausdruck könnte dann z. B. so aussehen:

```
/child::Rezept/ child::Rezeptname/ .
```

Tests

Ein Achsenschritt besteht aus einer Achsenangabe gefolgt von 2 Doppelpunkten und einem Knotentest. Dieser Test kann den Knotentyp oder den Knotennamen beitreffen.

Prädikate

Bei einem Lokalisierungsschritt können Prädikate zum Beispiel zum Ausfiltern bestimmter Knoten angegeben werden. Diese werden dann in eckigen Klammern notiert.

Abgekürzte Syntax

Für einige Pfadausdrücke existiert eine abkürzende Schreibweise. Beispiele dafür sind die oben erwähnten Schrägstriche oder auch `[@type="xyz"]` für `[attribute::type="xyz"]` und `..` für `parent::node()`.

Sequenzausdrücke

XQuery unterstützt Operatoren, um Sequenzen von Items zu konstruieren, zu filtern und zu kombinieren. Sequenzen sind nie verschachtelt. Wenn zum Beispiel die Werte `1`, `(2, 3)` und `()` zu einer Sequenz zusammengefügt werden, entsteht die Sequenz `(1, 2, 3)`.

Konstruieren von Sequenzen

Eine Möglichkeit eine Sequenz zu konstruieren ist der Kommaoperator, der alle seine Operanden auswertet und die resultierenden Sequenzen zu einer Ergebnissequenz verkettet. Leere Klammern bezeichnen eine leere Sequenz. Eine Sequenz kann Duplikate von atomaren Werten oder Knoten enthalten, aber eine Sequenz ist nie in einer anderen Sequenz enthalten. Wenn eine neue Sequenz erzeugt wird, indem zwei oder mehrere Eingangssequenzen verkettet werden, enthält die neue Sequenz alle Items (Einträge) der Eingangssequenzen. Beispiele sind `(10, 1, 2, 3, 4)` als eine Sequenz mit fünf Integerwerten und `(10, (1, 2), (), (3, 4))`, wobei dieser Ausdruck vier Sequenzen der Länge eins, zwei, null und zwei in eine einzelne Sequenz der Länge fünf verkettet. Das Resultat dieses Ausdruckes ist die Sequenz `10, 1, 2, 3, 4`. Als zweite Möglichkeit kann ein Intervallausdruck verwendet werden, um eine Sequenz aus aufeinander folgenden Integern zu konstruieren. So erzeugt z. B. `2 to 5` die Sequenz `(2, 3, 4, 5)`.

Filterausdrücke

Ein Filterausdruck besteht aus einem *Primärausdruck*, der von keinem oder mehr Prädikaten gefolgt wird. Das Resultat des Primärausdruck Filterausdruckes besteht aus allen Items, die durch den Primärausdruck als für alle Prädikate zutreffend zurückgegeben werden. Wenn keine Prädikate spezifiziert werden, ist das Resultat einfach das Resultat des Primärausdruckes.

So filtert z. B. `$Rezept [Zubereitungszeit lt „PT1H“]` alle Rezepte mit einer Zubereitungszeit heraus, die kürzer als eine Stunde sind, `(2 to 5) [3]` filtert den 3. Eintrag aus der Sequenz heraus und ergibt also 4.

Kombinierte Knotensequenzen

Combining node sequences

XQuery stellt Mengenoperatoren – allerdings nur für Knoten – für kombinierte Knotensequenzen zur Verfügung. Für die Vereinigung gibt es zwei gleichwertige Operatoren: `union` und `|`. Der Schnittmengenoperator `intersect` hat zwei Sequenzen von combining node sequences Knoten als Operanden und gibt eine Sequenz zurück, die alle Knoten enthält, die in beiden Operanden auftreten. Die Mengendifferenz hat den Operator `except`. Hier wird eine Sequenz erzeugt, die alle Knoten enthält, die im ersten Operanden, nicht aber im zweiten Operanden, auftreten.

Arithmetische Ausdrücke

XQuery akzeptiert Ausdrücke mit Operatoren für die Grundrechenarten: `+`, `-`, `*`, `div`. `Div` steht hier für die Division und `idiv` für die Division zweier Ganzahlen, wobei der Rest verfällt. Die Modulo Operation ist `mod`. Sie liefert den Rest einer Ganzzahldivision. Weitere mathematische Operationen sind als Funktionen realisiert. Ansonsten gelten die allgemeinen Rechenregeln wie „Punkt vor Strich“ und die Auswertungsreihenfolge von geklammerten Ausdrücken.

Vergleichsausdrücke

In XQuery gibt es unterschiedliche Arten von Vergleichsoperationen: Wertevergleiche, allgemeine Vergleiche und Knotenvergleiche. Im Folgenden werden die Vergleichsoperationen im Überblick dargestellt.

| Art des Vergleichs | Beschreibung | Operatoren |
|---------------------------|--|---|
| Wertevergleich | Operatoren, um einzelne Werte zu vergleichen (von wesentlicher Bedeutung in where-Klauseln von FLWOR-Ausdrücken) | <code>eq</code> , <code>ne</code> , <code>lt</code> , <code>le</code> , <code>gt</code> , <code>ge</code> |

| | | |
|-----------------------|--|------------------------|
| Allgemeiner Vergleich | Zwei Sequenzen von Werten werden verglichen. Ein Ausdruck gibt dann wahr zurück, wenn es mindestens in jeder Sequenz einen Wert gibt, der dem Operator genügt. | =, !=, <=, <, >, >= |
| Knotenvergleich 1 | In diesen Ausdrücken wird die Identität von einzelnen Knoten überprüft. | is, isnot |
| Knotenvergleich 2 | Operatoren, um die relative Position von zwei Knoten zueinander zu überprüfen (in document order). | <<, >> |

Tabelle 3.1 Vergleichsausdrücke

Logische Ausdrücke

Logische Ausdrücke werden durch die Operatoren and und or gebildet. Für Negation gibt es keinen Operator, daher muss auf die Funktion fn:not() zurückgegriffen werden. Zu beachten ist: wenn bei der Auswertung eines Operanden ein Laufzeitfehler auftritt, können implementierungsabhängig unterschiedliche Ergebnisse ausgegeben werden. Dies betrifft entweder den booleschen Wert eines Teilausdrucks oder error.

Konstruktoren

Konstruktoren dienen dazu, neue Knoten zu erzeugen. Es wird zwischen direkten und berechnenden Konstruktoren unterschieden. Bei direkten Elementkonstruktoren wird ein Element erzeugt, indem es in XML-Schreibweise notiert wird:

```
<Rezept land="xyz">xxxxx xx xxx</Rezept>
```

Es ist auch möglich, einen eingeschlossenen Ausdruck zu verwenden. Dem Prozessor muss dann durch geschweifte Klammern signalisiert werden, dass der Inhalt nicht direkt übernommen werden kann, sondern erst ausgewertet werden muss:

```
<ZutatenAnzahl>{3*4-2}</ZutatenAnzahl>
```

Attribute werden dabei notiert wie in XML üblich, wobei der Wert eines Attributes auch aus einem Ausdruck in geschweiften Klammern berechnet werden kann wie z. B.

```
Zutat Anzahl="3">Zwiebeln</Zutat>
```

oder

```
<Zutat Anzahl="{$Portionen*2}">Zwiebeln</Zutat>.
```

Namensraumdeklarationen

Ein Elementkonstruktor kann auch eine oder mehrere Namensraumdeklarationen enthalten:

```
<Rezept xmlns = „http://irgendwas.com/rezepte“>
```

Die Behandlung von Leerräumen in Elementkonstruktoren hängt von einem eventuellen Eintrag `xml:space` im Prolog ab. Ist dieser Eintrag vorhanden, bleiben alle Leerräume erhalten. Andernfalls werden alle boundary white spaces, also die begrenzenden Leerräume, entfernt. Das betrifft alle Leerräume, die zwischen den Elementauszeichnungen oder den Ausdrücken liegen. Andere Leerräume wie z. B. durch die Zeichenreferenz festgelegte oder innerhalb von Anführungszeichen stehende, bleiben erhalten. Weitere Konstruktoren sind die für Verarbeitungsanweisungen und Kommentare:

```
<?xyz irgendwas="xxx" ?>
<!--Kommentar...-->
```

Einsatzmöglichkeiten von element

Berechnende Elementkonstruktoren berechnen den Elementnamen. Beim direkten Elementkonstruktor war der Elementname eine Konstante. Die Syntax für den berechnenden Elementkonstruktor ist das Schlüsselwort `element`, gefolgt von einem Ausdruck in geschweiften Klammern, der den Elementnamen berechnet oder von einem QName; dann folgt der Elementinhalt, wieder in geschweiften Klammern:

```
element {fn:node-name($irgendwas)}
```

oder

```
{element Rezept{...irgendwas...}}
```

Berechnende Attributkonstruktoren erzeugen einen neuen Attributknoten. Dieser ist die Entsprechung zu einem Attribut in einem XML-Element.

Die Syntax ist der eines berechnenden Elementkonstruktoren ähnlich: Schlüsselwort, Attribute, QName oder {ein geschlossener Ausdruck für Name} und {ein geschlossener Ausdruck für Attributwert}:

```
attribute Menge {$x div 10}
```

oder

```
attribute {$name} {$x div 10}
```

Ähnlich aufgebaut sind die anderen berechnenden Konstruktoren für Dokumentknoten, Textknoten, Verarbeitungsanweisungsknoten und Kommentarknoten.

FLWOR-Ausdrücke

FLWOR steht für die möglichen Klauseln For, Let, Where, Order by und Return eines FLWOR-Ausdrucks. Dieser Ausdruck ist der zentrale Teil einer XQuery-Abfrage, ähnlich einer SELECT FROM WHERE-Anweisung in SQL. FLWOR-Ausdrücke können verschachtelt sein.

For- und Let-Klauseln

For- und Let-Klauseln dienen der Bindung von Variablen und können auch ohne den Kontext eines FLWOR-Ausdrucks sinnvoll eingesetzt werden. Die for-Klausel ist vergleichbar mit der FROM-Anweisung in SQL und leitet einen einfachen Iterationsausdruck ein. Die Syntax ist z. B.

```
for Variable in Ausdruck_1
return Ausdruck_2

for $rezept in document("RezeptDB.xml") //Rezept
return $rezept//Rezeptname/text()
```

Die Semantik dieses exemplarischen Ausdrucks ist: Die Variable wird an jeden Knoten, der von Ausdruck1 zurückgegeben wird, gebunden. Im Beispiel wird \$rezept an alle Rezepte aus Rezept DB.xml gebunden. Für jede Bindung wird dann Ausdruck_2 ausgewertet, die Ergebnisse werden zu einer Sequenz verkettet und Duplikate werden entfernt.

Die let-Klausel hat die Bedeutung einer lokalen Variablen Deklaration. Die Deklaration der XQuery-Variablen beginnt immer mit einem \$-Zeichen:

```
let Variable:=Ausdruck_1
return Ausdruck_2

let $rezept:=document("RezeptDB.xml") //Rezept
return $rezept//Rezeptname/text()
```

Hierbei wird die Variable nicht an jeden Knoten, sondern an das gesamte Ergebnis von Ausdruck_1 gebunden.

Where-, Order By- und Return-Klauseln

Die where-Klausel ist für das Filtern der Daten zuständig und ähnelt der Funktion von WHERE in SQL. Geschrieben wird sie where Ausdruck also z. B. where \$Kochzeit < PT1H (Bedingung Kochzeit < 1Std.).

Die Klausel order by veranlasst die Sortierung der Ergebnissequenz und return selektiert die Ausgabewerte für das Ergebnisdokument wie SELECT bei SQL:

```
for $rezept in fn:doc("RezeptDB.xml") //Rezept
```

```

for $Zeit1 in $rezept//Garzeit
let $Zeit := xdt:dayTimeDuration($Zeit1)
let $vergl := xdt:dayTimeDuration("PT45M")
where $Zeit lt $vergl
order by $Zeit
return {$rezept//Rezeptname/text()}

```

Fehlt die order by-Klausel im FLWOR-Ausdruck, wird die Reihenfolge durch die for-Klausel bestimmt. Die Sortierrichtung kann mit den Schlüsselwörtern ascending bzw. descending festgelegt werden.

Ausdrücke

Ausdrücke können wie erwähnt explizit geordnet oder ungeordnet zurückgegeben werden. Sinn eines ungeordneten Ausdrucks ist es, Ausführungszeit einzusparen.

If-then-else-Ausdrücke treten vor allem in FLWOR-Ausdrücken auf und werden wie in vergleichbaren Programmiersprachen behandelt.

[Quantifizierende Ausdrücke](#)⁹ ermöglichen die Behandlung von Existenzquantoren \exists durch die some ... in ... satisfies-Anweisung und von Allquantoren durch die all ... in ... satisfies-Anweisung.

Wie bei anderen Programmiersprachen besteht auch bei XQuery die Notwendigkeit, Ausdrücke für den Umgang mit Typen zur Verfügung zu haben. Für die Typabfrage von allen Sequenztypen gibt es den Ausdruck instance of. Mit typeswitch kann in Abhängigkeit vom Typ eines Ausdrucks eine von mehreren vorher festgelegten Alternativen ausgewählt werden. Zur Typumwandlung dient cast eventuell nach einer Typüberprüfung durch castable. Eine Typzusicherung erfolgt über treat as. Hierbei wird zur Laufzeit ein Fehler erzeugt, wenn ein Ausdruck nicht dem angegebenen Typ entspricht.

Funktionen

Mit den in XQuery definierten Ausdrücken ist bereits die Konstruktion einer Vielzahl von Operationen möglich. Erweitert werden diese Möglichkeiten durch eine breite Palette von Funktionen, wie sie in [W3C2] beschrieben sind. Darunter sind solche für boolesche und numerische Werte, für Zeit- und Datumswerte, zur Arbeit mit Zeichenketten und viele andere. Wenn diese Möglichkeiten nicht ausreichen, besteht noch die Möglichkeit, eigene Funktionen zu definieren. Zum Speichern von Funktionen und Variablen, um diese wiederverwenden zu können, dienen Bibliotheksmodule. Die dabei anzulegende XQuery-Datei beginnt mit einer Modul-Deklaration und

⁹ <http://www.w3.org/TR/xquery/#id-quantified-expressions>

dem entsprechendem Namensraum. Enthalten diese Module auch noch eine Anfrage, werden sie Hauptmodule genannt.

Die Werte von Variablen in XQuery können, wie auch bei XSLT, nicht mehr verändert werden, wenn ein Wert einmal zugewiesen ist. Variablen dienen eher dazu, den Code lesbarer zu gestalten, denn wenn ein Ausdruck an eine Variable gebunden wird, kann dieser leicht wiederverwendet werden.

Optionale Funktionen und Konsequenzen in der Praxis

In der XQuery-Spezifikation gibt es einige Funktionalitäten, die bei der Implementierung nicht verpflichtend sind. Die wesentlichen dieser optionalen Funktionalitäten sind im Folgenden kurz erläutert.

Wenn die Schemaimportfunktionalität nicht unterstützt wird, stehen nur die vordefinierten Typen aus XML-Schema zur Verfügung.

Bei der statischen Analyse kann eine Typprüfung entfallen, wenn sichergestellt ist, dass Typfehler zur Laufzeit erkannt werden.

Es ist freigestellt die Unterstützung für die Achsen ancestor, ancestor-or-self, preceding, following und following-sibling zu unterstützen.

Eine Implementierung von XQuery kann auf die Unterstützung von Bibliotheksmodulen verzichten.

Interessante XQuery-Option

Mit der Funktion doc() werden jeweils einzelne Dateien für die Abfrage zur Verfügung gestellt. Dagegen kann man mit der Funktion collection() gleich eine ganze Reihe von Dateien auswerten, beispielsweise kann als Parameter ein URI für einen Ordner mit XML-Dokumenten angegeben werden.

Neuere Möglichkeiten von XQuery

Während XQuery 1.0 weitestgehend auf die Abfrage von Daten begrenzt war, hat das W3C im März 2011 die Empfehlung für XQuery Update Facility verabschiedet. Diese unterstützt insbesondere Änderungen an den ausgewerteten Datenquellen. Mit Ausdrücken wie Insert, Delete, Replace, Rename oder Transform können Knoten eingefügt oder vorhandene Knoten oder deren Attribute manipuliert werden.

Mit XQuery and XPath Full Text ist nun auch eine Volltextsuche möglich. Beide gehören nicht zum Sprachkern von XQuery.

Einige Erweiterungen in XQuery 3.0:

- Try-catch Expression: Fehlerbehandlung bei dynamischen Fehlern

- Group by - clause in FLWOR Expressions: Durchführung der Return Clause gruppiert
- Switch Expression: Alternative je nach Wert
- Count - clause in FLWOR Expressions
- Empty – clause, ähnlich der outer joins in SQL
- Function Items, String Concatenations, Serialization ...

Selbsttestaufgabe 4.8:

Wiederholen Sie die unterschiedlichen Arten von Vergleichsoperationen bei XQuery-Abfragen.

In welche Gruppe gehören die folgenden Operatoren:

isnot

ne

!=

>>

gt

Selbsttestaufgabe 4.9:

Wofür steht die Abkürzung FLWOR?

XML und Datenbanken

Als Reaktion auf stetig wachsende XML-Datenbestände haben alle führenden Datenbankhersteller, wie Oracle, DB2, Microsoft SQL Server etc., effiziente XML-Verwaltungssysteme entwickelt. Dadurch ist die Verwaltung von herkömmlichen Daten und die Verwaltung von XML-Dokumenten zusammengewachsen.

Zusammenfassung

Anhand der frei verfügbaren Implementierung – XMLSpy® können erste Erfahrungen in der Entwicklung von Anwendungen mit XQuery gemacht werden.

Selbsttestaufgabe 4.10:

Betrachten Sie die folgende XML-Datei:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bestellung bestellnummer="123456"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="bestellung.xsd">
<kunde>FernUniversität in Hagen</kunde>
<lieferadresse>
```

```
<name>Monika Lücke</name>
<adresse>Universitätsstraße 1</adresse>
<stadt>58084 Hagen</stadt>
<land>Deutschland</land>
</lieferadresse>
<ware>
<beschreibung>Laserdrucker</beschreibung>
<bemerkung>Postscriptfähig</bemerkung>
<menge>1</menge>
<preis>810.90</preis>
</ware>
<ware>
<beschreibung>Tonerkartusche Schwarz</beschreibung>
<menge>1</menge>
<preis>200.49</preis>
</ware>
</bestellung>
```

Erstellen Sie XQuery-Ausdrücke für folgende Aufgaben:

1. Fragen Sie die Lieferadresse der Beschreibung ab.
2. Fragen Sie die Beschreibungen aller Waren mit einem Preis über 500.00 ab.
3. Fragen Sie die Beschreibungen aller Waren mit einem Preis über 5.00 ab und geben Sie die Ausgabe in umgekehrter alphabetischer Reihenfolge der Beschreibung aus.
4. Fragen Sie alle bestellten Waren ab, zu denen es eine Bemerkung gibt.

Die Möglichkeit, Ihre Abfragen zu testen gibt es durch die Installation unter
<http://saxon.sourceforge.net/> oder <https://artfiles.org/apache.org/xalan-xalan-j/binaries/>

<http://baseX.org/products/live-demo/>

4. Anhang

Zusatz-Material:

Datei „[Batman.xml](#)“

Datei „[Buch.xml](#)“

Relevante W3C Standards und Empfehlungen

Wichtige Quellen:

[www.w3.org/TR/xmlbase/](#) XML Base (Second Edition)- Januar 2009

[www.w3.org/TR/xlink11/](#) XML Linking Language (XLink) Version 1.1 Mai 2010

[www.w3.org/TR/xmlschema-11-1/](#) XML-Schema Part 1: Structures (Second Edition April 2012)

[www.w3.org/TR/xmlschema-11-2/](#) XML-Schema Part 2: Datatypes (Second Edition) April 2012

[www.w3.org/TR/xpath-30/](#) XML Path Language (XPath) Version 3.0 April 2014

[www.w3.org/TR/xmlstylesheet/](#) Associating Style Sheets with XML documents 1.0 Second Edition Oktober 2010

[www.w3.org/TR/REC-xml-names/](#) Namespaces in XML 1.0 (Third Edition) Dezember 2009

[www.w3.org./TR/xquery/](#) XQuery 1.0 Second Edition Dezember 2010

[www.w3.org/TR/xquery-semantics/](#) XQuery 1.0 and XPath 2.0 Formal Semantics (Second Edition) Dezember 2010

[www.w3.org/TR/xquery-30/](#) XQuery 3.0 April 2014

[www.w3.org/TR>xpath-functions-30/](#) XQuery and XPath Functions and Operators 3.0 April 2014

[www.w3.org/TR/xqueryx-30/](#) XML-Syntax for XQuery 3.0 (XQueryX 3.0) April 2014

[www.w3.org/TR>xpath-datamodel-30/](#) XQuery and XPath Data Model (XDM) 3.0 April 2014

[www.w3.org/TR/xslt-xquery-serialization-30/](#) XSLT and XQuery Serialization 3.0 April 2014

[www.w3.org/TR/html5/](#) HTML5 Oktober 2014

[W3C 1] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/xpath-datamodel/>.

[W3C 2] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Functions and Operators W3C Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/xpath-functions/>.

[W3C 3] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Formal Semantics. W3C Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/2005/CR-xquery-semantics-20051103/>.

[W3C 4] World Wide Web Consortium: XML Path Language XPath 1.0. W3C Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/1999/REC-xpath-19991116>.

[W3C 5] World Wide Web Consortium: XML Path Language (XPath) 2.0. W3C Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/2005/CR-xpath20-20051103/>.

[W3C 6] World Wide Web Consortium: XSL Transformations (XSLT) Version 2.0. W3C Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/2005/CR-xslt20-20051103/>.

[W3C 7] World Wide Web Consortium: XQuery 1.0: An XML Query Language. W3C Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/2005/CR-xquery-20051103/>.

[W3C 8] World Wide Web Consortium: XML Query Use Cases. W3C Working Draft 15 September 2005. Internet: <http://www.w3.org/TR/2005/WD-xquery-use-cases-20050915/>.

[W3C 9] World Wide Web Consortium: XML Query (XQuery) Requirements. W3C Working Draft 3 June 2005. Internet: <http://www.w3.org/TR/2005/WD-xquery-requirements-20050603>.

[W3C 10] World Wide Web Consortium: XSLT 2.0 and XQuery 1.0 Serialization. Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/2005/CR-xslt-xquery-serialization-20051103/>.

[W3C 11] World Wide Web Consortium: XQuery and XPath Full-Text Requirements. W3C Candidate Recommendation 3 November 2005. Internet:
<http://www.w3.org/TR/2003/WD-xquery-full-text-requirements-20030502/>.

[W3C 12] World Wide Web Consortium: XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004. Internet:
<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>.

5. Literatur

- [B06] Petrie, Bussler-Industrial Semantics and Magic.In IEEE INTERNET COMPUTING: 96-98, 07/08 2006
- [BHL01] Berners-Lee, Hendler, Lassila The Semantic Web. Scientific American Magazine: 35-44, May 2001
- [BBB97] Bayardo ,. Bohrer, Brice, Cichocki, Fowler, Helal, Kashyap,. Ksiezyk, Martin, Nodine, Rashid, Rusinkiewicz, Shea, Unnikrishnan, Unruh, and WoelkInfoSleuth: Agent- Based Semantic Integration of Information in Open and Dynamic Environments.In ACM 0-89791-911 -4/97 /0005: 195-206, 1997
- [BCV99] Bergamaschi, Castano, Vincini Semantic Integration of Semistructured and Structured Data Sources. In SIGMOD Record, 28(1): 54-59, March 1999
- [vH04] Antoniou, van Harmelen A Semantic Web Primer. The MIT Press, 2004
- [DMHF00] Decker, Melnik, van Harmelen, Fensel,. Klein, Broekstra, Erdmann, Horrocks The Semantic Web: The Roles of XML and RDF In IEEE INTERNET COMPUTING: 63-74, 09/10 2000
- [LWGG06] Lemmens,. Wytzisk, de By, Granell, Gould, van OosteromIntegrating Semantic and Syntactic Descriptions to Chain Geographic Services. In IEEE INTERNET COMPUTING: 42-52, 09/10 2006
- [M01] Harold., Means. XML in a Nutshell, Dt. Ausgabe. O'Reilly Verlag, Köln, 2001.
- [SvH05] Stuckenschmidt, van Harmelen Information Sharing on the Semantic Web Springer-Verlag, Berlin Heidelberg New York, 2005
- [SSR94] Sciore, M. Siegel, A. Rosenthal Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems.In ACM Transactions on Database Systems, 19(2): 254-290, June 1994
- [W3C1] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/xpath-datamodel/>
- [W3C2] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Functions and Operators W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/xpath-functions/>
- [W3C4] World Wide Web Consortium: XML Path Language XPath 1.0. W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/1999/REC-xpath-10>

19991116

- [W3C5] World Wide Web Consortium: XML Path Language (XPath) 2.0. W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/2005/CR-xpath20-20051103/>
- [W3C6] World Wide Web Consortium: XSL Transformations (XSLT) Version 2.0. W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/2005/CR-xslt20-20051103/>
- [W3C7] World Wide Web Consortium: XQuery 1.0: An XML Query Language. W3C Candidate Recommendation 3 November 2005. Internet: <http://www.w3.org/TR/2005/CR-xquery-20051103/>

6. Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 2.1 XSLT..... | 17 |
| Abbildung 2.2 Xpath innerhalb von XSLT | 19 |
| Abbildung 3.1 Achsen in XPath | 26 |

7. Tabellenverzeichnis

| | |
|---------------------------------------|----|
| Tabelle 3.1 Vergleichsausdrücke | 29 |
|---------------------------------------|----|

8. Lösungen der Selbsttestaufgaben

Selbsttestaufgabe 4.1

Beschreiben Sie kurz, was XSLT-Stylesheets sind, und warum man sie nicht auf HTML-Dateien anwenden kann.

- Ein XSLT-Dokument beinhaltet Regeln, nach denen ein anderes XML-Dokument transformiert werden kann.
- So ist es zum Beispiel möglich, ein XML-Dokument zu einem HTML-Dokument zu transformieren, welches das XML-Dokument in Browsern in bestimmter Weise darstellt. Ein XSLT-Dokument wird u. a. auch Stylesheet genannt. Ein Stylesheet enthält Vorlagen oder Muster (engl. *templates*), die jeweils aus einem Muster und einem Inhalt bestehen. XSLT benutzt bei der Transformation nur das im XML-Dokument vorhandene Markup.
- Inhaltsmodelle wie DTDs oder XML-Schemata werden nicht einbezogen.
- Genau aus diesem Grund können XSLT-Stylesheets auch nicht auf HTML-Dateien angewendet werden.

Selbsttestaufgabe 4.2

Wie muss der Pfadausdruck aussehen zur Ausgabe aller Knoten (die nach Filmen benannt wurden), in denen ein actor mit der ID Batman vorkommt?

Verwenden sie auch den xpath Tester auf der Seite <https://www.freeformatter.com/xpath-tester.html>

Lösung:

```
//actor[@id='Batman']/parent::*/*
```

Ausgabe:

```
Element=<Batman>

<actors>

<actor id="Batman">Michael Keaton</actor>

<actor id="Joker">Jack Nicholson</actor>

<actor id="Vicki Vale">Kim Basinger</actor>

<actor id="Alfred">Micheal Gough</actor>

</actors>

<filmcrew:members xmlns:filmcrew="https://persondata.toolforge.org/p/">
```

```
<filmcrew:member id="Regie">Tim Burton</filmcrew:member>

<filmcrew:member id="Drehbuch">Sam Hamm</filmcrew:member>

<filmcrew:member id="Drehbuch">Warren Skaaren</filmcrew:member>

<filmcrew:member id="Musik">Danny Elfman</filmcrew:member>

</filmcrew:members>

</Batman>' Element='<BatmanBegins>

<actors>

<actor id="Batman">Christian Bale</actor>

<actor id="Alfred">Michael Caine</actor>

<actor id="Ra's al Ghul">Liam Neeson</actor>

<actor id="Rachel Dawes">Katie Holmes</actor>

</actors>

<filmcrew:members xmlns:filmcrew="https://persondata.toolforge.org/p/">

<filmcrew:member id="Regie">Christopher Nolan</filmcrew:member>

<filmcrew:member id="Drehbuch">Christopher Nolan</filmcrew:member>

<filmcrew:member id="Drehbuch">David S. Goyer</filmcrew:member>

<filmcrew:member id="Musik">Hans Zimmer</filmcrew:member>

<filmcrew:member id="Musik">James Newton Howard</filmcrew:member>

</filmcrew:members>

</BatmanBegins>'
```

Selbsttestaufgabe 4.3

Schreiben Sie ein XSLT-Stylesheet, das aus einem XML-Dokument mit einem Element namens buch ein HTML-Dokument erzeugt, in dem die Nummern aller Kapitel und Unterkapitel ausgegeben werden.

Das XSLT-Stylesheet lautet:

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

<xsl:transform version = „1.0“ xmlns:xsl = „http://www.w3.org/1999/XSL/Transform“>

<xsl:template match = „buch“>

<html>

<head>

<title>Kapitelnummern</title>

</head>

<body>

<b>Kapitelnummern</b>

<hr />

<xsl:apply-templates select = „Inhalt“/>

</body>

</html>
</xsl:template>

<xsl:template match =“Inhalt“>

<xsl:apply-templates select =“Kapitel“/>

</xsl:template>

<xsl:template match =“Kapitel“>

<b>

<xsl:value-of select =“@Nummer“ />

</b>

<br />

<xsl:apply-templates select =“Unterkapitel“ />
```

```
<hr />

</xsl:template>

<xsl:template match =“Unterkapitel”>

<em>

<xsl:value-of select =“@Nummer” />

</em><br />

</xsl:template>

</xsl:transform>
```

Es erzeugt die Ausgabe:

```
<html>

<head>

<META http-equiv=“Content-Type” content=“text/html; charset=UTF-8”>

<title>Kapitelnummern</title>

</head>
<body>

<b>Kapitelnummern</b>

<hr>

<b>1</b>

<br>

<em>1.3</em>

<br>

<hr>
```

```
<b>2</b>
```

```
<br>
```

```
<hr>
```

```
</body>
```

```
</html>
```

Das Attribut **select** mit Wert Unterkapitel im Template für Kapitel kann übrigens nicht weggelassen werden, da sonst als zusätzliche Ausgabe der Textinhalt bei Kapitel 2 erscheinen würde!

Selbsttestaufgabe 4.4

Wiederholen Sie die sieben möglichen XML-Knotentypen und das Standard-Verhalten, falls im Stylesheet keine Regel für den

jeweiligen Knoten vorhanden ist.

Wurzel Element

Attribut Text Kommentar

Steueranweisung Namensraum

Für Element- und Wurzelknoten werden die Kindelemente untersucht.

Für Text- und Attributknoten wird der Text bzw. der Attributwert ausgegeben. Für Kommentar- und Steueranweisungsknoten wird nichts ausgegeben.

Für Namensraumknoten werden alle Namensraum-Deklarationen automatisch in das Ausgabedokument eingefügt.

Selbsttestaufgabe 4.5

Schreiben Sie eine XSLT Datei, welche die Mitglieder der Filmcrew sowie ihre Tätigkeitsbereich(en) auf-
listet.

Verwenden Sie dafür eine Variable *crewmembers* die durch folgende Codezeile declariert wird:

```
<xsl:variable name="crewmembers" select="//filmcrew:member//text()[not(.=../preceding- sibling::filmcrew:member/text())]"/>
```

Lösung zum [Download](#)

Lösung:

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

<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:filmcrew="https://persondata.toolforge.org/p/">

<xsl:output method="text"/>

<xsl:variable name="crewmembers" select="//filmcrew:member//text()[not(.=../preceding-sibling::filmcrew:member/text())]"/>

<xsl:template name="describe-crewmembers">

    <xsl:param name="crewmembers"/>

    <xsl:for-each select="$crewmembers">

        <xsl:variable name="thiscrewmember" select=". "/>

        <xsl:value-of select=". "/>

        <xsl:text> war tätig im Bereich </xsl:text>

        <xsl:for-each select="//filmcrew:member[./text()=$thiscrewmember]">

            <xsl:choose>

                <xsl:when test="position()=1">

                    <xsl:otherwise>

                        <xsl:text> und </xsl:text>

                    </xsl:otherwise>

                </xsl:when>
            </xsl:choose>

            <xsl:value-of select="./@id"/>

        </xsl:for-each>

        <xsl:text> tätig.&#xa;</xsl:text>

    </xsl:for-each>

</xsl:template>
```

```
<xsl:template match = "root">

<xsl:text>Eine Übersicht über Mitglieder der Filmcrew und ihre Tätigkeiten</xsl:text>

<xsl:call-template name="describe-crewmembers">

  <xsl:with-param name="crewmembers" select="$crewmembers"/>

</xsl:call-template>

</xsl:template>

</xsl:transform>
```

Textausgabe:

Eine Übersicht über Mitglieder der Filmcrew und ihre Tätigkeiten Tim Burton war tätig im Bereich Regie tätig.

Sam Hamm war tätig im Bereich Drehbuch tätig.

Warren Skaaren war tätig im Bereich Drehbuch tätig. Danny Elfman war tätig im Bereich Musik tätig.

Christopher Nolan war tätig im Bereich Regie und Drehbuch tätig. David S. Goyer war tätig im Bereich Drehbuch tätig.

Hans Zimmer war tätig im Bereich Musik tätig.

James Newton Howard war tätig im Bereich Musik tätig.

Selbsttestaufgabe 4.6

Ergänzen Sie die fehlenden Begriffe im Text:

XQuery ist eine Abfragesprache für XML-Dokumente, die ähnlich wie SQL für relationale **Datenbanken** funktioniert und als Ergebnisse Datensätze zurückliefert. Für XQuery gibt es eine **non-XML-Syntax** und eine XML-Syntax.

Für XQuery und XPath gilt, dass sie selbst keine XML-**Anwendungen** sind, sondern eine eigene Syntax zur Bildung von Zeichenketten enthalten, vor allem zur Adressierung von **Untermengen** eines Dokumentes.

Selbsttestaufgabe 4.7

Worin unterscheiden sich die statische Analysephase und die dynamische Auswertungsphase bei XQuery-Abfragen?

X-Query definiert zwei Phasen der Verarbeitung:

- statische Analysephase:

Während dieser Phase wird die Abfrage in eine interne Darstellung, den sogenannten Operationsbaum überführt.

- dynamische Auswertungsphase:

Phase, während der der Wert eines Ausdruckes berechnet wird. Sie beginnt nach Vollendung der statischen Analysephase und kann nur starten, wenn keine Fehler während der statischen Analysephase ermittelt wurden.

Selbsttestaufgabe 4.8

Wiederholen Sie die unterschiedlichen Arten von Vergleichsoperationen bei XQuery Abfragen.

In welche Gruppe gehören die folgenden Operatoren:

isnot - ne - != - >> - gt

| Art des Vergleichs | Beschreibung | Operatoren |
|-----------------------|--|------------------------|
| Wertevergleich | Operatoren, um einzelne Werte zu vergleichen (von wesentlicher Bedeutung in where-Klauseln von FLWOR-Ausdrücken) | eq, ne, lt, le, gt, ge |
| Allgemeiner Vergleich | Zwei Sequenzen von Werten werden verglichen. Ein Ausdruck gibt dann wahr zurück, wenn es mindestens in jeder Sequenz einen Wert gibt, der dem Operator genügt. | =, !=, <=, <, >, >= |
| Knotenvergleich 1 | In diesen Ausdrücken wird die Identität von einzelnen Knoten überprüft. | is, isnot |

| | | |
|-------------------|--|--------|
| Knotenvergleich 2 | Operatoren, um die relative Position von zwei Knoten zueinander zu überprüfen (in document order). | <<, >> |
|-------------------|--|--------|

Selbsttestaufgabe 4.9

Wofür steht die Abkürzung FLWOR?

FLWOR steht für die möglichen Klauseln For, Let, Where, Order by und Return eines FLWOR-Ausdrucks. Dieser Ausdruck ist der zentrale Teil einer XQuery-Abfrage, ähnlich einer SELECT FROM WHERE-Anweisung in SQL. FLWOR-Ausdrücke können verschachtelt sein.

Selbsttestaufgabe 4.10

Betrachten Sie die folgende XML-Datei:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<bestellung bestellnummer="123456" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="bestellung.xsd">

<kunde>FernUniversität in Hagen</kunde>

<lieferadresse>

<name>Monika Lücke</name>

<adresse>Universitätsstraße 1</adresse>

<stadt>58084 Hagen</stadt>

<land>Deutschland</land>

</lieferadresse>

<ware>

<beschreibung>Laserdrucker</beschreibung>

<bemerkung>Postscriptfähig</bemerkung>

<menge>1</menge>

<preis>810.90</preis>
```

```
</ware>

<ware>

<beschreibung>Tonerkartusche Schwarz</beschreibung>

<menge>1</menge>

<preis>200.49</preis>

</ware>

</bestellung>
```

Erstellen Sie XQuery-Ausdrücke für folgende Aufgaben:

1. Fragen Sie die Lieferadresse der Beschreibung ab.
2. Fragen Sie die Beschreibungen aller Waren mit einem Preis über 500.00 ab.
3. Fragen Sie die Beschreibungen aller Waren mit einem Preis über 5.00 ab und geben Sie die Ausgabe in umgekehrter alphabetischer Reihenfolge der Beschreibung aus.
4. Fragen Sie alle bestellten Waren ab, zu denen es eine Bemerkung gibt.
 - a. for \$x in doc(„bestellung.xml“)/bestellung/lieferadresse return \$x
 - b. for \$x in doc(„bestellung.xml“)/bestellung/ware where \$x/preis>500
 - c. return \$x/beschreibung
 - d. for \$x in doc(„bestellung.xml“)/bestellung/ware where \$x/preis>5
 - e. order by \$x/beschreibung descending return \$x/beschreibung
 - f. for \$x in doc(„bestellung.xml“)/bestellung/ware where \$x/bemerkung != „“
 - g. return \$x

000 000 000 (00/23)

00000-0-00-S 1

Prof. Dr.-Ing. Matthias L. Hemmje

01877

Dokumenten- und Wissensmanagement im Internet

Kurseinheit 3:

Einführung in XML, DTD und XSD

Fakultät für
Mathematik und
Informatik

Der Inhalt dieses Dokumentes darf ohne vorherige schriftliche Erlaubnis durch die FernUniversität in Hagen nicht (ganz oder teilweise) reproduziert, benutzt oder veröffentlicht werden. Das Copyright gilt für alle Formen der Speicherung und Reproduktion, in denen die vorliegenden Informationen eingeflossen sind, einschließlich und zwar ohne Begrenzung Magnetspeicher, Computerausdrucke und visuelle Anzeigen. Alle in diesem Dokument genannten Gebrauchsnamen, Handelsnamen und Warenbezeichnungen sind zumeist eingetragene Warenzeichen und urheberrechtlich geschützt. Warenzeichen, Patente oder Copyrights gelten gleich ohne ausdrückliche Nennung. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Inhaltsverzeichnis

| | |
|--|-----|
| Inhaltsverzeichnis | III |
| 1. Einleitung | 4 |
| 2. Motivation und grundlegende Konzepte von XML..... | 4 |
| 3. Die Struktur von XML-Dokumenten | 7 |
| 4. Der DTD und XSD | 24 |
| 5. Konfliktvermeidung bei XML | 32 |
| 6. Verweise per XML..... | 38 |
| 7. Anhang..... | 48 |
| 8. Literatur | 51 |
| 9. Abbildungsverzeichnis | 51 |
| 10. Tabellenverzeichnis | 51 |
| 11. Lösung der Selbsttestaufgaben | 52 |

1. Einleitung

Dies ist die dritte Kurseinheit des Kurses 1877 „Dokumenten- und Wissensmanagement im Internet“.

Die Kursteilnehmer setzen sich in dieser Kurseinheit detailliert mit XML als einer Meta-Auszeichnungssprache, mit der andere Auszeichnungssprachen definiert werden können, sowie als Datenformat und insbesondere auch als Datenaustauschformat zwischen Anwendungen auseinander. XML-Tags, -Attribute und -Elemente werden analysiert und in Übungsbeispielen angewendet.

Die Studierenden können außerdem die Rolle von Dokumenttyp-Definitionen (kurz DTDs) in Bezug auf XML-Dokumente beschreiben. Sie erlernen Grundlagen ihrer Struktur und Syntax, prüfen Stärken und Schwächen und vergleichen sie mit den sogenannten XML-Schema-Definitionen (kurz XSDs). Die Teilnehmer des Kurses sind in der Lage, eigene Schemabeschreibungen für XML-Dokumente zu erstellen.

Wichtige Aspekte der Konfliktvermeidung, wie etwa die Definition von Namensräumen, werden ebenfalls aufgezeigt.

Abschließend für den Inhaltsbereich der Einführung XML befassen sich die Studierenden mit der Realisierung von Verweisen, wobei verschiedene Methoden ausgeführt und beurteilt werden.

2. Motivation und grundlegende Konzepte von XML

XML ist in den vorhergehenden Kurstexten schon erwähnt worden. In dieser Kurseinheit wird XML nun detailliert betrachtet. Einige Stellen der folgenden Kurseinheit werden Ihnen eventuell bekannt vorkommen. Die meisten hier eingeführten Sachverhalte sind jedoch neu in diesem Kurszusammenhang.

Historie

Die Sprache XML ist ein Abkömmling der Sprache SGML. Das Ziel der Entwicklung war eine Teilmenge von SGML speziell zur Verwendung im Internet, wobei mit etwa 20 % des Sprachumfangs 80 % der Leistung von SGML erreicht werden sollten. Den Kern der Sprachfamilie bilden die XML-1.0-Spezifikation von 1998, ihre Erweiterung mit XML-Namespaces von 1999 und die Sprache zur Definition von Inhaltsmodellen, XML-Schema von 2001. Die aktuelle fünfte Edition der XML 1.0-Spezifikation erschien im November 2008, welche bis heute – abgesehen von einigen Korrekturen – weitgehend unverändert geblieben ist.

Das W3C hat 2004 eine Version XML-1.1 verabschiedet, um der Weiterentwicklung von Unicode besser zu entsprechen. Die zweite Änderung betrifft die Behandlung von Leer-
raum. XML in der Version 1.1 ist in dieser Hinsicht nicht abwärtskompatibel. Daher kann es passieren, dass ein XML-Parser-1.0 XML-Dokumente für Version 1.1 als nicht wohlge-
formt zurückweist. Die XML-Version 1.1 spielt derzeit in der praktischen Arbeit mit XML-
Daten kaum eine Rolle, zumal, wie oben bereits erwähnt, die 5. Edition von XML 1.0
gelockerte Regeln in Bezug auf Element- und Attributnamen aus XML-Version 1.1 über-
nommen hat.

Bis einschließlich Version 4.01 ist auch HTML ein Abkömmling von SGML. Während aber HTML nur eine feste Menge von Tags benutzt, entfällt diese Einschränkung in XML. Hier können beliebige Tags definiert werden, wobei allerdings in einer konkreten Anwendung die benutzbaren Tags und die Art ihrer Anwendung durch eine sogenannte Dokumen-
tenttyp-Definition eingeschränkt sein können. Damit ist XML eine sogenannte Meta-
Auszeichnungssprache, mit der andere Auszeichnungssprachen definiert werden kön-
nen, die dann häufig auch als XML-Anwendungen bzw. etwas informeller als XML-
Sprachen bzw. XML-Dialekte bezeichnet werden.

XML beinhaltet eine Grammatik, durch welche die Syntax eines XML-Dokumentes strikt definiert wird. Ähnlich wie bei einem Software-Programm bei einem Compiler kann ein XML-Dokument zunächst einem Parser zugeführt werden, der die Einhaltung der Grammatik prüft. XML-Dokumente, die der Grammatik genügen, heißen wohlgeformt. XML-
Dokumente, die der Grammatik nicht genügen, können nicht verarbeitet werden. Hier-
bei verhält es sich ähnlich wie etwa bei einem C-Compiler, der ein Programm mit Sy-
ntaxfehlern nicht in ein ausführbares Programm übersetzen kann. Ein validierender Parser kann somit auch die Einhaltung des Dokumentenmodells, also die Strukturbeschreibung mittels DTD oder XML-Schema, prüfen. Ein wohlgeformtes XML-Dokument, das mit ei-
nem Dokumentenmodell verknüpft ist und dem Schema genügt, heißt gültig. Ansonsten heißt es ungültig. Dabei ist entscheidend, dass die Wohlgeformtheit und Gültigkeit ma-
schinell geprüft werden können.

Parse
Wohlgeformtheit
Validierung
Gültigkeit

Anhand dieser allgemeinen Eigenschaften von XML kann hier nun ein Unterschied zu HTML deutlich gemacht werden: XML ist nicht vorrangig eine Sprache zur Darstellung von Dokumenten in Webbrowsersn. Mithilfe eines XML-Dokumentes können vielmehr beliebige, u. U. auch (multi-)mediale Inhalte inklusive Faktendaten in eine bezüglich ihrer Struktur oder anderer Eigenschaften annotierte Form überführt werden. Zur Weiterver-
arbeitung von XML wird eine dementsprechende Applikation benötigt, die das resultie-
rende XML-Dokument verarbeiten kann. Im Falle von XHTML kann dies lediglich ein Plug-
in eines Webbrowsers sein. Im Falle der Benutzung anderer Schema-Definitionen kann dies eine Datenbank, eine Tabellenkalkulation oder auch ein für einen sehr dedizierten Zweck eigens erstelltes Anwendungsprogramm sein.

Folglich bietet XML über die Repräsentation von Dokumenten und (ggf. multi-)medialen Inhalten hinaus durch seine Markierungs- und Annotationssyntax, also durch die Tags und Schema-Definitionen, die Möglichkeit, nahezu beliebige Datenkollektionen strukturiert sowie anwendungs- und plattformunabhängig zu speichern. Damit kann XML als Datenformat für alle möglichen Anwendungen und insbesondere auch als Datenaustauschformat zwischen Anwendungen und deren proprietären Datenformaten dienen.

XML ist um eine Reihe von weiteren Standards erweitert worden, die alle zur XML-Sprachfamilie gehören.



Abbildung 2.1 Überblick über die wichtigsten Mitglieder der Sprachfamilie XML [V15]

Die Extensible Stylesheet Language – kurz XSL – ist eine Formatsprache, welche aus drei Teilen besteht:

- XML Path Language – kurz XPath – für die exakte Adressierung der einzelnen Baumbestandteile (aktuelle Version seit 2014 ist XPath 3.0),
- XSL Transformations – kurz XSLT – für die Umwandlung eines XML-Dokuments in ein anderes XML-Format oder ein anderes Format (aktuelle Version seit 2007 XSLT 2.0),
- XSL Formatting Objects – kurz XSL-FO – für Formatierungsregeln und Stilangaben (aktuelle Version seit 2006 XSL 1.1).

Mit der XML-Anwendung XLink wird die Verbindung zwischen XML-Dokumenten im Web beschrieben. Dies funktioniert zwar in ähnlicher Weise wie die Tags zur Auszeichnung von Links in HTML, aber mit mächtigerem Umfang. Per XML-Anwendung XPointer werden einzelne Bestandteile bzw. Elemente eines XML-Dokumentes adressiert.

Abstrakte Schnittstellen für den Zugriff auf XML-Dokumente hat das W3C in der Spezifikation für das Dokumentenobjektmodell (engl. *document object model*, kurz *DOM*) definiert (aktuell Level 3). Beim DOM wird eine Baumstruktur zum XML-Dokument aufgebaut. Eine andere einfache Programmierschnittstelle für XML (engl. *simple API for XML*,

SAX) verfolgt ein ereignisorientiertes Konzept und hat sich neben DOM in der Praxis etabliert. In der Java-Welt werden zahlreiche Java-Programmierschnittstellen für XML (engl. *Java API for XML processing, JAXP*) zum Validieren, Parsen, Generieren und Transformieren von XML-Dokumenten zur Verfügung gestellt.

Mittlerweile gibt es auch einige verbreitete XML-Anwendungen, die in ihrem Anwendungsbereich selbst schon wieder den Status von de-facto-Standards haben. Beispiele sind XHTML, die XML-kompatible Reformulierung von HTML 4.0, skalierbare Vektorgrafiken (engl. *scalable vector graphics, SVG*) zur Codierung von Linienzeichnungen, X3D zur Codierung von 3-D-Szenen, MathML zur Codierung von mathematischen Gleichungen und schließlich ein einfaches Objektzugriffsmodell (engl. *simple object access protocol, SOAP*), ein Vokabular für den Nachrichtenaustausch zwischen Anwendungen. CML ist eine Beschreibungssprache im Bereich der Chemie und verwandter Wissenschaften wie der Molekularbiologie. Innerhalb des vorliegenden Kurses wird jedoch nur auf XHTML kurz eingegangen.

Die Entwicklung von XML und der mit XML verwandten Sprachstandards ist darüber hinaus nicht abgeschlossen. Insbesondere zeigt die Entwicklung in der Vergangenheit, dass auch existierende Standards jederzeit – aus durchaus gerechtfertigten Gründen – erneut zerlegt oder geändert und neu zusammengefasst werden können.

Die Notation in dieser Kurseinheit folgt weitgehend der Notation im Buch von Harold und Means [HM01], welches sich auch zum weiterführenden Studium eignet.

Selbsttestaufgabe 3.1:

Erläutern Sie den Unterschied zwischen einem wohlgeformten und einem gültigen XML-Dokument.

3. Die Struktur von XML-Dokumenten

Der Text eines XML-Dokumentes kann sich in einer oder mehreren Dateien befinden. Texte können allerdings auch als Datensatz in einer Datenbank oder als Information auf einer IP-Verbindung im Internet existieren.

XML-Tags

Ein XML-Dokument enthält XML-Elemente. XML-Elemente mit Inhalt bestehen aus einem öffnenden XML-Tag, dem Inhalt und einem schließenden XML-Tag. Der öffnende Tag besteht aus dem Elementnamen und eventuell einer Liste von Attributen, d. h. von Paaren aus Attributnamen und Attributwert, umschlossen von spitzen Klammern. Der schließende Tag besteht aus dem Elementnamen mit einem führenden Schrägstrich, umschlossen von spitzen Klammern. Bei Elementen ohne Inhalt kann alternativ zur Folge

von öffnendem und schließendem Tag auch ein einzelnes Tag aus Elementnamen, eventuell gefolgt von Attributen und einem abschließenden Schrägstrich, umschlossen von spitzen Klammern, benutzt werden. Beispiele für Tags sind:

```
<person geboren='ja'>  
    J. Keller  
</person>  
<Person geboren='ja' name='J. Keller'></Person>  
<br/>
```

Die Beispiele illustrieren unter anderem, dass in XML, anders als in HTML, zwischen Groß- und Kleinschreibung unterschieden wird. Die beiden Elemente `<person>` und `<Person>` sind also verschieden. Das letzte Beispiel ist der Zeilenumbruch aus XHTML. Hier wird sichtbar, warum sich HTML nicht ohne Änderung als XML-Anwendung definieren lässt: in HTML kann auch ein öffnender Tag ohne schließenden Tag benutzt werden, wie zum Beispiel `
`.

Der Inhalt eines Elementes kann aus Zeichendaten bestehen wie im ersten Beispiel, allerdings kann ein Element auch weitere Elemente enthalten.

Ein wohlgeformtes XML-Dokument enthält einen Baum, der aus Elementen besteht. Dies ist nicht nur eine Anleihe bei dem Bereich der Datenstrukturen, sondern analog werden die Bezeichnungen Wurzel-, Eltern-, Kind- und Geschwisterelement verwendet. Hier zeigt sich auch noch ein weiterer Unterschied zu HTML: Elemente, die sich überschneiden, wie zum Beispiel:

```
<em><h3>Test</em></h3>
```

sind in XML wegen der zugrundeliegenden Baumstruktur verboten.

Attribute von XML-Elementen

Die Attribute von XML-Elementen bestehen aus einem Attributnamen, einem Gleichheitszeichen und einem Attributwert. Letzterer ist von einfachen oder doppelten Anführungszeichen umschlossen. Dazwischen dürfen sich Leerzeichen befinden.

Bei der Definition eines Elements stellt sich die Frage, ob für Inhalt zusätzlich zu Textdaten Kindelemente oder Attribute genutzt werden sollen. Dies ist nur teilweise technisch zu entscheiden: Ein Element darf nur ein Attribut mit einem gleichen Namen haben, aber mehrere Kindelemente mit gleichem Namen. Kindelemente können wiederum Kindelemente haben, ein Attribut hingegen nicht.

XML-Namen

XML-Elementnamen und -Attributnamen sowie weitere Objekte müssen einigen Regeln genügen. Diese Namen werden unter dem Begriff XML-Namen zusammengefasst. Wie bereits erwähnt, beinhaltet die 5. Edition von XML 1.0 einige Lockerungen in Bezug auf die erlaubten Element- und Attributnamen. Die Regeln lauten:

- Alle XML-Namen oder Bezeichner müssen mit einem Buchstaben, Unterstrich oder Doppelpunkt beginnen.
- Ab der zweiten Stelle dürfen alle Zeichen verwendet werden, die als Namenszeichen zugelassen sind: Neben den Zeichen für die erste Stelle sind das die Zahlen, der Bindestrich und der Punkt. Die XML-Spezifikation bezieht sich auf einen sehr weiten Bereich des gesamten Unicode-Systems, z. B. sind auch Umlaute, Akzente etc. für Markup verwendbar.
- Bei XML-Namen wird zwischen Groß- und Kleinschreibung unterschieden.
- Namen dürfen nicht mit der Zeichenfolge „xml“ beginnen, ebenfalls unabhängig von der Groß- und Kleinschreibung, weil diese Zeichenfolge durch die XML-Spezifikation reserviert ist.
- Es dürfen keine Leerzeichen enthalten sein.

Praxistipps:

- Der Doppelpunkt sollte nur als Trennzeichen bei Namensräumen verwendet werden. Empfehlenswert ist auch, sich auf den ASCII-Zeichensatz zu beschränken und auf Umlaute oder Sonderzeichen zu verzichten.
- Die Maximallänge der Namen ist nicht definiert, aber mögliche Anwendungen, die auf die Daten zugreifen, können die Länge einschränken. Die Namen sollten also so kurz wie möglich sein und gleichzeitig die Art des Elements eindeutig kennzeichnen.
- In XML gibt es keine semantischen Tags und laut Standard können beliebige Elementtypen bestimmt werden. Der Entwickler sollte eine „natürliche Semantik“ benutzen, ähnlich der Namen beim Datenbankentwurf. Die Tags würden somit nicht nur die Struktur des Dokuments bestimmen, sondern zugleich den Inhalt des Dokumentes beschreiben.

Selbsttestaufgabe 3.2:

Welche der folgenden Elemente genügen *nicht* den Syntax-Regeln?

Würde das folgende XML-Dokument nach Korrektur der fehlerhaften Elemente wohlgeformt sein?

```
<br/>
<buch>
  < titel>
    Dies ist der Buchtitel
  </titel>
  <autor name = 'Nachname' name = "Vorname" / >
  <Inhalt>
    <Kapitel Nummer = '1'>
      <Unterkapitel Nummer = "1.3">
        Der Name des Unterkapitels 1.3
      </Unterkapitel>
    </Kapitel>
    <Kapitel Nummer = '2'>
      Der Name des zweiten Kapitels
    </Kapitel>
  </Inhalt>
</buch>
```

Entitäts-Referenzen

Da in XML einige Zeichen wie die spitzen Klammern eine besondere Bedeutung haben, gibt es für die Verwendung dieser Zeichen an anderen Stellen eines Elements einen Ersatz ähnlich wie in HTML, sogenannte Entitäts-Referenzen (engl. *entity references*):

- < steht für die öffnende spitze Klammer <,
- > steht für die schließende spitze Klammer >,
- & steht für das *Ampersand* oder „kaufmännische Und“ &,
- " steht für die doppelten Anführungszeichen „.,
- ' steht für das einfache Anführungszeichen bzw. den Apostroph ’..

Die Benutzung von < und & ist verpflichtend. Die Benutzung der anderen Referenzen ist dort optional, wo sie nicht zu Fehlern führen kann. Weitere Entitäts-Referenzen können in einer DTD definiert werden.

Kommentare

Ein Kommentar in einem XML-Dokument beginnt mit einer öffnenden spitzen Klammer, einem Ausrufezeichen und zwei Bindestrichen und er endet mit dem ersten Auftreten von zwei Bindestrichen und einer schließenden spitzen Klammer. Innerhalb des Kommentars sollten keine zwei Bindestriche auftreten, das letzte Zeichen vor dem Kommentatende darf kein Bindestrich sein. Ein Beispiel für einen Kommentar ist

```
<!-- Dies ist ein Kommentar -->
```

Ein Kommentar darf in den Zeichendaten eines Elements stehen oder vor oder hinter dem Wurzelement, jedoch nicht innerhalb von Tags, Deklarationen oder anderen Kommentaren.

XML-Parser können Kommentare entfernen oder durchlassen. Deshalb sollte keine Anwendung, die ein geparstes XML-Dokument verarbeitet, auf das Vorhandensein von Kommentaren angewiesen sein.

CDATA-Abschnitte

CDATA-Abschnitte dienen zur Einbettung von Quelltext größerem Umfangs in XML-Dokumente. Enthalten die Zeichendaten XML- oder HTML-Quelltext, müssen die vorbelegten Zeichen durch Entity-Referenzen ersetzt werden. Dies ist mühsam, da in Quelltext die Anzahl dieser Zeichen hoch sein kann. In einem CDATA-Abschnitt ist dies nicht notwendig. Er interpretiert alle Zeichen bis zu seinem Ende als Zeichendaten. Spitzeklammern beginnen keine Tags, ein Ampersand leitet keine Entity-Referenzen ein. Es gibt also keine Auszeichnungen innerhalb eines CDATA-Abschnitts.

Ein CDATA-Abschnitt beginnt mit <![CDATA[und endet mit dem ersten Auftreten von]]>. Die Zeichendaten, die innerhalb des CDATA-Abschnitts platziert sein sollen, dürfen also selbst die Zeichenkette]> nicht enthalten.

Sonderstellung von „]“

Steueranweisungen

Steueranweisungen in XML dienen ähnlich wie in HTML dazu, Daten an Anwendungen außerhalb des Standards zu übergeben. Eine Steueranweisung beginnt mit einer öffnenden spitzen Klammer, einem Fragezeichen und einem XML-Namen, dem Ziel. Eine Steueranweisung endet mit einem Fragezeichen und einer schließenden spitzen Klammer. Das Ziel bezeichnet die Anwendung, der der Inhalt der Steueranweisung übergeben werden soll. Der Inhalt der Steueranweisung kann ein beliebiger, beliebig langer Text sein, der nur vom Ziel abhängt. Beispielsweise kann mit dem Ziel php ein Programm in der Skriptsprache PHP an den PHP-Interpreter übergeben werden. Ein Beispiel für eine Steueranweisung, die im PHP-Interpreter die Ausgabe der Versionsnummer und anderer PHP-Konfigurationsdaten bewirkt, ist:

```
<?php phpinfo(); ?>
```

Steueranweisungen sind keine Elemente. Deshalb können sie wie Kommentare überall in einem XML-Dokument außerhalb eines Tags erscheinen.

XML-Deklarationen

Ein XML-Dokument sollte mit einer XML-Deklaration beginnen. Diese ist zwar nicht verpflichtend, aber unbedingt zu empfehlen, weil damit das Dokument sofort als XML-Dokument identifiziert werden kann. Falls die XML-Deklaration verwendet wird, muss sie das erste im Dokument sein. Selbst Kommentare oder Leerzeichen vor ihr sind nicht erlaubt. Der Grund hierfür ist, dass der XML-Parser anhand der ersten 5 Zeichen Mutmaßungen über die Codierung anstellt. Dies ist einfacher, wenn im Falle der Deklaration diese Zeichen fest sind.

Eine XML-Deklaration sieht aus wie eine Steueranweisung mit dem Ziel `xml`, ist allerdings aus Sicht des XML-Parsers keine Steueranweisung. Ein Beispiel ist

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

Das erste Attribut `version` ist verpflichtend. Die Attributwerte `1.0` und `1.1` sind momentan gültig. In der Praxis wird, außer in begründeten Ausnahmefällen, die Angabe `version="1.0"` benutzt. Dies könnte sich ändern, wenn es einmal eine weitere Version von XML geben sollte.

optionale Attribute und Default Werte

Die beiden anderen Attribute sind optional. Das Attribut `encoding` gibt dem XML-Parser Informationen über den verwendeten Zeichensatz, d. h. eigentlich über die verwendete Codierung. XML-Parser müssen nur die Zeichensätze UTF-8, dies ist eine Obermenge von ASCII, und UTF-16 unterstützen. Es werden aber mehr Zeichensätze unterstützt. Die XML-Spezifikation erwähnt explizit die in Tabelle 3.1 aufgeführten Zeichensätze. Fehlt das Attribut `encoding`, so wird UTF-8 angenommen. Das Attribut `standalone` kann als Werte nur `yes` oder `no` annehmen. Der Wert `yes` bedeutet, dass alle Informationen zum validierenden Parsing sich innerhalb des Dokuments befinden, d. h. dass das Dokument verschickt und auf einem Rechner ohne Netzanschluss von einem validierenden XML-Parser verarbeitet werden kann. Fehlt das Attribut, wird der Wert `no` angenommen. Der Wert kann auf `yes` gesetzt werden, wenn entweder keine Dokumententyp-Definition benutzt wird oder sich eine benutzte Dokumententyp-Definition vollständig innerhalb des XML-Dokuments befindet. Befindet sich die DTD ganz oder teilweise außerhalb des Dokuments, dann muss der Wert auf `no` gesetzt werden.

| Name | Erläuterung |
|--------|--|
| UTF-8 | Der Standardzeichensatz für XML-Dokumente. Unicode-Zeichensatz. Zeichen werden mit 1 bis 4 Byte codiert. ASCII-Dokumente sind gültige UTF-8 Dokumente. |
| UTF-16 | Ein Unicode-Zeichensatz. Zeichen werden mit 2 Byte codiert, Zeichen aus Unicode 3.1 werden mit 4 Byte |

| | |
|-----------------|---|
| | codiert. |
| ISO-10646-UCS-2 | Wie UTF-16, ohne 4-Byte-Zeichen. |
| ISO-10646-UCS-4 | Wie ISO-10646-UCS-2, Zeichen werden mit 4 Byte codiert. |
| ISO-8859-1 | Latin-1, also ASCII mit westeuropäischen Sonderzeichen. |
| ISO-8859-2 | Latin-2, also ASCII mit zentraleuropäischen Sonderzeichen. |
| ISO-8859-3 | Latin-3, also ASCII mit Zeichen für Esperanto, Maltisch, Türkisch, Galizisch. |
| ISO-8859-4 | Latin-4, also ASCII mit Zeichen für den baltischen Sprachraum. |
| ISO-8859-5 | ASCII mit kyrillischen Zeichen. |
| ISO-8859-6 | ASCII mit arabischen Zeichen. |
| ISO-8859-7 | ASCII mit modernem Griechisch. |
| ISO-8859-8 | ASCII mit Hebräisch. |
| ISO-8859-9 | Latin-5, entspricht bis auf wenige Zeichen Latin-1. |
| ISO-8859-10 | Latin-6, also ASCII mit Zeichen der nordeuropäischen Länder. Entspricht weitgehend Latin-4. |
| ISO-8859-11 | ASCII mit Zeichen der Thai-Sprache. |
| ISO-8859-12 | Zeichensatz zurzeit nicht existent. |
| ISO-8859-13 | Entspricht weitgehend Latin-6. |
| ISO-8859-14 | Latin-8, eine Variante von Latin-1 mit gälischen und walisischen Buchstaben. |
| ISO-8859-15 | Latin-9, weitgehend identisch zu Latin-1, enthält das Eurosymbol. |
| ISO-2022-JP | Der japanische Zeichensatz JIS X-0208-1997 in 7-Bit Codierung. |
| Shift-JIS | Der japanische Zeichensatz JIS X-0208-1997 von Microsoft Windows. |
| EUC-JP | Der japanische Zeichensatz JIS X-0208-1997 von UNIX. |

Tabelle 3.1 In der XML-Spezifikation genannte Codierungen

Selbsttestaufgabe 3.3:

Studieren Sie das XML-Dokument in der Datei [selbsttest-3-2.xml](#). Ist es wohlgeformt?

Besorgen Sie sich einen XML-Parser. Prüfen Sie mit dem XML-Parser, ob `selbsttest-3-2.xml` wohlgeformt ist. Falls nicht, korrigieren Sie die Fehler einen nach dem anderen und parsen Sie solange erneut, bis das XML-Dokument wohlgeformt ist.

Dokumenttyp-Definitionen

Ein wohlgeformtes XML-Dokument erfüllt die XML-Syntax. Trotzdem kann das Dokument wertlose Daten in Bezug auf die gedachte Anwendung enthalten. So wird zum Beispiel bei der Prüfung auf Wohlgeformtheit nicht untersucht, ob ein Element überhaupt Attribute haben sollte. Das kann der XML-Parser ohne zusätzliche Informationen auch gar nicht, da er die Semantik der Ziel-Anwendung nicht kennt. Solche Informationen werden dem Parser durch eine Dokumententyp-Definition zur Verfügung gestellt.

Von der Seite der Anwendung aus betrachtet muss diese auch nicht notwendigerweise alle wohlgeformten XML-Dokumente verarbeiten können. Eine Grafik-Anwendung, in die ein SVG-Dokument importiert werden soll, sollte in diesem Dokument nur die Elementstrukturen finden, die in SVG definiert sind und nicht etwa XHTML-Elemente. Aus dieser Sicht wirkt der validierende Parser wie ein Filter, der alle „anwendungsfremden“ Dokumente abweist, d. h. Dokumente, die dem zur Anwendung gehörenden schematischen Modell nicht genügen. Das Modell enthält Regeln, die bestimmen, welche Elemente und Attribute erforderlich oder zulässig sind, welche Eigenschaften diese Elemente haben müssen oder können und wie Element und Attribute innerhalb des Dokuments verwendet werden.

Entspricht ein Dokument diesem Modell, wird es als Instanz des Modells bezeichnet, welches die eigentlichen in Struktur gebetteten Informationen enthält. Das ist vergleichbar mit dem objektorientierten Paradigma der Beschreibung von Objekten durch eine Klasse, die eine Menge von möglichen Objekten gleichen Typs vorgibt.

Noch ein Hinweis: Wird gegen solche existierenden Modelle validiert, besteht die Gültigkeit des Dokumentes darin, dass es den hinterlegten Regeln entspricht. Die Gültigkeit sagt in diesem Zusammenhang nichts über die Richtigkeit des Inhaltes im XML-Dokument aus. Ein Element <fernuniversitaet> kann korrekt aus den Unterelementen <fakultaet>, <lehrgebiet> und <hochschullehrer> zusammengesetzt sein, dennoch können Lehrgebiete und Fakultäten falsch zugeordnet sein.

Struktur einer Dokumententyp-Definition

Die Deklaration einer Dokumententyp-Definition muss innerhalb eines XML-Dokumentes direkt nach einer Deklaration platziert werden. Dazwischen dürfen sich lediglich Steueranweisungen oder Kommentare befinden. Die Dokumententyp-Definition selbst hat einen inneren oder einen externen Teil oder beides. Der innere Teil besteht aus Festlegungen, die sich im XML-Dokument selbst befinden. Der externe Teil besteht aus Festlegungen, die sich in einer anderen Datei befinden und über einen URI referenziert werden. Dies erlaubt, standardisierte DTDs (wie z. B. die für XHTML) an einer zentralen Stelle im World Wide Web einheitlich zur Verfügung zu stellen. Enthält ein XML-Dokument eine DTD mit einem externen Teil, so ist in der XML-Deklaration das Attribut standalone auf den Wert no zu setzen.

URI
URL
URN

Ein Uniform Resource Identifier (URI) ist entweder ein Uniform Resource Locator (URL) oder ein Uniform Resource Name (URN), wobei letztere außer in einigen Nischenanwendungsdomänen noch keine weite Verbreitung gefunden haben, sodass derzeit in der Regel URI und URL pragmatisch betrachtet als weitestgehend synonym gesehen werden können.

Die Syntax der Deklaration einer DTD mit dem Wurzelement `buch`, die das XML-Dokument aus Selbsttestaufgabe 3.1 beschreiben soll, ist

```
<!DOCTYPE buch SYSTEM "http://www.kurs1873.de/buch.dtd">
```

bei einem externen Teil und

```
<!DOCTYPE buch [  
  <!ELEMENT buch (titel,autor,Inhalt)>  
  <!ELEMENT titel (#PCDATA)>  
  <!ELEMENT autor EMPTY>  
  <!ELEMENT Inhalt (Kapitel*)>  
  <!ELEMENT Kapitel (#PCDATA | Unterkapitel)*>  
  <!ELEMENT Unterkapitel (#PCDATA)>  
  <!ATTLIST autor name CDATA #REQUIRED>  
  <!ATTLIST Kapitel Nummer CDATA #REQUIRED>  
  <!ATTLIST Unterkapitel Nummer CDATA #REQUIRED>  
>]
```

bei einem internen Teil. Hat die DTD beide Teile, so muss der externe Teil zuerst deklariert werden. Beim inneren Teil können beliebig viele Regeln zwischen den eckigen Klammern definiert werden. Dabei dürfen aber lediglich Entity-Deklarationen umdefiniert werden.

Anmerkung: Zum Referenzieren eines externen Teils kann auch das Schlüsselwort PUBLIC verwendet werden.

Das Format des externen Teils – in diesem Fall der Inhalt der Datei buch.dtd – entspricht dem des internen Teils zwischen den eckigen Klammern. Da der externe Teil sich in einer eigenen Datei befindet, muss zu Beginn dieser Datei eine XML-Deklaration stehen, falls der in dieser Datei verwendete Zeichensatz nicht UTF-8 ist.

Die Regeln definieren zunächst die Elemente, die es gibt und ihre Hierarchie. Danach werden die Attribute für die Elemente definiert. Die Definition innerhalb einer DTD ist abschließend, d. h. alles, was nicht definiert ist, ist verboten. Daraus folgt, dass ein Dokument, welches keine DTD enthält, von einem validierenden XML-Parser als ungültig eingestuft wird.

Deklaration von Elementen

Die Deklaration eines Elements beginnt mit einer spitzen Klammer, einem Ausrufezeichen und dem Wort ELEMENT, gefolgt vom Namen des Elementes, seinem Inhaltsmodell und einer schließenden spitzen Klammer. Ein Beispiel mit einem Inhaltsmodell, welches nur Zeichendaten zulässt ist:

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

Hierbei steht #PCDATA für parsed character data. Gemeint ist hier Zeichentext, der lediglich noch Entitäts-Referenzen beinhalten darf, aber keine Auszeichnung.

Elemente, die keinen Inhalt haben, werden als leere Elemente bezeichnet. Diese haben das Inhaltsmodell EMPTY. Dies kann sinnvoll sein, wenn ein Element nur Attribute hat. Elemente, die einen beliebigen Inhalt haben können, haben das Inhaltsmodell ANY. Bei solchen Elementen wird nur festgelegt, dass es sie überhaupt in einem für diese DTD gültigen XML-Dokument geben darf.

Im Inhaltsmodell dürfen auch wiederum Elementnamen möglicher Kindelemente vorkommen. Sind sie durch ein Komma getrennt, dann bilden sie eine geordnete Liste. Sie können aber auch durch das Auswahlzeichen | getrennt sein, dann ist eins dieser Elemente ein mögliches Kindelement. Damit ein solcher Ausdruck eindeutig bleibt, werden Klammern benutzt. So bedeutet

```
<!ELEMENT buch (titel,autor,Inhalt)>
<!ELEMENT Kapitel (kaptitel | Unterkapitel)>
```

dass das Element buch drei Kindelemente mit den angegebenen Namen in der angegebenen Reihenfolge hat und dass das Element Kapitel entweder ein Kindelement kaptitel oder ein Kindelement Unterkapitel hat.

Elemente oder Unterausdrücke können von einem Bezeichner für eine Anzahl gefolgt sein: ? bedeutet ein- oder keinmal, * bedeutet keinmal, einmal oder mehrmals, + bedeutet einmal oder mehrmals. So bedeutet:

```
<!ELEMENT buch (titel,(autor|editor)+,Inhalt)>
```

dass das Element buch zunächst ein Kindelement titel hat, danach ein oder mehrere Kindelemente, die jeweils entweder autor oder editor sind, und zuletzt ein Kindelement Inhalt.

Das einzige Inhaltsmodell, bei dem Zeichendaten und Kindelemente gemischt auftreten dürfen, ist eine Auswahl, bei der #PCDATA an erster Stelle stehen muss, mit weiteren Kindelementen und mit dem * für beliebige Anzahl versehen. Ein Beispiel ist

```
<!ELEMENT Kapitel (#PCDATA | Unterkapitel)*>
```

Eine Konsequenz dieser Beschränkung ist, dass es bei Elementen, die auch Zeichentext beinhalten können, nicht möglich ist, Kindelemente in einer festen Reihenfolge oder in bestimmter Anzahl vorzusehen.

Generell haben DTDs den Nachteil, dass sie kein hinreichendes Typsystem unterstützen. Eine Eingrenzung von Inhalten über Zeichentext hinaus ist nicht möglich.

Selbsttestaufgabe 3.4:

Definieren Sie ein DTD-Element telefonnummer, das aus den Elementen *landesvorwahl*, *vorwahl* und *nummer* in genau dieser Reihenfolge besteht.

Deklaration von Attributen

Attribute können nur für bereits deklarierte Elemente deklariert werden. Eine Attributdeklaration beginnt mit einem Kleiner- und einem Ausrufezeichen, dem Schlüsselwort ATTLIST und dem Namen des Elements, für das Attribute definiert werden sollen. Danach folgen vor dem Größer-Zeichen ein oder mehrere Attribute, bestehend jeweils aus dem Attributnamen, dem Attributtyp und eventuell einem Standardwert. Ein Beispiel ist

```
<!ATTLIST autor name CDATA #REQUIRED>
```

Hier wird für das bereits deklarierte Element autor das Attribut name definiert. Der Typ des Attributwerts ist CDATA und der Standardwert ist #REQUIRED.

Zunächst gibt es vier Arten von Standardwerten:

- #IMPLIED Dieses Attribut ist optional.
- #REQUIRED Dieses Attribut ist verpflichtend.
- #FIXED Wert Der Wert dieses Attributs ist fest und wird hinter dem Schlüsselwort FIXED angegeben. Dieses Attribut muss bei der Nutzung des betreffenden Elements nicht angegeben werden. Wird es aber angegeben, dann muss der dort benutzte Wert dem Standardwert entsprechen.
- Wert Der Standardwert wird in Anführungszeichen direkt angegeben.

Zudem gibt es zehn Attributtypen:

1. CDATA Dies ist der ganz allgemeine Attributtyp. Jeder Zeichentext, der die Wohlgeformtheit des Attributwertes nicht verletzt, ist erlaubt. CDATA steht für *character data*.
2. NMTOKEN Dies entspricht einem XML-Namen, außer dass alle Zeichen als Beginn des Namens erlaubt sind. NMTOKEN steht für *XML-Namens-Token*.
3. NMOKENS Dieser Wert besteht aus einem oder mehreren XML-Namens-Tokens, die durch Leerzeichen getrennt sind.

4. **ID** Dies bedeutet, dass der Attributwert im XML-Dokument eindeutig sein muss in dem Sinne, dass kein anderes Attribut mit Attributtyp ID im gleichen XML-Dokument den gleichen Wert haben darf. Der Attributwert muss ein XML-Name sein.
5. **IDREF** Der Wert dieses Attributs verweist auf ein anderes Attribut vom Typ ID. Der Wert muss deshalb ein XML-Name sein, der in dem anderen Attribut ebenfalls als Wert vorkommt.
6. **IDREFS** Der Wert besteht aus einer Liste von IDREF-Werten, die durch Leerzeichen getrennt sind.
7. **ENTITY** Der Wert enthält den Namen einer nicht vom XML-Parser ersetzen Entität, ein Beispiel wäre der Name einer Datei mit Bilddaten.
8. **ENTITIES** Der Wert besteht aus einer Liste von ENTITY-Werten, die durch Leerzeichen getrennt sind.
9. **NOTATION** Der Wert eines Attributs dieses Typs ist der Name einer Notation, die mittels der Anweisung `<!NOTATION name "Wert">` in der DTD deklariert wurde. Wir werden in diesem Kurs nicht weiter auf Notationen eingehen.
10. **Aufzählung** Anstelle eines Schlüsselworts folgt hier, in Klammern und durch Striche | getrennt, eine Liste von möglichen Werten, wobei jeder Wert ein XML-Namens-Token sein muss.

Selbsttestaufgabe 3.5:

Schreiben Sie eine einfache externe DTD in einer Datei rezept.dtd.

Deklarieren Sie dazu ein Element *rezept*, das als Kindelement *hauptzutat* und eine oder mehrere weitere Elemente *zutat* hat.

Alternativ zu *zutat* kann auch einmal *dressing* enthalten sein. Die Reihenfolge der Elemente ist beliebig.

Deklarieren Sie die Elemente *hauptzutat*, *zutat* und *dressing* jeweils mit dem Inhaltsmodell #PCDATA.

Deklaration von Entitäten

Bereits vorgestellt wurden die Entitäts-Referenzen, die von XML für die Benutzung des Ampersand und anderer Sonderzeichen zur Verfügung gestellt werden. Mit Entitäts-Deklarationen können darüber hinaus weitere Entitäts-Referenzen definiert werden. Das Format einer Entitäts-Deklaration ist einfach:

```
<!ENTITY Name "Wert">
```

Die Anwendung der Entität im XML-Dokument ist &Name;. Wert kann dabei ein beliebiger wohlgeformter Text sein, d. h. auch Auszeichnungen und Entitäten enthalten. Der Begriff wohlgeformter Text meint hierbei, dass nach der Ersetzung der Referenz &Name; durch den Text das XML-Dokument wohlgeformt sein muss.

Soll eine Entität mit einem größeren Text benutzt bzw. Text an mehreren Stellen benutzt werden, dann kann dieser Text in einer eigenen Datei untergebracht werden. In der Entitäts-Referenz wird darauf verwiesen mittels:

```
<!ENTITY Name SYSTEM "URI">
```

wobei URI den Verweis auf die Datei enthält. Der Inhalt einer solchen Datei heißt extern geparse Entität. Die Anwendung einer Referenz auf eine extern geparse Entität ist in Attributwerten verboten. Weiterhin hat ein XML-Parser die Freiheit zu entscheiden, ob er die Referenz durch die externe geparse Entität ersetzt oder nicht. Ein validierender Parser wird es tun, ein nicht-validierender Parser muss es nicht tun. Eine extern geparse Entität enthält eventuell auch Informationen über den verwendeten Zeichensatz.

Oft haben viele Elemente oder Attribute das gleiche Inhaltsmodell. In diesem Fall ist es sinnvoll, das Inhaltsmodell nur einmal zu definieren und in den Element- bzw. Attribut-Deklarationen darauf zu referenzieren. Hierzu dienen Parameter-Entitäten. Die Syntax ist ähnlich der von Entitäts-Deklarationen, allerdings steht vor dem Namen der Entität noch ein Prozent-Zeichen. Ein Beispiel ist

```
<!ENTITY % buchinhalt
```

```
"titel,autor,Inhalt">
<!ELEMENT buch (%buchinhalt)>
```

Parameter-Entitäten sind auch deshalb hilfreich, weil sie umdefiniert werden können. Benutzt ein XML-Dokument eine DTD mit einem externen und einem internen Teil, dann kann im internen Teil eine Parameter-Entität des externen Teils umdefiniert werden. So könnte zum Beispiel ein vierter Kindelement von buch definiert werden, das nur in diesem speziellen XML-Dokument benötigt wird. Anwendungen von Parameter-Entitäten finden sich auch bei der Verwendung von Namensräumen.

Codierung

In Kurseinheit 1 wurde bereits beschrieben, wie Zeichensätze aufgebaut sind und welche Mächtigkeit speziell Unicode-Zeichensätze haben. In diesem Abschnitt soll speziell auf Fragen eingegangen werden, die sich beim Verwenden mehrerer verschiedener Zeichensätze bzw. Sprachen im Zusammenhang mit einem Dokument stellen.

Textdeklarationen

Dokumente bestehen oft aus mehreren Dateien, wie etwa DTDs und dort externe geparse Entitäten. Dabei ist nicht auszuschließen, dass die verschiedenen Dateien verschiedene Zeichensätze benutzen. Falls Zugriff auf alle diese Dateien besteht, kann umcodiert werden. Ansonsten gibt es die Möglichkeit, für jede Datei mittels einer Text-Deklaration den verwendeten Zeichensatz anzugeben. Die Text-Deklaration muss ganz am Anfang einer Datei stehen. Sie hat die gleiche Form wie die XML-Deklaration, allerdings fehlt das Attribut standalone und das Attribut version kann fehlen. Ein Beispiel für eine Text-Deklaration in einem DTD-Fragment ist

```
<?xml encoding = "ISO-8859-1"?>
<!ELEMENT titel (#PCDATA)>
```

Konvertierung

Zur Konvertierung einer Datei von einem Zeichensatz in einen anderen steht eine Reihe von Möglichkeiten zur Verfügung, von denen wir hier drei vorstellen wollen.

Steht der Editor zur Verfügung, mittels dessen die betreffende Datei erstellt worden ist, so ist zu prüfen, ob die Datei auch im gewünschten Zielformat abgespeichert werden kann. Mit Microsoft Word 2000 können zum Beispiel reine Textdateien in Unicode gespeichert werden. Unter UNIX liegen Textdateien in der Regel in einem der ISO Zeichensätze vor.

Zusätzlich gibt es das Programm [recode](#)¹, welches als Open Source Werkzeug vorliegt. Bei Benutzung des Java Development-Kit kann das dortige Werkzeug native2ascii benutzt werden, um eine Datei aus einem fremden Zeichensatz zunächst in Javas spezielles

¹ <https://github.com/pinard/Recode>

Format zu konvertieren und sie dann in einem zweiten Lauf aus diesem Format in das gewünschte Zielformat zu konvertieren.

Zeichen können auch durch ihre Nummer im Zeichensatz unter Voranstellen des Amper-sand und des Doppelkreuzes &# mit abschließendem Semikolon eingegeben werden. Wird die Nummer hexadezimal angegeben, wird sie von einem x angeführt. Solche Zei-chenreferenzen können als Elementinhalt, in einem Attributwert oder in einem Kom-mentar verwendet werden. Sie können jedoch nicht in einem XML-Namen, dem Ziel einer Steueranweisung oder Schlüsselworten erscheinen. Im Beispiel:

```
<titel>
The &#x3BB;-Calculus
</titel>
```

steht die Zeichenreferenz λ für das griechische Zeichen Lambda λ. Die Verwen-dung bestimmter, häufiger genutzter Zeichenreferenzen kann Dokumente unleserlich und die Eingabe aufwändig machen. Abhilfe schafft in diesem Fall die Verwendung von Entity-Referenzen in einer DTD. Nach Einfügen von

```
<!ENTITY lambda "&#x3BB;">"
```

in die DTD zu obigem Dokument kann der Titel wie folgt geschrieben werden:

```
<titel>
The &lambda;-Calculus
</titel>
```

Natürlich müssen solche Listen von Entitäts-Referenzen nicht immer wieder vollständig neu erstellt werden. Die [DTD zu XHTML](#)² enthält nützliche Listen von Entitäten zu Latin-1-Zeichen, Sonderzeichen sowie griechischen und mathematischen Zeichen. Solche Lis-ten können entweder kopiert und in eigene DTDs eingebaut werden. Speziell, wenn sie anerkannt und somit längerfristig unter einer bestimmten URI verfügbar sind, können sie auch als externe Entitäten referenziert werden.

XML-Dokumente eignen sich auf Grund der Verwendung von Unicode auch für die Speicherung mehrsprachiger Daten. In diesem Fall ist es für eine weiterverarbei-tende Software oft hilfreich zu wissen, in welcher Sprache der Inhalt eines Elements verfasst ist. Zum Beispiel kann ein Programm zur Rechtschreibprüfung ohne diese Sprachinformation nicht sinnvoll arbeiten.

verwendete Sprachen

²http://www.edition-w3.de/TR/2001/REC-xhtml-modularization-20010410/#a_module_XHTML_Latin_1_Character_Entities

Zur Bereitstellung von Informationen über die verwendete Sprache gibt es das Attribut `xml:lang`. Der Wert des Attributs sollte einer der Codes aus zwei Buchstaben sein, die in [ISO 639.2](#)³ definiert sind. Für Sprachen, die nicht in dieser Liste aufgeführt sind, lohnt sich eine Suche bei den bei der [IANA](#)⁴ registrierten Identifikatoren. Ansonsten wird der Code eben selbst definiert. Zusätzlich können einem Sprachcode beliebig viele Subcodes folgen, die innerhalb einer Sprache eine bestimmte Region kennzeichnen. Der Sprachcode und die Subcodes werden jeweils durch einen Bindestrich getrennt. Der erste Subcode sollte aus zwei Großbuchstaben gemäß ISO 3166 bestehen, siehe URL

http://www.iso.org/iso/home/standards/country_codes.htm.

Im Folgenden ist ein Beispiel für einen Titel in deutscher Sprache

```
<titel xml:lang = "de">
  Dies ist der Titel
</titel>
```

Verwendung von
Aufzählungstypen

Soll `xml:lang` benutzt werden, so erfolgt die Deklaration in einer DTD. Je nach Verbindlichkeit kann `#IMPLIED`, `#REQUIRED` oder `#FIXED` benutzt werden. Als Typ kann `NMOKEN` verwendet werden, da alle Codes gültige XML-Namen sind. Es kann auch ein Aufzählungstyp verwendet werden, wenn die Anzahl der zu verwendenden Sprachen eingeschränkt ist oder wenn die Verwendung gültiger Codes sichergestellt werden soll. Ein Parser prüft von sich aus nicht, ob die Werte des Attributs `xml:lang` gültige Codes sind. Dies ist eine Folge des unzureichenden Typsystems von DTDs.

Selbsttestaufgabe 3.6:

Deklarieren Sie die Attribute eines bereits deklarierten Elements `Name`.

Es soll ein optionales Attribut `Alter` geben, ein Attribut `Form` mit festem Wert `Nix` und ein verpflichtendes Attribut `Kundentyp` mit den möglichen Werten `Gut`, `Schlecht` und `Unbekannt`.

Einfache XML-Schemasprachen

Die Schema-Sprachen werden zunächst auf ihre Ausdrucksstärke betrachtet. Darüber hinaus werden die Aufgaben, welche für XML-Schemata anfallen, aufgezeigt und die Komplexität der zugehörigen Algorithmen erläutert. Durch die Flexibilität, die viele Anwendungen XML abfordern, ist es zunehmend notwendig ausdrucksfähigere Erweiterungen und zusätzlich Spezifikationen für den verwendeten XML-Code zu ermöglichen. Hierzu werden Schema-Sprachen verwendet, welche die XML-Struktur selbst und deren

³ <http://lcweb.loc.gov/standards/iso639-2/langhome.html>

⁴ <http://www.iana.org/assignments/language-tags>

erlaubte Anwendungsformen definieren. Im Folgenden werden einfache Schema-Sprachen, wie sie durch Dokument-Typ-Definitionen und spezialisierte DTD gegeben sind, näher beschrieben und auf ihre Ausdrucksstärke und die Komplexität ihrer Algorithmen hin untersucht.

Schemasprachen dienen der Validierung. Validiert wird, ob ein Dokument einem gegebenen Typ und damit einer bestimmten Struktur und Reihenfolge entspricht. Zu den XML-Schema-Sprachen gehören:

- Document Type Definition – kurz: DTD,
- XML-Schema Definition – kurz: XSD,
- regular language description for XML New Generation (RELAX NG),
- ISO Schematron.

Mit jeder dieser Sprachen können Regeln festgelegt werden, die durch XML-Dokumente einzuhalten sind. Da alle XML-Dokumente eine Baumstruktur besitzen, gilt auch, dass mit den obigen Sprachen Schemata für Baumsprachen erzeugt werden können. Um die Struktur von XML-Dokumenten zu beschreiben, wird sich sogenannter Schemasprachen bedient. Mit Schemasprachen wird eine Grammatik der Regeln für eine Klasse von XML-Dokumenten definiert. Von den oben bereits erwähnten Schemasprachen sind die zwei bekanntesten DTD und XSD.

Ein Schema ist allgemein zunächst ein formales Modell der Struktur von Daten. Wenn die Daten in einer Datenbank abgelegt werden, geschieht dies nach einem Datenbankschema. Üblicherweise ist das Schema selbst in einer formalen Sprache definiert, sodass sich Daten automatisch darauf überprüfen lassen, ob sie dem Schema genügen. Für XML-Daten gibt es dazu beispielsweise den XML-Schema-Standard. Schemata sind nicht zu verwechseln mit Datenstrukturen, die sich auf die interne Speicherung beziehen und Datenformaten, einer konkreten Form der Speicherung. Schemata können hinsichtlich ihrer Komplexität von einfachen Attributlisten bis zu komplexen Ontologien reichen.

Ein XML-Schema beschreibt Einschränkungen der Struktur und des Inhalts zusätzlich zu den grundsätzlichen Beschränkungen von XML an sich. Beispielsweise werden durch ein XML-Schema zulässige Namen sowie Strukturen für Elemente und Attribute definiert. Ein XML-Schema definiert also genau, wie zulässige XML-Dokumente aussehen müssen, sodass Computer-Programme automatisch durch Validierung feststellen können, ob ein bestimmtes XML-Dokument durch ein Schema zugelassen ist. Darüber hinaus kann ein Schema als Grundlage zur Erstellung von Anwendungsprogrammen verwendet werden z. B. durch Generierung von Code-Rahmenwerken.

Deshalb spielen Schemata eine sehr wichtige Rolle bei der Entwicklung XML-basierter Anwendungen. Ein XML-Schema wird durch eine XML-Schemasprache

XML-Schemasprachen

definiert. XML-Schemasprachen lassen sich als eingeschränkte reguläre Baumsprachen auffassen.

4. Der DTD und XSD

Die Regeln zur strukturellen Definition eines Dokumenttyps bilden zusammen eine Dokumentgrammatik (Einführung in Grammatiken siehe [S08]), die als Dokumenttyp-Definition bezeichnet wird. DTD war die erste Schemasprache für XML-Dokumente und wurde als Bestandteil der XML-Spezifikation 1.0 vom W3C veröffentlicht. Leider weisen DTDs einige Defizite auf. Ihre Darstellungsmöglichkeiten sind begrenzt und die Syntax der DTDs ist keine XML-Syntax. Diese Defizite führten dazu, dass die Schemabeschreibungsmethode XML-Schema entwickelt wurde. Beide Varianten zur strukturellen Definition von Dokumentengrammatiken werden wir nun im weiteren Verlauf der Kurseinheit etwas näher kennenlernen.

DTD

Eine DTD ist eine Komponente innerhalb einer XML-Umgebung. Sie definiert die syntaktischen Regeln, nach denen ein Dokument erstellt bzw. kreiert werden soll. Durch die Verwendung einer DTD und eines XML-Prozessors kann ein Dokument gegen die DTD validiert/geprüft werden, d. h. es kann daraufhin getestet werden, ob es mit einer gegebenen DTD übereinstimmt. DTDs verwenden reguläre Ausdrücke zur Beschreibung der Folge der Kinder eines Knotens.

regular expression

Ein regulärer Ausdruck (engl *regular expression*) ist dabei eine Zeichenkette, die der Beschreibung von Mengen beziehungsweise Untermengen von Zeichenketten mithilfe bestimmter syntaktischer Regeln dient. Allgemein gesprochen stellen reguläre Ausdrücke also eine Art Filterkriterium für Texte dar, in dem der Ausdruck in Form eines Musters mit dem Text abgeglichen wird. So ist es beispielsweise möglich, alle Wörter, die mit dem Buchstaben „S“ beginnen und mit dem Buchstaben „D“ enden, zu finden, ohne die zwischenliegenden Buchstaben vorgeben zu müssen.

Dokument-Type-Definitionen bieten somit die Möglichkeit die Bedingungen für XML-Dokumente festzulegen. Sie verwenden hierzu eine eigene Sprache, welche es ermöglicht die Elemente eines Dokuments mit ihren Attributen festzulegen und zudem deren Struktur zueinander zu beschreiben. DTDs verwenden reguläre Ausdrücke zur Beschreibung der Folge der Kinder eines Knotens.

Beispiel-DTDs

```
<! DOCTYPE WM [
<! ELEMENT WM (Austragungsland, Jahr, Teams, Gewinner)>
<! ELEMENT Teams (Team*)>
<! ELEMENT Team (Land, Platz, Spieler*)>
```

```
<! ELEMENT Spieler (Name, Alter, Größe, Spielertyp,
AnzTore)
...
]>
```

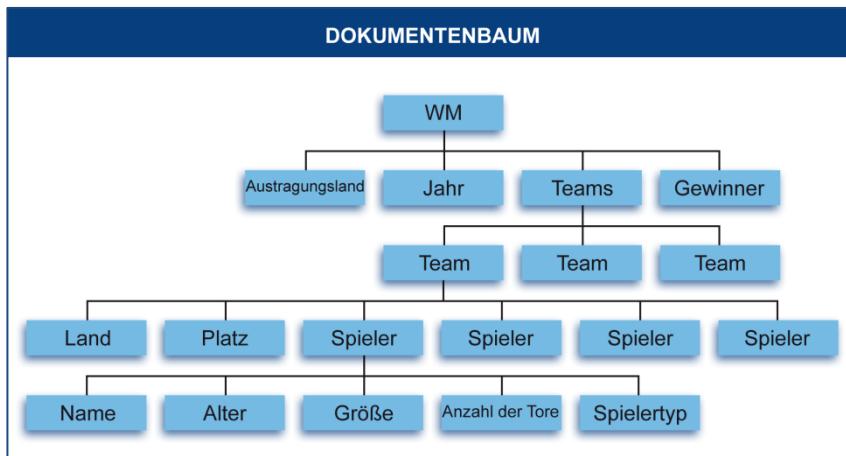


Abbildung 4.1 Beispiel-Dokument zur Beispiel-DTD

Eine aus SGML übernommene Einschränkung für den Aufbau einer DTD gibt es allerdings. Die regulären Ausdrücke in der DTD müssen eindeutig sein und können somit nicht mehrfach belegt werden. Darüber hinaus besteht eine weitere Schwäche von DTDs darin, dass Elemente mit gleichem Namen innerhalb der Struktur nicht an zwei unterschiedlichen Stellen verwendet werden können, obwohl es in einigen Anwendungsfällen strukturell sinnvoll und wünschenswert wäre.

Eine Einschränkung zur effizienten Auswertung von DTDs ist die Eindeutigkeit (engl. *one unambiguity*). Falls eindeutig festgestellt werden kann, welches Symbol eines regulären Ausdrucks mit einem Symbol des Eingabeworts übereinstimmt, so ist dieser reguläre Ausdruck eindeutig (engl. *unambiguous*). Hierbei wird die Kenntnis des gesamten Wortes angenommen. Kann die Eindeutigkeit weiterhin auch festgestellt werden, ohne den Rest des Wortes nach dem entsprechenden Symbol zu kennen, so ist dieser reguläre Ausdruck ein-eindeutig.

It is an error if an element in the document can match more than one occurrence of an element type in the content model [without looking ahead]“ [XML 1.0, W3C recommendation Oct. 2000]

Das bedeutet, dass alle Ausdrücke in der DTD ein-eindeutig sein müssen. Sie dürfen höchstens einmal vorkommen. Hier wird nun anhand eines weiteren Beispiels aus [K05] der Aufbau eines XML-Dokuments und einer dazu passenden DTD noch einmal ausführlich dargestellt. In diesem Beispiel findet sich ein XML-Dokument, mit dem ein Bestand von Personen eingerichtet werden soll. Dabei besitzt jede Person folgende Ele-

mente: Vorname, Nachname, Alter, E-Mail-Adresse und, falls die Person studiert, die Matrikelnummer. Das dazugehörige XML-Dokument sieht wie folgt aus:

```
<Bestand>
  <Person>
    <Vorname>Tim</Vorname>
    <Nachname>Müller</Nachname>
    <Alter>23</Alter>
    <E-Mail>tim@bsp.de</E-Mail>
    <Matrikel>3450678</Matrikel>
  </Person>
  <Person>
    <Vorname>Laura</Vorname>
    <Nachname>Schmidt</Nachname>
    <Alter>25</Alter>
    <E-Mail>laura@bsp.de</E-Mail>
  </Person>
  <Person>. . .</Person>
</Bestand>
```

Eine mögliche passende DTD zu dem obigen XML-Dokument ist:

```
<!DOCTYPE Bestand [
  <!ELEMENT Bestand (Person*)>
  <!ELEMENT Person (Vorname, Nachname, Alter, E-Mail,
Matrikel?)>
  <!ELEMENT Vorname (#PCDATA)>
  <!ELEMENT Nachname (#PCDATA)>
  <!ELEMENT Alter (#PCDATA)>
  <!ELEMENT E-Mail (#PCDATA)>
  <!ELEMENT Matrikel (#PCDATA)>
]>
```

Die erste Zeile der DTD besagt, dass *Bestand* das Wurzel-Element sein soll. Die zweite Zeile ist für eine beliebige Anzahl von *Person*-Elementen zuständig. Jedes *Person*-Element besteht aus den Elementen *Vorname*, *Nachname*, *Alter*, *E-Mail* und vielleicht *Matrikel*. Dies wird in der dritten Zeile deutlich. Die Elemente *Vorname*, *Nachname*, *Alter*, *E-Mail* und *Matrikel* enthalten nur normalen Text, der auch als *#PCDATA* bezeichnet wird. Ein Element kann auch Attribute besitzen. Diese tauchen im Start-Tag eines Elements auf. Besäße zum Beispiel das *Person*-Element in unserem Bestandsdokument Staatsangehörigkeit als Attribut, so wäre dies im Dokument durch die folgende Zeile darzustellen:

```
<Person Staatsangehörigkeit='...'>
```

Wichtige Eigenschaften, die ein XML-Dokument charakterisieren, sind die folgenden:

- Ein XML-Dokument heißt zulässig, valide oder gültig bezüglich eines XML-Schemas, falls es dieses Schema erfüllt.
- Ein Dokument heißt wohlgeformt, falls alle Start- und End-Tags richtig und der XML-Syntax entsprechend angeordnet sind und alle Element-Attribute eindeutig sind.
- Die Wohlgeformtheit eines Dokuments garantiert, dass das Dokument durch einen XML-Parser geparsed werden kann.

Validität
Parse
Wohlgeformtheit

Schwächen von DTDs

Das folgende Beispiel zeigt eine DTD mit dem Element ad, das zwei unterschiedliche Strukturen besitzt.



Abbildung 4.2 DTD Beispiel aus [S05]

Die Intention ist: je nach Kontext wird der entsprechende Typ von ad gewählt. Dies ist gerade die Schwäche von einfachen DTDs, da es bei diesen nicht möglich ist, Typen für Elemente zu definieren.

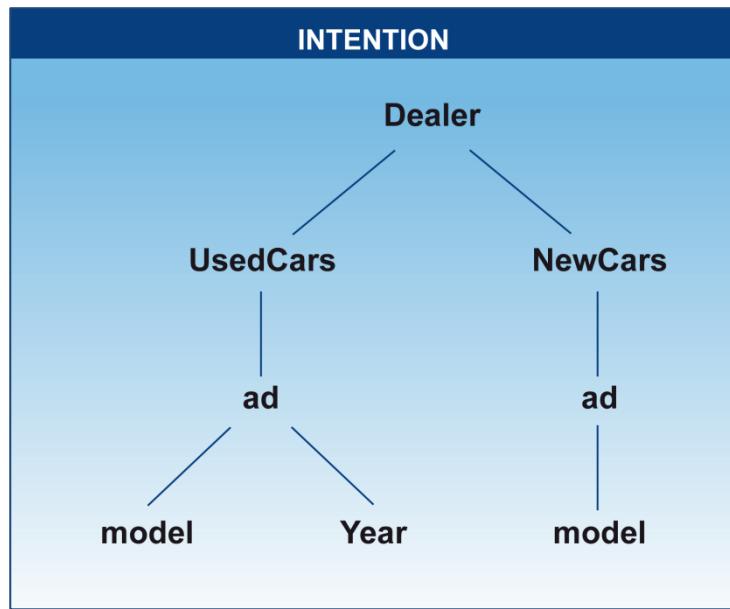


Abbildung 4.3 Entwurf aus [S05]

Entwickler, die häufig mit Datenbanken zu tun haben, vermissen aber nicht nur die präzise Unterscheidung von Datentypen, sondern auch die Festlegung von erlaubten Wertebereichen, eine Kontrolle über bestimmte Zeichenmuster oder die Vorgabe von Standardwerten etc. Die Einschränkungen der DTDs führen auch dazu, dass ein hoher Programmieraufwand betrieben werden muss, um die Eingabe korrekter Daten zu erzwingen.

XSDs

Wie bereits erwähnt weisen DTDs einige Defizite auf. Ihre Darstellungsmöglichkeiten sind begrenzt und die Syntax der DTDs ist keine XML-Syntax. Diese Defizite führten dazu, dass die Schemabeschreibungsmethode XML-Schema entwickelt wurde. Um XML-Schemata zu definieren, die eine größere Ausdrucksstärke besitzen als Schemata, die durch DTDs definiert werden, sind XML-Schema-Definitionen (engl. *XML schema definitions, XSDs*) nützlich. Mithilfe eines rekursiven Typvergabemechanismus erreichen XSDs gegenüber DTDs eine höhere Ausdrucksstärke, sind dadurch aber auch komplexer. Dieser Mechanismus erlaubt es, für dasselbe Schema-Element unterschiedliche Typen zu definieren. Dadurch sind mit XSD auch genauere und strengere Prüfungen möglich. Da XSDs selbst XML-Dokumente sind, können sie entsprechend auf Wohlgeformtheit und Gültigkeit geprüft werden.

XML-Schema-Definition

XSD ist somit neben DTD eine Möglichkeit, die Struktur von XML-Dokumenten zu beschreiben. XSD bietet auch die Möglichkeit, den Inhalt von Elementen und Attributen zu beschränken, z. B. auf Zahlen, Datumsangaben oder Texte, beispielsweise mittels regulärer Ausdrücke. Die Syntax und Semantik einer DTD ist Bestandteil der XML-Spezifikation. Diese Entscheidung wurde später kritisiert, da die DTD-Syntax an sich kein XML ist. Mit XSD existiert eine eigene Spezifikation zur Definition von Elementtypen für

XML-Dokumente. XSDs werden in einer XML-Syntax formuliert und bieten somit mehr Möglichkeiten, sind allerdings auch komplexer als DTDs.

XML-Schema enthält umfangreichere Darstellungsmöglichkeiten als DTDs und bietet

- vielfältige, vordefinierte Datentypen, welche auch in SQL, Java und anderen weit verbreiteten Programmiersprachen üblich sind,
- die Möglichkeiten zur Definition eigener Datentypen,
- explizite Vererbungsmechanismen, die leicht zu verstehen und zu handhaben sind,
- umfangreiche Darstellungsmöglichkeiten,
- Erweiterbarkeit,
- die Möglichkeit, Integritätsbedingungen darzustellen, die die Korrektheit der XML-Dokumente beschreiben,
- eine Codierung in XML-Syntax,
- Berücksichtigung von Namensräumen.

Das XML-Schema bietet also gegenüber einer DTD ein erheblich erweitertes Konzept einer schematischen Beschreibung von XML-Dokumenten. Dieses Konzept hat gegenüber einer DTD die gravierenden Vorteile, dass zum einen ein Schema-Dokument in XML formuliert wird und darüber hinaus, dass XML-Schema ein vielfältiges Typkonzept hat.

In XSDs wird zwischen einfachen und komplexen Typen unterschieden. XML-Schema ermöglicht somit die Verarbeitung von daten- und dokumentenorientierten XML-Dateien. Einfache Datentypen werden als Grundlage für Attributwerte und Elementinhalte verwendet.

| Typname | Bemerkung |
|-----------------|---|
| anyType | allgemeiner Basisdatentyp: vereinigt alle folgenden Typen |
| string | beliebiges Unicode-Symbol |
| boolean | um zweiwertige Logik abzubilden |
| decimal | Dezimalzahl z. B.: -6.78, 1234567.666666 |
| float | 32-Bit-Zahl z. B.: -1.23E4, INF |
| double | 64-Bit-Zahl z. B.: -1.23E4, INF |
| duration | Zeitraumangabe in der Form: P1Y2M3DT4H5M6S Zeitraum: 1 Jahr, 2 Monaten, 3 Tagen, 4 Stunden, 5 Minuten und 6 Sekunden |
| dateTime | Zeitpunkt in Datum und Uhrzeit z. B.: 2000-12-10T07:12:3.000 |
| time | Uhrzeit z. B.: 12:57:00 |

| | |
|---------------------|---|
| date | Datum z. B.: 2000-12-10 |
| gYearMonth | Datum aus Jahr + Monat, g steht für gregorianischer Kalender, z. B.: 2000-12 |
| gYear | Datum - Jahresangabe z. B.: 2021 |
| gMonthDay | Datum aus Tag und Monat z. B.: --25-08 |
| gDay | Datum - Tag des Monats z. B.: ----25 |
| gMonth | Datum - Monatsangabe z. B.: --08 |
| hexBinary | hexadezimale Darstellung beliebiger binär interpretierter Inhalte z. B.: 01DF |
| base64Binary | Base64-Darstellung binär-interpretierter Inhalte z. B.: SGVabcXYZ123456+*_Q |
| anyURI | eine gültige URI z. B.: http://www.fernuni-hagen.de |
| QName | qualifizierter Name des Namensraums z. B.: anyElement, xsd:nocheinElement, |
| NOTATION | DTD-Typ NOTATION: XSD Variante |

Tabelle 4.1: Einfache primitive Typen aus XML-Schema

Komplexe Datentypen bieten die Möglichkeit der Kombination von einfachen Datentypen und Definitionsmöglichkeiten für sogenannte Einschränkungen (engl. *constraints*) innerhalb der Unterelemente eines Datentyps. Dieses Typkonzept wird in Tabelle 4.1 und Tabelle 4.2 im Überblick aufgeführt

| Typname | Bemerkung | spezialisiert von Typ Aufzählung von Typ |
|-------------------------|---|---|
| normalizedString | Unicode ohne LF, CR, VT | String |
| Token | normalizedString ohne führende und abschließende LF, CR, VT | normalizedString |
| Language | Sprachcodierung z. B.: de, en-US | Token |
| NMOKEN | DTD-Typ NMOKEN: XSD-Variante | Token |
| NMOKENS | DTD-Typ NMOKENS: XSD-Variante | Aufzählung von NMOKEN s |
| Name | wohlgeformter XML-Name | Token |
| NCName | Name ohne Doppelpunkt | Name |
| ID | DTD-Typ ID: XSD-Variante. | NCName |
| IDREF | DTD-Typ IDREF: XSD-Variante | NCName |
| IDREFS | DTD-Typ IDREFS: XSD-Variante | Aufzählung von IDREF s |
| ENTITY | XSD-Darstellung des DTD-Typ ENTITY: XSD-Variante | NCName |

| ENTITIES | DTD-Typ ENTITIES: XSD-Variante | Aufzählung von ENTITY s |
|---------------------------|-----------------------------------|-------------------------|
| Integer | -n...0...+n | Decimal |
| nonPositiveInteger | -n...0 | Integer |
| negativeInteger | -n...-1 | nonPositiveInteger |
| Long | $-2^{63} \dots 2^{63}-1$ | Integer |
| Int | $-2^{31} \dots 2^{31}-1$ | Long |
| Short | $-2^{15} \dots 2^{15}-1$ | Int |
| Byte | -27... 27-1 | Short |
| nonNegativeInteger | 0...+n | Integer |
| unsignedLong | 0... $2^{63}-1$ | nonNegativeInteger |
| unsignedInt | 0... $2^{31}-1$ | unsignedLong |
| unsignedShort | 0... $2^{15}-1$ | unsignedInt |
| unsignedByte | 0... 27-1 | unsignedShort |
| positiveInteger | 1...+n | nonNegativeInteger |

Tabelle 4.2: Einfache abgeleitete Typen aus XML-Schema

Auch wenn DTDs und XSDs eine große strukturelle Ausdrucksfähigkeit besitzen, sind ihre Schwächen, dass: nicht alle erwünschten Beschränkungen dargestellt werden können, die Invarianten zwar hilfreich sind, aber auch die Handhabbarkeit der Daten erschweren können, komplexe Beschränkungen spezielle Sorgfalt beim Update erfordern.

Selbsttestaufgabe 3.7:

Erstellen sie eine XML-Schema-Datei bestellung.xsd, passend zu folgender XML-Datei:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<bestellung bestellnummer="123456"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="bestellung.xsd">
  <kunde>FernUniversität in Hagen</kunde>
  <lieferadresse>
    <name>Monika Lücke</name>
    <adresse>Universitätsstraße 1</adresse>
    <stadt>58084 Hagen</stadt>
    <land>Deutschland</land>
  </lieferadresse>
  <ware>
```

```

<beschreibung>Laserdrucker</beschreibung>
<bemerkung>Postscriptfähig</bemerkung>
<menge>1</menge>
<preis>810.90</preis>
</ware>
<ware>
  <beschreibung>Tonerkartusche Schwarz</beschreibung>
  <menge>1</menge>
  <preis>200.49</preis>
</ware>
</bestellung>

```

DTD oder Schema?

Wird eine Datenstruktur nur einmal verwendet, ist es nicht nötig ein Modell dafür zu entwerfen. Werden jedoch bestimmte Datenmengen mehrfach verwendet, lohnt sich ein Dokumentenmodell zur Kontrolle der Gültigkeit, aber auch weil XML-Editoren aus solchen Modellen Eingabemasken generieren können, die die Datenpflege wesentlich erleichtern. Die Entscheidung, ob DTD oder Schema günstiger ist, hängt davon ab, um welche Daten es geht. DTDs setzt man vorwiegend bei Anwendungen ein, die mit Textdokumenten zu tun haben. Handelt es sich um sehr spezielle Datenformate und vielfältige Datenstrukturen, sollten die Möglichkeiten von XML-Schema genutzt werden.

5. Konfliktvermeidung bei XML

Scalable Vector Graphics

Teilweise werden in einem XML-Dokument mehrere XML-Anwendungen benutzt. So könnte zum Beispiel in einem XHTML-Dokument zusätzlich eine Grafik mittels skalierbarer Vektorgrafiken (engl. *scalable vector graphics, SVGs*) spezifiziert sein. Ein Konflikt entstünde dann, wenn in beiden Anwendungen ein Element gleichen Namens definiert wäre, aber die Bedeutungen unterschiedlich wären. Dieser Konflikt ist besonders schlimm, da er nur semantisch auftritt und von einem Parser nicht erkannt werden kann.

Namensräume zur Konfliktvermeidung

Um solche Konflikte zu vermeiden, kommen Namensräume zum Einsatz. Namensräume haben in XML die Aufgabe, bei der Verwendung mehrerer XML-Anwendungen in einem Dokument zwischen Elementen, beziehungsweise Attributen mit gleichen Namen, die in verschiedenen XML-Anwendungen definiert sind, unterscheiden zu können und so die Eindeutigkeit von XML-Elementen sicherzustellen.

Darüber hinaus führt die Verwendung von Namensräumen bei Verwendung eines Präfixes zu der in der Praxis sehr nützlichen Situation, dass alle Elemente einer bestimmten

XML-Anwendung leicht von der Software, die das Dokument verarbeitet, identifiziert werden können.

Definition eines Namensraums

Mit der Angabe eines Namensraumes wird der Kontext angegeben, in dem ein bestimmter Name seine ganz spezielle Bedeutung erhält. Ein Namensraum definiert sich dadurch, dass Elemente dieses Namensraums an eine URI, in der Regel einen URL, gebunden werden. URI-Referenzen müssen immer eindeutig sein und somit ist gewährleistet, dass auch der Namensraum eindeutig ist. Das ist der einzige Grund, denn der Prozessor sieht bei dem entsprechenden URL nicht nach, ob der Namensraum dort abgelegt ist. Das W3C hinterlegt unter den URLs der von ihm selbst gepflegten Namensräume lediglich Hinweise, aber keine Liste der Namen.

Da URIs typischerweise Zeichen beinhalten, die in XML-Namen nicht erlaubt sind, wird den Elementen und Attributen ein Präfix genannter XML-Name mit Doppelpunkt vorangestellt. Dieser bindet das Präfix an die URI. Ein solcher „Doppel“-Name heißt qualifizierter Name, der Teil hinter dem Doppelpunkt heißt dann lokaler Teil. Ein Beispiel für die Verwendung eines Namensraums BUCH für das Element buch wäre

```
BUCH:buch
```

Hierbei ist BUCH das Präfix und buch der lokale Teil.

Durch die Verwendung des Präfixes kann die Zugehörigkeit eines Elements zu einem Namensraum sehr einfach überprüft werden, was einer weiterverarbeitenden Software ihre Aufgabe erleichtert. Die Bindung von Präfix-Namen an URIs ist nicht unbedingt injektiv, d. h. es kann mehrere Präfixe geben, die an die gleiche URI gebunden sind. Hier gilt, dass zwei Elemente zum gleichen Namensraum gehören, wenn ihre Präfix-Namen an die gleiche URI gebunden sind. Dabei können die Präfix-Namen selbst verschieden sein. Ein XML-Parser wird die Abbildung von jedem Präfix auf die gleiche URI vornehmen und so die beiden qualifizierten Namen als identisch ansehen. Ein Vorgehen wie das gerade beschriebene ist allerdings im Normalfall nicht zu empfehlen, da es die Verständlichkeit von Dokumenten erschwert.

Die Bindung eines Präfixes an eine URI erfolgt durch ein spezielles Attribut eines Elements. Dieses Präfix gilt dann für dieses Element und alle seine Kindelemente. Das Attribut hat die Form xmlns:PRAEFIX="URI", wobei für PRAEFIX das gewünschte Präfix und für URI die URL eingesetzt werden muss, an die das Präfix gebunden werden soll. Ein Beispiel wäre

```
<BUCH:buch xmlns:BUCH = "http://www.kurs1873.de/buch">
<BUCH:titel>
  Dies ist der Buchtitel
```

```

</BUCH:titel>
<BUCH:autor BUCH:name = 'nachname' />
<BUCH:Inhalt>
  <BUCH:Kapitel BUCH:Nummer = '1'>
    Titel des Kapitels
  </BUCH:Kapitel>
</BUCH:Inhalt>
</BUCH:buch>
```

Würde im gleichen Dokument ein Element Inhalt ohne das vorgestellte Präfix benutzt, so wäre dieses nicht als zum Namensraum zugehörig zu betrachten und somit als von BUCH:Inhalt zu unterscheiden. Bei der Prüfung, ob zwei Präfix-Namen an die gleiche URI gebunden sind, werden die beiden URIs in den xmlns-Attributen übrigens zeichenweise verglichen. Dadurch führt selbst der geringste Unterschied zur Entscheidung, dass die URIs, und damit die Namensräume, verschieden sind. Selbst abweichende Groß- und Kleinschreibung der Domännamen, die bei Verwendung im HTTP-Protokoll ignoriert wird, führt zu diesem Unterschied. Bei der Schreibweise der URIs ist also Sorgfalt anzuwenden. In nachfolgendem Beispiel werden zwei Namensräume verwendet, einer beschreibt einen Katalog, der andere ein Buch.

```

<KAT:katalog xmlns:KAT =
  "http://www.kurs1873.de/katalog">
  <KAT:titel>
    Dies ist der Titel des Katalogs
  </KAT:titel>
```

```

<BUCH:buch xmlns:BUCH =
  "http://www.kurs1873.de/buch">
  <BUCH:titel>
    Dies ist der Buchtitel
  </BUCH:titel>
  <BUCH:autor BUCH:name = 'nachname' />
  <BUCH:Inhalt>
    <BUCH:Kapitel BUCH:Nummer = '1'>
      Titel des Kapitels
    </BUCH:Kapitel>
  </BUCH:Inhalt>
</BUCH:buch>
</KAT:katalog>
```

Ist hingegen klar, dass ein Element und alle seine Kindelemente nur zu einem Namensraum gehören, so kann die Zugehörigkeit zu diesem Namensraum auf einfachere Weise definiert werden: benutzt wird das xmlns-Attribut, das Präfix weggelassen. Das sähe im Falle des Namensraums BUCH wie folgt aus:

```
<buch xmlns = "http://www.kurs1873.de/buch">
<titel>
  Dies ist der Buchtitel
</titel>
<autor name = 'nachname' />
<Inhalt>
  <Kapitel Nummer = '1'>
    Titel des Kapitels
  </Kapitel>
</Inhalt>
</buch>
```

Hierbei sind drei Dinge zu beachten:

- Bei diesem Vorgehen entfällt der Nebeneffekt, dass alle Elemente eines Namensraums durch eine Software einfach identifiziert werden können.
- Dieses Vorgehen gilt nur für Elemente, nicht aber für Attribute. Im vorliegenden Beispiel würden also die Attribute name und Nummer nicht zum Namensraum mit der URI <http://www.kurs1873.de/buch> gehören.
- Der Namensraum hat durch die Abwesenheit des Präfixes keinen symbolischen Namen mehr, sondern ist nur noch durch die URI definiert, was für Menschen die Lesbarkeit erschwert.

Bei der Verwendung mehrerer Namensräume können sowohl solche mit als auch solche ohne Präfix gemeinsam verwendet werden. Wird innerhalb eines Namensraums ohne Präfix in einem Kindelement ein weiterer Namensraum ohne Präfix definiert, so gilt dieser für dieses Kindelement und alle seine Kindelemente. Dies entspricht in etwa der Regel für die Sichtbarkeit lokal definierter Variablen bei geschachtelten Prozeduren in der Programmiersprache PASCAL. Es ist sogar möglich, in einem Element einen Namensraum ohne Präfix zu definieren, zu dem dieses Element dann gehört, und zusätzlich einen weiteren Namensraum mit Präfix. Ein Beispiel hierfür wäre der folgende öffnende Tag:

```
<buch xmlns = "http://www.kurs1873.de/buch" xmlns:SONST =
"http://www.kurs1873.de/sonstiges">
```

Selbsttestaufgabe 3.8:

Bestimmen Sie für alle Elemente und Attribute des folgenden XML-Dokuments, ob sie zu einem Namensraum gehören, und wenn ja, zu welchem.

```
<katalog xmlns = "http://www.kurs1873.de/katalog">
  <titel>
    Dies ist der Titel des Katalogs
  </titel>
  <BUCH:buch xmlns:BUCH = "http://www.kurs1873.de/buch">
    <BUCH:titel>
      Dies ist der Buchtitel
    </BUCH:titel>
    <BUCH:autor BUCH:name = 'nachname' />
    <BUCH:Inhalt>
      <BUCH:Kapitel BUCH:Nummer = '1'>
        Titel des Kapitels
      </BUCH:Kapitel>
    </BUCH:Inhalt>
  </BUCH:buch>
  <buch xmlns = "http://www.kurs1873.de/buch">
    <titel>
      Dies ist der zweite Buchtitel
    </titel>
    <autor name = 'nachname' />
    <Inhalt>
      <Kapitel Nummer = '1'>
        Titel des Kapitels
      </Kapitel>
    </Inhalt>
  </buch>
</katalog>
```

Namensräume und DTDs

Die Verwendung von Namensräumen und DTDs ist unabhängig voneinander. Damit kann es XML-Dokumente geben, die weder Namensräume noch DTDs benutzen. Es kann XML-Dokumente geben, die entweder Namensräume oder DTDs benutzen, und es kann XML-Dokumente geben, die sowohl Namensräume als auch DTDs benutzen. In diesem Fall muss bei der DTD die Definition des Namensraums berücksichtigt werden. Im einfachsten Fall wird ein Namensraum mit Präfix in einem Element deklariert. In diesem

Fall muss in der Deklaration aller Elemente und Attribute das Präfix erscheinen. In der DTD zur Beschreibung eines Buchs müsste das Element titel wie folgt deklariert werden:

```
<!ELEMENT BUCH:titel (#PCDATA)>
```

Wird ein Namensraum ohne Präfix benutzt, so erfolgt die Definition des Namensraums mittels einer Attribut-Deklaration. Bei Verwendung eines #FIXED Standardwertes muss das xmlns-Attribut bei der Verwendung des Elements im Dokument selbst nicht unbedingt angegeben werden. Wird es aber angegeben, muss es den in der DTD vorgegebenen Wert haben. Im Fall unseres Buchs sähe die Deklaration des Attributs wie folgt aus:

```
<!ATTLIST buch xmlns CDATA #FIXED  
"http://www.kurs1873.de/buch">
```

Es kann aus Gründen der Modularisierung sinnvoll sein, bei der Verwendung von Namensräumen mit Präfix, das Präfix in einer Entitäts-Deklaration zu kapseln. Gleiches erfolgt mit dem Doppelpunkt. In einer weiteren Entity-Deklaration wird dann aus diesen beiden Entitäten und dem lokalen Namen der qualifizierte Namen deklariert. Diese zweistufige Vorgehensweise ist notwendig, da bei direkter Verwendung der Entitäten in einer Element-Deklaration um die Entitäten herum Leerzeichen eingefügt würden! Soll das für diesen Namensraum verwendete Präfix verändert werden, so muss nur die entsprechende Entitäts-Deklaration in der DTD geändert werden. Lokales Überschreiben funktioniert auch. Von der Verwendung eines Namensraums kann abgesehen werden, indem die beiden Entitäts-Deklarationen für das Präfix und den Doppelpunkt modifiziert werden, sodass beide eine leere Zeichenkette erzeugen.

Selbsttestaufgabe 3.9:

Gegeben sei folgende DTD für ein Buch:

```
<!ELEMENT buch      (titel,autor,Inhalt)>  
<!ELEMENT titel  
(#PCDATA)>  
<!ELEMENT autor  
EMPTY>  
<!ELEMENT  
Inhalt (Kapitel+)>  
<!ELEMENT  
Kapitel (#PCDATA)>  
<!ATTLIST autor name CDATA #REQUIRED>  
<!ATTLIST Kapitel
```

Nummer CDATA #REQUIRED>

Ergänzen Sie die gegebene DTD so, dass der Namensraum mit URI <http://www.kurs1873.de/buch> benutzt wird, dass aber existierende Dokumente, die diese DTD benutzen, nicht geändert werden müssen.

Namensräume in XML-Schema

Im Unterschied zu den DTDs bietet XML-Schema eine direkte Unterstützung von Namensräumen. Dadurch wird es zum Beispiel möglich, die Gültigkeit von XML-Dokumenten durch Schemata zu prüfen, die verschiedene Vokabulare mischen, ohne dass es zu Namenskonflikten kommt.

Namensräume werden mithilfe des Attributes `xmlns` definiert. Dieses Attribut kann an beliebige XML-Elemente angehängt werden und setzt sich wie folgt zusammen: `xmlns:PRÄFIX="URI"`. PRÄFIX steht für ein selbst gewähltes Präfix und stellvertretend für einen Namensraum.

Es gibt auch die Möglichkeit, einen Default-Namensraum zu vergeben, also einen Namensraum, der für alle Elemente und Attribute ohne Präfix gelten soll. Allerdings hat dieses Vorgehen den Nachteil, dass Bezüge auf von XML-Schema vorgegebenen Datentypen und auf benutzerdefinierte Datentypen nicht mehr sofort unterschieden werden kann.

Praxistipp:

Werden Schemata für aktuelle XML-Dokumente modelliert, sollten die Möglichkeiten der Namensräume unbedingt genutzt werden. Das gilt vor allem, wenn das Schema ein Vokabular für einen bestimmten Gegenstandsbereich definiert, auf den sich zahlreiche Anwendungen beziehen.

6. Verweise per XML

XLink, XPath und XPointer

Verweise oder Links, die in HTML durch das A-Element realisiert werden, hatten einen guten Anteil am Erfolg des World Wide Web. Deshalb erscheint es nur konsequent, auch in XML Verweise vorzusehen. Die erste Möglichkeit hierzu ist ein sogenannter *XLink*. Im einfachsten Fall hat er die gleiche Funktionalität wie ein HTML-Link, die Syntax von XLink erlaubt aber sehr viel allgemeinere Strukturen, und zwar die Beschreibung beliebiger gerichteter Grafen zwischen Dokumenten

XML Base schließt eine Lücke, um die Verknüpfungsmöglichkeiten die HTML bietet, vollständig abzudecken. Um innerhalb von Dokumenten eine bestimmte Stelle zu identifizie-

ren, dient ein sogenannter XPointer, welcher wiederum auf einem sogenannten XPath aufbaut.

XLink

Ein XLink wird im einfachsten Fall durch Attribute in einem Element des Dokuments, von dem aus verwiesen werden soll, realisiert. XLinks können aber noch viel komplexere Strukturen aufbauen, insbesondere auch Links zwischen zwei externen Dokumenten. Für XLink existiert ein Namensraum mit der URI <http://www.w3.org/1999/xlink>.

Als verwendetes Präfix hat sich xlink eingebürgert. Ein XLink verwendet folgende Attribute: `xlink:type` Dieses Attribut spezifiziert, welche Art von Link bzw. Teil eines Links vorliegt.

Es gibt sieben mögliche Werte: simple, extended, locator, arc, title, resource, none. Davon beschreiben die ersten beiden Werte einfache und erweiterte Links. Ein einfacher/simple Link ist ein Verweis vom lokalen Dokument zu einer anderen URI und entspricht damit im Wesentlichen den aus HTML bekannten Links. Ein erweiterter/extended Link ist ein gerichteter Graf aus Knoten, die mittels URIs oder Labels referenziert werden, und gerichteten Kanten. Die letzten vier Werte kommen als Teile eines erweiterten Links zur Anwendung. Ein sogenannter locator beschreibt dabei einen externen Knoten in einem gerichteten Grafen, wobei der Knoten über eine URI und eventuell auch über ein Label referenziert wird.

Eine resource beschreibt dabei ein lokales Element, das einen Knoten in einem gerichteten Grafen bildet und das über ein Label referenziert wird. Ein arc beschreibt dabei eine gerichtete Kante in einem gerichteten Grafen. Der Wert title wird benutzt, wenn ein Kindelement eine Beschreibung einer Ressource bilden soll. Das Element none ist jedoch kein Link im Sinne von XLink.

Der Wert des `xlink:href` Attributs ist eine URI, die das Ziel eines einfachen Links oder eine Ressource in einem erweiterten Link bildet. Der Wert des `xlink:role` Attributs ist eine URI, die eine längere Beschreibung der externen Ressource enthält, auf die der Link verweist. Dieses Attribut ist optional. Der Wert des `xlink:title` Attributs ist ein kurzer Text zur Beschreibung einer Ressource, ähnlich dem Text, der erscheint, wenn in einem Browser der Mauszeiger für kurze Zeit auf einem Link verweilt. Dieses Attribut ist optional.

Das Attribut `xlink:show` beschreibt, wie die Ressource, in deren Zusammenhang es verwen-det wird, angezeigt werden soll. Es gibt fünf Werte: new, replace, embed, other, none. Bei new soll ein neues Fenster geöffnet werden, in dem die Ressource dargestellt wird. Bei replace soll die Ressource im aktuellen Fenster an-stelle der gegenwärtigen Seite dargestellt werden. Bei embed soll die Ressource im aktuellen Fenster innerhalb der gegenwärtigen Seite an der Stelle des Links dargestellt werden. Dies ist für Bilder oft empfehlenswert. Bei other soll eine andere Aktion als die bisher be-

schrie-bene ausgeführt werden. Diese muss allerdings durch eine zusätzliche, anwendungs-spezi-fische Auszeichnung näher spezifiziert werden. Der Wert none legt kein Verhalten fest. Dieses Attribut ist optional. Wird es verwendet, stellt es lediglich einen Vorschlag dar, an den der Browser nicht gebunden ist.

Das Attribut xlink:actuate beschreibt, wann einem Link gefolgt werden soll. Es gibt vier Werte: onLoad, onRequest, other, none. Bei onLoad soll der Link sofort verfolgt werden, wenn die Anwendung ihn sieht. Bei onRequest soll der Link nur auf eine Benutzeraktion hin verfolgt werden. Bei other gibt eine zusätzliche, anwendungsspezifische Auszeichnung an, wann der Link verfolgt werden soll. Der Wert none legt kein Verhalten fest. Dieses Attribut ist optional. Wird es verwendet, stellt es lediglich einen Vorschlag dar, an den der Browser nicht gebunden ist.

Der Wert des Attributes xlink:label ist ein XML-Name ohne Doppelpunkt, der einen locator oder eine resource in einem erweiterten Link kennzeichnet. In einem arc kennzeichnet das Attribut xlink:from die Quelle der gerichteten Kante. Der Wert ist ein XML-Name ohne Doppelpunkt, der als Wert eines xlink:label Attributes auftritt. In einem arc kennzeichnet das Attribut xlink:to das Ziel der gerichteten Kante. Der Wert ist ein XML-Name ohne Doppelpunkt, der als Wert eines xlink:label Attributes auftritt. Der Wert des Attributs xlink:arcrole ist eine URI, die eine längere Beschreibung der Bedeutung der gerichteten Kante enthält, bei der dieses Attribut angesiedelt ist. Das folgende XML-Dokument enthält Beispiele für einfache Links:

```
<katalog xmlns = "http://www.kurs1873.de/katalog"
  xmlns:xlink = "http://www.w3.org/1999/xlink">
  <titel xlink:type = "simple"
    xlink:href = "http://www.kurs1873.de/katalog1.xml"
    xlink:actuate = "onRequest" xlink:show = "replace"
    xlink:title = "Weitere Informationen zum Katalog">
    Dies ist der Titel des Katalogs
  </titel>
  <BUCH:buch xmlns:BUCH = http://www.kurs1873.de/buch">
    <BUCH:titel>
      Dies ist der Buchtitel
    </BUCH:titel>
    <BUCH:autor BUCH:name = 'nachname' />
    <BUCH:Inhalt xlink:type = "simple"
      xlink:href = "urn:isbn:0123456789"
      xlink:title ="Bestellung des Buchs">
      <BUCH:Kapitel BUCH:Nummer = '1'>
        Titel des Kapitels
      </BUCH:Kapitel>
```

```

</BUCH:Inhalt>
</BUCH:buch>

<buch xmlns = "http://www.kurs1873.de/buch">
<titel>
</titel>
<autor name = 'nachname' xlink:type = "simple"
xlink:href = "http://www.kurs1873.de/autor1.jpg"
xlink:actuate = "onLoad" xlink:show = "embed" />
<Inhalt xlink:type = "simple"
xlink:href = "http://www.kurs1873.de/buchtext.html"
xlink:actuate = "onRequest" xlink:show = "new"
xlink:title = "Kompletter Buchtext"
xlink:role = "http://www.kurs1873.de">
<Kapitel Nummer = '1'>
    Titel des Kapitels
</Kapitel>
</Inhalt>
</buch>
</katalog>

```

Der erste Link im Titel des Katalogs gleicht einem Link in einem HTML-Dokument. Er führt zu einer anderen URL und er wird auf Benutzeraktion im gleichen Fenster ausgeführt. Der zweite Link im Inhalt des ersten Buchs hat als Besonderheit, dass keine URL sondern eine URN-Adressierung benutzt wird, die auf die ISBN des betreffenden Buchs verweist. Die Benutzung dieses Links könnte bei Vorhandensein einer geeigneten Anwendung dazu führen, dass das Buch bestellt wird. Der dritte Link beim Autor des zweiten Buchs lädt ein Bild des Autors nach. Der vierte Link verweist auf den kompletten Buchtext, der auf Benutzeranfrage in einem separaten Fenster geöffnet wird.

Selbsttestaufgabe 3.10:

Schreiben Sie eine Parameter-Entity-Referenz zur Einbindung einfacher Links in die DTD eines XML-Dokuments, wobei der Wert other bei xlink:show und xlink:actuate verboten sein soll.

Der folgende Teil eines XML-Dokuments enthält ein Beispiel eines erweiterten Links, wobei wir annehmen, dass der Namensraum für xlink bereits in einem übergeordneten Element oder mittels einer DTD definiert ist.

erweiterte XLinks

```

BUCH:buch xmlns:BUCH = "http://www.kurs1873.de/buch"
xlink:type = "extended">
<BUCH:titel>

```

```

Dies ist der Buchtitel
</BUCH:titel>
<BUCH:autor BUCH:name = „nachname“ />
<BUCH:Inhalt xlink:type = "resource"
  xlink:label = "start">
<BUCH:Kapitel BUCH:Nummer = '1' xlink:type = "locator"
  xlink:href = http://www.kurs1873.de/buchtext-k1.html
  xlink:label = "k1">
  Titel des Kapitels 1
</BUCH:Kapitel>
<BUCH:Kapitel BUCH:Nummer = '2' xlink:type = "locator"
  xlink:href = "http://www.kurs1873.de/buchtext-2.html"
  xlink:label = "k2">
  Titel des Kapitels 2
</BUCH:Kapitel>
<BUCH:Kapitel BUCH:Nummer = '3' xlink:type = "locator"
  xlink:href = "http://www.kurs1873.de/buchtext-3.html"
  xlink:label = "k3">
  Titel des Kapitels 3
</BUCH:Kapitel>
</BUCH:Inhalt>

<!-- gerichtete Kanten -->
<FF xlink:type = "arc" xlink:from = "start" xlink:to =
  "k1" />
<FF xlink:type = "arc" xlink:from = "k1" xlink:to =
  "k2" />
<FF xlink:type = "arc" xlink:from = "k2" xlink:to =
  "k3" />
<REW xlink:type = "arc" xlink:from = "k3" xlink:to =
  "k2" />
<REW xlink:type = "arc" xlink:from = "k2" xlink:to =
  "k1" />
<REW xlink:type = "arc" xlink:from = "k1" xlink:to =
  "start" />
</BUCH:buch>
```

In diesem Fall gibt es vier Knoten im Grafen: einen internen, der den Start des Inhalts angibt, und drei externe, die Kapiteltexte darstellen. Die gerichteten Kanten verketten diese Knoten. Bei einer gerichteten Kante darf `xlink:from` fehlen, dann gehen Kanten von allen Knoten des Grafen zum Zielknoten. Gleiches gilt für `xlink:to`.

Selbsttestaufgabe 3.11:

Welche Kanten kreiert das Element <Kanten xlink:type = "arc" /> ?

XML Base

Im Jahr 2001 hat das W3C eine Empfehlung zu XML Base verabschiedet (seit Januar 2009 XML Base – Second Edition), um eine Lücke in XLink zu schließen. Eine Anforderung an XLink ist es, HTML-Verweiskonstrukte generisch zu unterstützen. XML Base definiert eine Methode ähnlich zum Base-Tag in HTML 4.01, womit es möglich ist, explizit eine Basisadresse für URIs zu definieren, um relative URIs im Dokument aufzulösen, z. B. relative URIs zu externen Grafiken, Formular verarbeitende Programme etc. XML Base stellt ein entsprechendes XML-Attribut `xml:base` zur Verfügung, das nicht nur für XLink-Anwendungen genutzt werden kann. Damit kann eine andere Basisadresse bestimmt werden als die, die ein XML-Dokument selbst verwendet. Der Wert des Attributs wird bei der Verarbeitung als URI interpretiert. Das verwendete Präfix `xml` ist an den Namensraum <http://www.w3.org/XML/1998/namespace> gebunden.

Für XLink, die innerhalb eines Elementes mit `xml:base`-Attribut liegen, werden relative URIs mithilfe des dort angegebenen URIs ergänzt.

```
<?xml version="1.0" encoding="UTF-8"?>
<doc xmlns:xml="http://www.w3.org/XML/1998/namespace"
      xml:base="http://www.fernuni-hagen.de/">
  <link xlink:href="/beispiel/doc01873.xml" />
</doc>
```

Der in `xlink:href` angegebene URL wird vervollständigt zu <http://www.fernuni-hagen.de/beispiel/doc01873.xml>, unabhängig davon, wo das Dokument physikalisch liegt.

XPath und XPointer

Seitdem immer mehr Informationen in Form von XML-Dokumenten gespeichert werden, wird auch eine Möglichkeit benötigt, in strukturierter Weise Zugriff auf diese Informationen zu erhalten. Es existieren mehrere Spezifikationen für Sprachen, die es ermöglichen, Beziehungen zwischen Dokumenten, die Auswahl von Teilbereichen eines Dokuments sowie die gezielte Suche nach Inhalten auszudrücken.

XPath selbst ist keine XML-Anwendung. Die Sprache enthält eine eigene Syntax zur Bildung von Zeichenketten, die sich zur Adressierung von Untermengen eines Dokumentes verwenden lassen. Die Empfehlung befindet sich unter <http://www.w3.org/TR/xpath>. XPath spielt vor allem in XSLT- oder XSL-Dokumenten eine wichtige Rolle, die ja wohlgeformte Dokumente sind.

XPath ist eine Sprache zur Lokalisierung von Teilen eines XML-Dokuments und arbeitet auf einer Baumansicht eines XML-Dokuments, wobei verschiedene Knoten unterschieden werden:

- Wurzelknoten,
- Elementknoten,
- Attributknoten,
- Textknoten für Text innerhalb von Elementen,
- Namensraumknoten,
- Verarbeitungsanweisungsknoten,
- Kommentarknoten.

Lokalisierungsschritt

Dabei entsteht ein XPath-Ausdruck aus einer Folge von Lokalisierungsschritten. Jeder Lokalisierungsschritt selektiert eine Teilmenge der Knoten eines Dokuments, wobei das Ergebnis eines Lokalisierungsschritts der Ausgang für den nächsten Lokalisierungsschritt ist. Die Syntax eines Lokalisierungsschritts besteht aus einer Achse in die Richtung, in der die zu selektierenden Knoten, ausgehend von der Ausgangsmenge, gesucht werden, einem Knotentest (einer Einschränkung der durch die Achsenangaben erreichbaren Knoten durch Angabe von geforderten Knotennamen oder Knotentypen) und Prädikaten. Durch Prädikate können optional weitere Filterbedingungen angegeben werden. Dabei ist XPath noch keine XML-Abfragesprache, da die Kombination von XML-Fragmenten als Ergebnismenge nicht möglich ist.

XPath ist somit eine Anfragesprache, die speziell für die Adressierung und Selektion von XML-Dokumentteilen entworfen wurde. Mithilfe sogenannter Pfadausdrücke kann eine Menge von Elementen oder auch ganz gezielt ein einzelnes Element eines Dokuments bzw. dessen Inhalt ausgewählt werden. Pfadausdrücke geben die Position der gesuchten Elemente immer bezüglich des sogenannten Kontextknotens an, der standardmäßig die Wurzel des Dokuments ist. Die Notation ist deklarativ und bezieht sich auf den XML-Dokumentenbaum, der den Inhalt des Dokuments als eine Hierarchie von Knoten repräsentiert. Die Syntax orientiert sich dabei an sogenannten Uniform Resource Identifiers (URIs), die zur Navigation in Verzeichnissen dienen.

XPath-Anfragen ermöglichen es somit, die Suche nach einem bestimmten Muster auf einem ausgewählten Kontextknoten auszuführen, um dann relativ zu der Ergebnisknotenmenge zusätzliche Operationen anzuwenden. Diese Notation gibt XPath-Ausdrücken eine außergewöhnliche Flexibilität bei der Suche in einem Dokumentenbaum. Ein Pfadausdruck besteht aus einem sogenannten Lokationsschritt (engl. *location step*) oder einer Reihe solcher Lokationsschritte, die jeweils durch das Zeichen / getrennt sind und die folgende Form aufweisen: axis::node_test[predicate1][predicate2]...

Jeder dieser Lokationsschritte liefert eine Menge von Knoten zurück, die eine neue Kontextknotenmenge bilden, auf die dann der folgende Lokationsschritt angewendet wird. Die Achse bestimmt die Beziehung der auszuwählenden Knoten zur aktuellen Kontextknotenmenge. Für die Achse existieren folgende Ausdrücke.

```
ancestor, ancestor-or-self, attribute, child, descendant,
descendant-or-self, following, following-sibling, parent,
preceding, preceding-sibling, self, attribute (abgekürzt
@), namespace
```

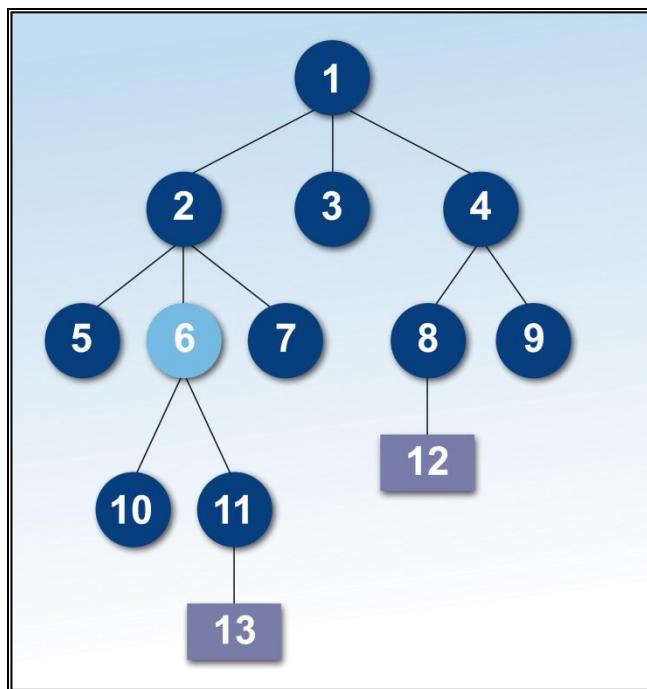


Abbildung 6.1 Die Achsendimensionen von XPath-Pfadausdrücken

- ancestor-or-self:: 6, 2, 1
- preceding-sibling:: 5
- preceding:: 5
- following-sibling:: 7
- following:: 7, 3, 4, 8, 9, 12

Der Knotentest kann zum einen verwendet werden, um die vom Lokationsschritt gelieferte Knotenmenge auf einen bestimmten Typ einzuschränken; dazu werden die Knotentests

```
comment(), node(), processing-instruction(), text()
#«FragmentID»
```

eingesetzt. Zum anderen kann die Knotenmenge auf der selektierten Achse aber auch bezüglich eines Elementnamens, der als Knotentest-Wert vorgegeben wird, gefiltert werden. Bei Verwendung des Zeichens * werden alle Elemente ungefiltert selektiert. Schließlich kann die bisher selektierte Knotenmenge in einem letzten Schritt durch eine Reihe von Prädikaten noch weiter eingeschränkt werden. Ein Prädikat ist ein boolescher Ausdruck, der für jeden Knoten in der aus der Achse und dem Knotentest resultierenden Knotenmenge berechnet wird. XPath bietet eine Reihe von Funktionen, die verwendet werden können, um die gewünschten Knoten zu selektieren. Diese Funktionen reichen von der einfachen Positionsbestimmung innerhalb einer Knotenmenge bis hin zur Stringkonkatenation. Mithilfe von Vergleichsoperatoren und mathematischen Funktionen können komplexe Ausdrücke gebildet werden. Einige der wichtigsten XPath-Funktionen sind:

```
count(), position(), last(), name(), concat(), contains(),
substring()
```

Für oft verwendete XPath-Konstrukte existieren folgende Abkürzungen: Wenn keine Achse spezifiziert wird, ist die Achse child:: der Default-Wert. Anstatt der Achse attribute:: kann das Zeichen @ verwendet werden. Das Zeichen // steht für den oft verwendeten Ausdruck /descendant-or-self::node(). Außerdem ist das Zeichen . die Abkürzung für self::node() und die Zeichen .. stehen für parent::node().

| Pfadausdruck | Abgekürzter Ausdruck |
|---------------------------------------|----------------------------|
| child)::Book[position() = 1] | Book[position() = 1] |
| Book[position() = 1]/attribute::color | Book[position() = 1]@color |
| /descendant-or-self::node()/Title | //Title |
| self::node()//Title | .//Title |
| parent::node() /Title | ../Title |

Tabelle 6.1: XPath-Pfadausdrücke und ihre Abkürzungen

Tabelle 6.1 zeigt eine Übersicht von Beispielen der zuvor aufgeführten Abkürzungsnotationen. Abschließend folgen noch einige komplexere Beispiele von Pfadausdrücken, die die erläuterten Konzepte veranschaulichen sollen:

- /bookstore[@speciality="textbooks"]: Selektiere alle Bookstore-Elemente an der Wurzel des Dokuments, die ein speciality-Attribut mit dem Inhalt „textbooks“ besitzen.

- `book [/bookstore/@speciality=@style]`: Selektiere alle Bücher, deren Stil dem speciality-Attribut des Bookstore-Elements an der Wurzel des Dokuments entspricht.
- `book[abstract] [title]` : Finde alle Bücher, die ein abstract- und ein title-Element haben.
- `author [not (last-name[1] = "Bob")]` : Finde alle Autoren unmittelbar unterhalb des Kontextknotens, deren erster Nachname nicht „Bob“ ist.
- `/book[author/degree]`: Selektiere alle Bücher unterhalb des Kontextknotens, deren Autor mindestens ein degree-Element besitzt.

Selbsttestaufgabe 3.12:

Betrachten Sie die in Selbsttestaufgabe 3.3 vorgestellte XML-Datei. Erstellen Sie XPath-Ausdrücke, die:

1. alle bestellten Waren selektieren,
2. alle bestellten Waren mit einem Preis über 500.00 selektieren,
3. den Namen aller bestellten Waren mit einem Preis über 500.00 selektieren.

7. Anhang

Material zu Selbsttestaufgabe 3.2: [selbsttest-3-2.xml](#)

Relevante W3C Standards und Empfehlungen

Wichtige Quellen:

[www.w3.org/TR/xmlbase/](#) XML Base (Second Edition) - Januar 2009

[www.w3.org/TR/xlink11/](#) XML Linking Language (XLink) Version 1.1 Mai 2010

[www.w3.org/TR/xmlschema-11-1/](#) XML-Schema Part 1: Structures (Second Edition April 2012)

[www.w3.org/TR/xmlschema-11-2/](#) XML-Schema Part 2: Datatypes (Second Edition April 2012)

[www.w3.org/TR/xpath-30/](#) XML Path Language (XPath) Version 3.0 April 2014

[www.w3.org/TR/xmlstylesheet/](#) Associating Style Sheets with XML documents 1.0 Second Edition Oktober 2010

[www.w3.org/TR/REC-xml-names/](#) Namespaces in XML 1.0 (Third Edition) Dezember 2009

[www.w3.org./TR/xquery/](#) XQuery 1.0 Second Edition Dezember 2010

[www.w3.org/TR/xquery-semantics/](#) XQuery 1.0 and XPath 2.0 Formal Semantics (Second Edition) Dezember 2010

[www.w3.org/TR/xquery-30/](#) XQuery 3.0 April 2014

[www.w3.org/TR>xpath-functions-30/](#) XQuery and XPath Functions and Operators 3.0 April 2014

[www.w3.org/TR/xqueryx-30/](#) XML-Syntax for XQuery 3.0 (XQueryX 3.0) April 2014

[www.w3.org/TR>xpath-datamodel-30/](#) XQuery and XPath Data Model (XDM) 3.0 April 2014

[www.w3.org/TR/xslt-xquery-serialization-30/](#) XSLT and XQuery Serialization 3.0 April 2014

www.w3.org/TR/html5/

HTML5 Oktober 2014

[W3C 1] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Data Model (XDM).

W3C Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/xpath-datamodel/>.

[W3C 2] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Functions and Operators W3C Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/xpath-functions/>.

[W3C 3] World Wide Web Consortium: XQuery 1.0 and XPath 2.0 Formal Semantics.

W3C Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/2005/CR-xquery-semantics-20051103/>.

[W3C 4] World Wide Web Consortium: XML Path Language XPath 1.0. W3C Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/1999/REC-xpath-19991116>.

[W3C 5] World Wide Web Consortium: XML Path Language (XPath) 2.0. W3C Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/2005/CR-xpath20-20051103/>.

[W3C 6] World Wide Web Consortium: XSL Transformations (XSLT) Version 2.0. W3C Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/2005/CR-xslt20-20051103/>.

[W3C 7] World Wide Web Consortium: XQuery 1.0: An XML Query Language. W3C Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/2005/CR-xquery-20051103/>.

[W3C 8] World Wide Web Consortium: XML Query Use Cases. W3C Working Draft 15 September 2005.

Internet: <http://www.w3.org/TR/2005/WD-xquery-use-cases-20050915/>

[W3C 9] World Wide Web Consortium: XML Query (XQuery) Requirements. W3C Working Draft 3 June 2005.

Internet: <http://www.w3.org/TR/2005/WD-xquery-requirements-20050603>.

[W3C 10] World Wide Web Consortium: XSLT 2.0 and XQuery 1.0 Serialization. Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/2005/CR-xslt-xquery-serialization-20051103/>

[W3C 11] World Wide Web Consortium: XQuery and XPath Full-Text Requirements. W3C Candidate Recommendation 3 November 2005.

Internet: <http://www.w3.org/TR/2003/WD-xquery-full-text-requirements-20030502/>

[W3C 12] World Wide Web Consortium: XML Schema Part 2: Datatypes Second Edition.

W3C Recommendation 28 October 2004.

Internet: <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>

8. Literatur

- [HM01] E. R. Harold, W. S. Means. XML in a Nutshell, Dt. Ausgabe. O'Reilly Verlag, Köln, 2001
- [K05] Gjergji Kasneci. Implikationsprobleme für XML-Schema-Sprachen. Diplomarbeit - Philipps-Universität Marburg. 2005
- [S05] Thomas Schwentick. „Vor lauter Bäumen...“ XML und formale Sprachen, Marburg 2005
- [S08] Schöning, Uwe. Theoretische Informatik - kurz gefasst. Spektrum 2008, 5. Auflage. ISBN-13: 978-3827418241
- [V15] Vonhoegen, Helmut. Einstieg in XML Grundlagen, Praxis, Referenz. Rheinwerk 2015, 8. Auflage. ISBN-13: ISBN 978-3-8362-3798-7

9. Abbildungsverzeichnis

| | |
|---|----|
| Abbildung 2.1 Überblick über die wichtigsten Mitglieder der Sprachfamilie XML [V15].... | 6 |
| Abbildung 4.1 Beispiel-Dokument zur Beispiel-DTD | 25 |
| Abbildung 4.2 DTD Beispiel aus [S05]..... | 27 |
| Abbildung 4.3 Entwurf aus [S05] | 28 |
| Abbildung 6.1 Die Achsendimensionen von XPath-Pfadausdrücken | 45 |

10. Tabellenverzeichnis

| | |
|---|----|
| Tabelle 3.1 In der XML-Spezifikation genannte Codierungen | 13 |
| Tabelle 4.1: Einfache primitive Typen aus XML-Schema..... | 30 |
| Tabelle 4.2: Einfache abgeleitete Typen aus XML-Schema | 31 |
| Tabelle 6.1: XPath-Pfadausdrücke und ihre Abkürzungen | 46 |

11. Lösung der Selbsttestaufgaben

Selbsttestaufgabe 3.1

Erläutern Sie den Unterschied zwischen einem wohlgeformten und einem gültigen XML-Dokument.

XML beinhaltet eine Grammatik, durch welche die Syntax eines XML-Dokumentes strikt definiert wird. Ähnlich wie bei einem Software-Programm einem Compiler kann ein XML-Dokument zunächst einem Parser zugeführt werden, der die Einhaltung der Grammatik prüft. XML-Dokumente, die der Grammatik genügen, heißen wohlgeformt. XML-Dokumente, die der Grammatik nicht genügen, können nicht verarbeitet werden.

Ein wohlgeformtes XML-Dokument, das mit einem Dokumentenmodell verknüpft ist und dem Schema genügt, heißt gültig. Ansonsten heißt es ungültig. Dabei ist entscheidend, dass die Wohlgeformtheit und Gültigkeit maschinell geprüft werden können.

Selbsttestaufgabe 3.2

Welche der folgenden Elemente genügen nicht den Syntax-Regeln?

Würde das folgende XML-Dokument nach Korrektur der fehlerhaften Elemente wohlgeformt sein?

```
<br/>
<buch>
< titel>
    Dies ist der Buchtitel
</titel>
<autor name = 'Nachname' name = "Vorname" / >
<Inhalt>
<Kapitel Nummer = '1'>
<Unterkapitel Nummer = „1.1“>
    Der Name des Unterkapitels 1.1
</Unterkapitel>
<Unterkapitel Nummer = „1.2“>
    Der Name des Unterkapitels 1.2
<Unterkapitel Nummer = „1.3“>
    Der Name des Unterkapitels 1.3
</Unterkapitel>
</Unterkapitel>
</Kapitel>
```

```
<Kapitel Nummer = „2“>
    Der Name des zweiten Kapitels
</Kapitel>
</Inhalt>
</buch>
```

Das Element `<titel>` ist fehlerhaft, da beim öffnenden Tag ein Leerzeichen zwischen der öffnenden spitzen Klammer und dem Elementnamen vorkommt.

Das Element `<autor>` ist fehlerhaft, da zwischen dem Schrägstrich und der schließenden spitzen Klammer ein Leerzeichen vorkommt und da das Attribut Name zweimal vorkommt.

Das dritte Unterkapitel wird geöffnet, bevor das zweite geschlossen wird. Diese Überlappung ist nicht zulässig.

Da es kein eindeutiges Wurzelement gibt, bilden die Elemente auch nach der Korrektur der Fehler keinen hierarchischen Strukturabaum.

Damit wäre das XML-Dokument auch nach der Korrektur der Fehler in den Elementen nicht wohlgeformt.

Selbsttestaufgabe 3.3

Studieren Sie das XML-Dokument in der Datei `selbsttest-3-2.xml`. Ist es wohlgeformt? Besorgen Sie sich einen XML-Parser. Prüfen Sie mit dem XML-Parser, ob `selbsttest-3-2.xml` wohlgeformt ist. Falls nicht, korrigieren Sie die Fehler einen nach dem anderen und parsen Sie solange erneut, bis das XML-Dokument wohlgeformt ist.

Das XML-Dokument `selbsttest-3-2.xml` ist nicht wohlgeformt. Der Parser meldet zunächst einen Fehler in der ersten Zeile. Hier ist ein Leerzeichen zu viel. Danach findet er noch einen Fehler beim Tag `
`. Dieses muss in XML `
` heißen.

Selbsttestaufgabe 3.4

Definieren Sie ein DTD-Element `telefonnummer`, das aus den Elementen `landesvorwahl`, `vorwahl` und `nummer` in genau dieser Reihenfolge besteht.

```
<!ELEMENT telefonnummer (landesvorwahl, vorwahl, nummer)>
```

Selbsttestaufgabe 3.5

Schreiben Sie eine einfache externe DTD in einer Datei rezept.dtd.

Deklarieren Sie dazu ein Element rezept, das als Kindelement ein Element hauptzutat und eine oder mehrere weitere Elemente zutat hat. Alternativ zu zutat kann auch einmal dressing enthalten sein.

Die Reihenfolge der Elemente ist beliebig.

Deklarieren Sie die Elemente hauptzutat, zutat und dressing jeweils mit dem Inhaltsmodell #PCDATA. Die DTD lautet:

```
<!ELEMENT rezept ((hauptzutat), ((zutat)+ | (dressing))) | (((dressing) | (zutat)+), (hauptzutat))>

<!ELEMENT hauptzutat (#PCDATA)>
<!ELEMENT zutat (#PCDATA)>
<!ELEMENT dressing (#PCDATA)>
```

Selbsttestaufgabe 3.6

Deklarieren Sie die Attribute eines bereits deklarierten Elements Name. Es soll ein optionales Attribut Alter geben, ein Attribut Form mit festem Wert Nix und ein verpflichtendes Attribut Kundentyp mit den möglichen Werten Gut, Schlecht, und Unbekannt.

Die Attribut-Deklaration lautet

```
<!ATTLIST Name Alter NMOKEN #IMPLIED
Form NMOKEN #FIXED "Nix"
Kundentyp (Gut | Schlecht | Unbekannt) #REQUIRED>
```

Selbsttestaufgabe 3.7

Erstellen sie eine XML-Schema-Datei bestellung.xsd, passend zu folgender XML-Datei:

Eine passende Schema-Datei könnte wie folgt aussehen:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
<xs:element name="bestellung">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="kunde" type="xs:string"/>
      <xs:element name="lieferadresse">
    <xs:complexType>
```

```
<xss:sequence>
<xss:element name="name" type="xs:string"/>
<xss:element name="adresse" type="xs:string"/>
<xss:element name="stadt" type="xs:string"/>
<xss:element name="land" type="xs:string"
</xss:sequence>
</xss:complexType>
</xss:element>
<xss:element name="ware" maxOccurs="unbounded">
<xss:complexType>
<xss:sequence>
<xss:element name="beschreibung" type="xs:string"/>
<xss:element name="bemerkung" type="xs:string" minOccurs="0"/>
<xss:element name="menge" type="xs:positiveInteger"/>
<xss:element name="preis" type="xs:decimal"/>
</xss:sequence>
</xss:complexType>
</xss:element> </xss:sequence>
<xss:attribute name="bestellnummer" type="xs:string" use="required"/>
</xss:complexType>
</xss:element>
</xss:schema>
```

Selbsttestaufgabe 3.8

Bestimmen Sie für alle Elemente und Attribute des folgenden XML-Dokuments, ob sie zu einem Namensraum gehören, und wenn ja, zu welchem.

Die Elemente gehören alle zu einem Namensraum. Die Attribute von Elementen des ersten Buchs gehören zum Namensraum mit der URI <http://www.kurs1873.de/buch>, die Attribute von Elementen des zweiten Buchs gehören hingegen zu keinem Namensraum.

Das Element `katalog` und das zugehörige Kindelement `titel` gehören zum Namensraum mit der URI <http://www.kurs1873.de/katalog>. Alle anderen Elemente gehören zum Namensraum mit der URI <http://www.kurs1873.de/buch>. Dazu gehören weiterhin die Attribute von Elementen des ersten Buchs.

Selbsttestaufgabe 3.9

Gegeben sei folgende DTD für ein Buch: à Kurstext

Ergänzen Sie die gegebene DTD so, dass der Namensraum mit URI <http://www.kurs1873.de/buch> benutzt wird, dass aber existierende Dokumente, die diese DTD benutzen, nicht geändert werden müssen.

Da keine existierenden XML-Dokumente, die auf die DTD Bezug nehmen, geändert werden sollen, kommt nur ein Namensraum ohne Präfix in Frage. Aus dem gleichen Grund muss die Definition des Namensraums in einer Attribut- Deklaration zum Element `buch` unter Verwendung von #FIXED erfolgen. Die Definition erfolgt also durch die bereits im Text beschriebene Deklaration:

```
<!ATTLIST buch xmlns CDATA #FIXED "http://www.kurs1873.de/buch">
```

Selbsttestaufgabe 3.10

Schreiben Sie eine Parameter-Entity-Referenz zur Einbindung einfacher Links in die DTD eines XML-Dokuments, wobei der Wert other bei `xlink:show` und `xlink:actuate` verboten sein soll.

Die Parameter-Entity-Referenz mit dem Namen `einflink` könnte lauten:

```
<!ENTITY % einflink
  "xlink:type (simple) #FIXED 'simple' xlink:href CDATA #REQUIRED
   xmlns:xlink CDATA #FIXED 'http://www.w3.org/1999/xlink' xlink:role CDATA
   #IMPLIED xlink:title CDATA #IMPLIED
   xlink:show (replace | new | embed | none) 'none' xlink:actuate (onRequest
   | onLoad | none) 'none'"
>
```

Selbsttestaufgabe 3.11

Welche Kanten kreiert das Element `<Kanten xlink:type ="arc" />` ?

Damit werden gerichtete Kanten zwischen allen Knoten des gerichteten Grafen erzeugt, d. h. bei n Knoten werden n^2 gerichtete Kanten erzeugt.

Selbsttestaufgabe 3.12

Betrachten Sie die in Selbsttestaufgabe 3.7 vorgestellte XML-Datei. Erstellen Sie XPath-Ausdrücke, die:

1. alle bestellten Waren selektieren,
2. alle bestellten Waren mit einem Preis über 500.00 selektieren,
3. den Namen aller bestellten Waren mit einem Preis über 500.00 selektieren.

```
/bestellung/ware
/bestellung/ware[preis>500.00]
/bestellung/ware[preis>500.00]/beschreibung
```

000 000 000 (00/23)

00000-0-00-S 1

Prof. Dr. Matthias Hemmje

01877

Dokumenten- und Wissensmanagement im Internet

Kurseinheit 2:

Einführung in HTML, CSS und JavaScript

Fakultät für
**Mathematik und
Informatik**

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere das Recht der Vervielfältigung und Verbreitung sowie der Übersetzung und des Nachdrucks, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder ein anderes Verfahren) ohne schriftliche Genehmigung der FernUniversität reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Wir weisen darauf hin, dass die vorgenannten Verwertungsalternativen je nach Ausgestaltung der Nutzungsbedingungen bereits durch Einstellen in Cloud-Systeme verwirklicht sein können.

Der Inhalt dieses Studienbriefs wird gedruckt auf Recyclingpapier (80 g/m², weiß), hergestellt aus 100 % Altpapier.

Inhaltsverzeichnis

| | |
|---|-----|
| Inhaltsverzeichnis | III |
| 1 Einleitung..... | 4 |
| 2 Die Historie der Markup-Sprachen..... | 4 |
| 3 Die Hypertext-Markup-Language..... | 7 |
| 4 Cascading Stylesheets (CSS)..... | 27 |
| 5 JavaScript | 53 |
| 6 Zusammenfassung..... | 61 |
| 7 Anhang: Materialien zu HTML und CSS | 62 |
| 8 Literaturverzeichnis | 75 |
| 9 Abbildungsverzeichnis | 76 |
| 10 Tabellenverzeichnis | 77 |
| 11 Lösung der Selbsttestaufgaben | 78 |

1 Einleitung

HTML als Dokumentenbeschreibungssprache für Hypertext-Dokumente im World Wide Web wird in seiner Entstehung beleuchtet und seine grundlegenden syntaktischen Prinzipien werden skizziert und mit Anleitungen für die praktische Anwendung ergänzt.

Im Folgenden werden HTML und Cascading Stylesheets (CSS) in Beziehung gesetzt, auch in Hinblick auf den aktuellen HTML5-Standard. Die Anbindung von Stylesheets an HTML-Dokumente, die CSS-Syntax und die Verwendung von CSS-Selektoren werden ebenfalls dargestellt und können anhand von Praxisbeispielen übertragen und vertieft werden.

Im Anschluss wird im Ansatz auf JavaScript als Webprogrammiersprache eingegangen und die grundlegende Funktionsweise beleuchtet.

2 Die Historie der Markup-Sprachen

HTML

Die Einführung in die verschiedenen Dokumentbeschreibungssprachen (auch Auszeichnungssprachen, engl. *markup languages* genannt) erfolgt im Kursverlauf nicht etwa nach dem Zufallsprinzip, sondern spiegelt deren Entwicklungsgeschichte wider. Die Geschichte des World Wide Web (WWW, kurz Web) [F98] begann mit der Entwicklung von HTTP, URLs und HTML durch Tim Berners-Lee (<https://www.w3.org/People/Berners-Lee>) und seine Kolleginnen und Kollegen am CERN in Zürich im März 1989. Den Durchbruch für das Web brachte der erste grafisch orientierte Browser Mosaic im Februar 1993. Die erste HTML-Version wurde nie von einer Standardisierungsorganisation abgesegnet. Erst HTML 2.0 erhielt im November 1995 das Placet eines solchen Gremiums, nämlich der Internet Engineering Task Force. Der Sprachschatz von HTML 2.0 war eher spartanisch und ließ sich auf wenigen Seiten beschreiben. Er war auch stark logisch ausgerichtet. Die nächste HTML-Version 3.2 wurde im Mai 1996 vom inzwischen neu gegründeten World-Wide Web Consortium (W3C) verabschiedet und enthält zahlreiche Erweiterungen formatorientierter Natur. In der Version 4.0, die im Juli 1997 vom W3C verabschiedet wurde, sind die formatorientierten Sprachkonstrukte aus der Variante Strict verbannt und in die Variante Transitional relegiert. Dort sind sie aber auch als deprecated eingestuft. Das war praktikabel geworden durch das Aufkommen der sogenannten Cascading Style Sheets (CSS) als Sprache, in der Formatvorgaben für HTML und andere Dokumentensprachen gemacht werden können. Die Version 4.01 von HTML ist vom Dezember 1999. Mit HTML4 kam die Internationalisierung, denn es erfolgte ein Wechsel vom Zeichensatz ISO 8859-1 hin zu Unicode. Außerdem wurde der Sprachumfang von HTML4, jedenfalls in der strikten Variante, auf die wesentliche Funktionalität beschränkt, nämlich Inhalt und logische Struktur von Webdokumenten ausdrücken zu können. Im Gegenzug wurden Verankerungspunkte geschaffen, damit Sprachen für darüberhinausgehende Funktionalitäten mit

HTML kombiniert werden können. Zu solchen ergänzenden Sprachen gehören Style-sheet-Sprachen, Skriptsprachen oder Sprachen für andere Medien als Text (Bilder, Videos, Animationen).

Die Beschreibungssprache HTML ist zwar sehr verbreitet und damit auch bekannt – sie ist allerdings nicht die erste Markup-Sprache. Bis einschließlich der HTML-Version 4.01 basiert die Sprache auf der standardisierten generalisierten Auszeichnungssprache (engl. *standard generalized markup language, SGML*) und war lediglich eine Teilmenge aus SGML. Diese Teilmengenbeziehung ist in Abbildung 2.1 grafisch dargestellt. SGML ist – wie auch HTML – ein ISO-Standard und kann als Grundlage aller Beschreibungssprachen betrachtet werden. SGML wurde 1986 als Standard unter ISO 8879 offiziell registriert.

| SGML |

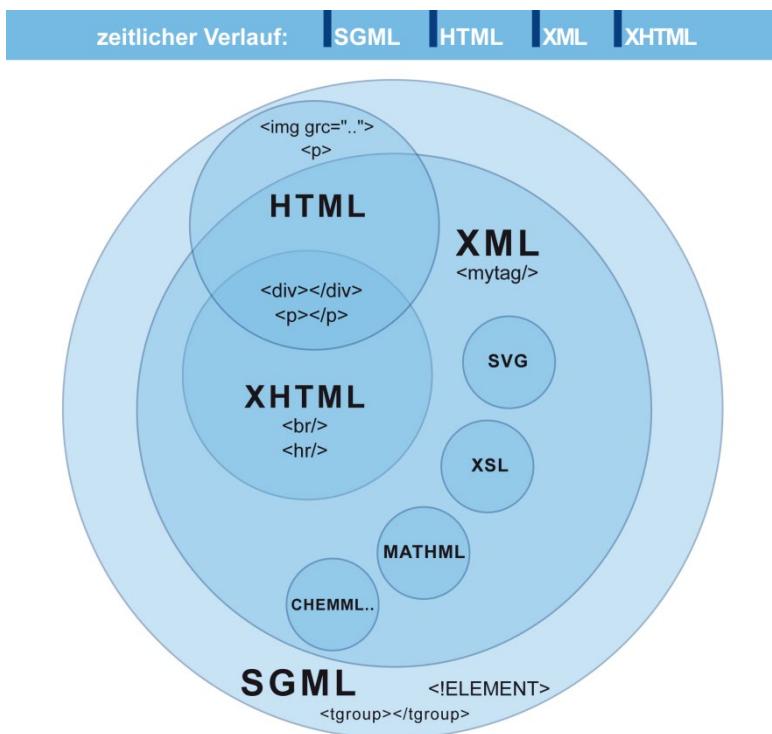


Abbildung 2.1 SGML als Basis

HTML 4 und CSS 2 waren sehr lang die normativen Referenzen für Qualität im Webdesign. Im Jahr 2004 jedoch gründeten die Entwickler von Opera, Apple und Mozilla die WHATWG (Web Hypertext Application Technology Working Group) um den Funktionsumfang von HTML zu erweitern und unter dem Namen HTML5 – jetzt Living Standard – zusammenzufassen. Das W3C entschied sich, die Arbeit der WHATWG als Grundlage für den kommenden W3C-HTML-Standard zu verwenden. Im Oktober 2014 hat das W3C HTML 5.0 den Recommendation-Status verliehen, womit es ein offizieller, fertiger Webstandard wurde. HTML5 basiert nicht mehr auf SGML. Wenn man ins Internet schaut, dann ist der Begriff HTML5 allgegenwärtig und die Aussage, HTML5 ist eine einfache Auszeichnungssprache, ist eher ungenügend. HTML5 ersetzt ja auch gleiche mehrere Standards – HTML 4.01, XHTML 1.0 und DOM HTML Level2 (siehe Abbildung 2.2).

In der Praxis wird vielerorts HTML5 sogar als Sammelbegriff für verschiedene Webtechnologien wie HTML, CSS, JavaScript und die vielen neuen Programmierschnittstellen (engl. *application programming interfaces, APIs*) für Webanwendungen etc. verwendet.

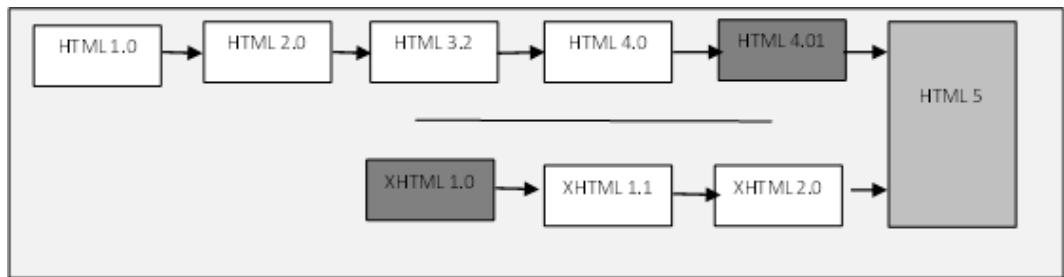


Abbildung 2.2 Werdegang von HTML zum heutigen HTML5 [F98]

Übergeordnetes Ziel bei der Entwicklung von HTML war immer, die Dokumente unabhängig von Plattformen und Browsern zu halten. Ein und dasselbe HTML-Dokument sollte über das weite Spektrum von möglichen Ausgabegeräten in puncto Auflösung, Farbtiefe, Bildschirmgröße, Bandbreite, Input- und Output Medium hinweg brauchbar sein und geräteunabhängig Darstellungen liefern, so dass HTML-Dateien auf PCs, Fernsehern oder Mobiltelefonen lesbar sein sollten. Dieses Ziel ist in vielen Teilen schon Realität, auch wenn z. B. im mobilen Sektor noch diverse technische Puzzleteile fehlen. HTML5 ist nicht irgendein weiterer Standard, sondern vielmehr der Standard, nicht nur für Webseiten, sondern für alle Arten moderner webbasierter Benutzungsschnittstellen, der ständig weiterentwickelt wird.

HTML selbst ist eine reine textbasierte Auszeichnungssprache. Der entscheidende Aspekt an HTML ist die Möglichkeit, Texte nicht nur zu formatieren, sondern Inhalte miteinander zu verknüpfen. Dies ermöglicht die Gestaltung von Texten, die nicht unbedingt linear angelegt sein müssen, sondern deren Abfolge durch die Leser bestimmt wird. Durch HTML ist nicht nur die Verknüpfung verschiedener Texte einfach gemacht worden, sondern die Quellen der Texte können auch weltweit verteilt im Web liegen. Dieser globale Aspekt der Verlinkung ist die eigentliche Innovation, die mit HTML einhergeht. Die nicht strenge Handhabung bei HTML – wie etwa die Zulässigkeit fehlender Abschluss-Tags – wurde später durch die Spezifizierung von XHTML beseitigt. Wie das Akronym bereits vermuten lässt, verhält sich XHTML konform zu XML. In Kurseinheit 3 wird auf XHTML ausführlicher eingegangen.

| Metadaten

Die Masse an Informationen, die über das Web zu erreichen sind, ist mittlerweile nahezu unüberschaubar geworden. Das Auffinden benötigter Dokumente über Suchanfragen gehört in der heutigen Zeit zum Standard bei der Nutzung des Webs. Bei der Menge an Text, Bildern, Sounddateien, Videos u. v. m. wäre es zu zeitintensiv, jedes im Web bereitgestellte Dokument Wort für Wort zu durchforsten. Inhaltsangaben, die schnell durchsucht oder katalogisiert werden können, sind eine der Voraussetzungen für eine angemessene Informationsspeicherung und Präsentation im Web. Diese Inhaltsangaben werden als Meta-Information und ihre maschinenlesbare Repräsentation als Metadaten, also als Daten, die Information

über Daten beinhalten, bezeichnet. Diese maschinenlesbaren Metadaten dienen auch zum Austausch von Informationen zwischen unterschiedlichen Programmen. Einer der Standards, die als Basis dazu dienen, ist das sogenannte Rahmenwerk für die Beschreibung von Ressourcen (engl. *resource description framework, RDF*) [W3Crdf14], das in der letzten Kurseinheit genauer beleuchtet werden wird.

Nach diesem kleinen Überblick zur Entwicklung der Markup-Sprachen folgt nun eine Einführung in die verschiedenen Dokumentbeschreibungssprachen im Detail.

Selbsttestaufgabe 2.1:

Auf welcher Auszeichnungssprache basierte HTML ursprünglich?

Selbsttestaufgabe 2.2:

Welches Gremium hat es seit 1996 zur Aufgabe, über die Entwicklungsstandards von HTML zu entscheiden?

Selbsttestaufgabe 2.3:

Warum ist die Verwendung von Metadaten in Dokumenten unverzichtbar geworden?

3 Die Hypertext-Markup-Language

Der HTML-Standard definierte die erste Form des frühen Webs als ein verteiltes Netzwerk von Informationsressourcen. Die Entwicklung des Webs beruht auf drei grundlegenden Konzepten:

- weltweite Adressierung von Ressourcen über URLs,
- einfaches Protokoll zum Anfordern und Ausliefern von Ressourcen (HTTP),
- Hypertextparadigma zur Navigation zwischen den Ressourcen.

Basis des WWW

Der Begriff der Informationsressource kann dabei sehr weit ausgelegt werden. Im Zusammenhang mit Dokumentbeschreibungssprachen ist eine Informationsressource wie das HTML-basierte frühe Web im Kern als verteiltes Repozitorium von vernetzten Textressourcen, die in HTML codiert sind, zu verstehen. Ein HTML-Text ist somit die Gesamtheit von thematisch zusammengehörigen Texteinheiten, den sogenannten Hypertextknoten. Die Beziehungen zwischen den Knoten werden durch Hypertextlinks ausgedrückt. Die Knoten eines HTML-Dokumentes bilden die elementaren Informationseinheiten, die Leserinnen und Leser auf einen Blick wahrnehmen können. Von diesen Knoten aus kann zu weiteren Informationseinheiten navigiert werden, indem den Links zwischen den Dokumenten gefolgt wird. Die Organisation eines Informationsbestandes in HTML-Hypertextknoten und HTML-Links als HTML-Hypertext eröffnet damit einen interessengesteuerten Zugang zur

Informationsressource

Information. Im Gegensatz zum HTML-Hypertextparadigma stehen z. B. Organisationsformen in Form eines monolithischen Dokumentes wie z. B. als Buch, welches im Wesentlichen fortlaufend gelesen wird, und z. B. in Form von sogenannten Datenbanken, deren Datenbestand nicht beliebig von jedem Nutzer erweitert werden kann, die also auf den zu verwaltenden Datenbestand beschränkt sind.

Durch die große Verbreitung von HTML-Dokumenten ist die Bedeutung des Dokumentenbegriffes erweitert worden. Die Knoten eines HTML-Hypertextes liegen, im Gegensatz etwa zu den Kapiteln eines Buches, nicht unbedingt in einer linearen Ordnung vor. Es ist zudem auch möglich, große Dokumente für die Onlinepräsentation entlang der hierarchischen Struktur von Kapiteln und Abschnitten in Hypertextknoten aufzubrechen und den linearen Zusammenhang des Ursprungsdokuments durch Hypertextlinks zu repräsentieren. Beispielsweise konvertiert das Werkzeug `latex2html` von Nikos Drakos ein LaTeX-Dokument in eine über HTML-Hypertextlinks organisierte Sammlung von HTML-Dokumenten.

HTML

HTML ist somit eine Dokumentenbeschreibungssprache für Hypertext-Dokumente im World Wide Web. Die sogenannte erweiterbare Auszeichnungssprache (engl. *extensible markup language, XML*) ist für diesen Zweck ebenfalls einsetzbar, wird aber zwischenzeitlich auch noch viel allgemeiner zur Repräsentation von großen Informationskollektionen auf der Grundlage unterschiedlicher Informationsparadigmen im Web eingesetzt. XML wird verwendet, um sowohl Hypertexte als auch völlig andere Formen von Informationsobjekten und Zugriffsmethoden zu repräsentieren. XML ist somit ein generisches und damit besonders leistungsfähiges Werkzeug.

XML kommt insbesondere beim verteilten Daten-, Informations- und Wissensmanagement im Web zum Einsatz. HTML hingegen wird in der Regel „nur“ zur eigentlichen Repräsentation von Hypertexten und Hypertext-Dokumentkollektionen benötigt.

Die Besonderheit des HTML-basierten frühen Web zum Zeitpunkt seiner Entstehung gegenüber anderen klassischen Hypertextsystemen bestand darin, dass der Informationsbestand zum ersten Mal in der Geschichte der elektronischen Informationsverarbeitung weltweit verteilbar wurde und somit einen offenen, universellen Hypertext realisierte.

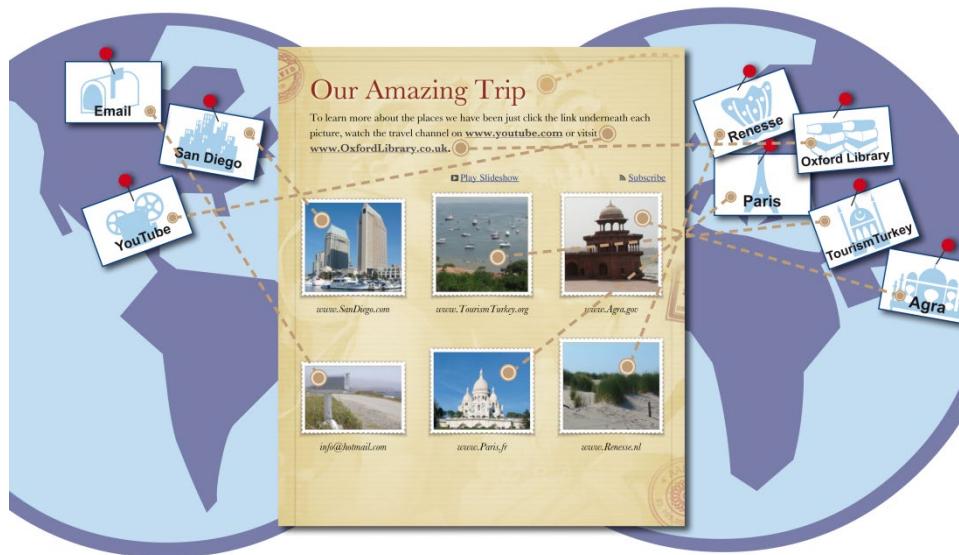


Abbildung 3.1 Weltweite Informationsverteilung durch das Web

Die Organisationsform im Web ist durch die frei definierbaren Hypertextlinks gegeben. Im HTML-basierten Web kann ein HTML-Dokument als eine Gruppe von thematisch zusammengehörigen Knoten kenntlich gemacht werden. Dies wird lediglich über rhetorische oder gestalterische Mittel erreicht. Diese sind jedoch nicht technisch, wie das bei späteren auf XML aufbauenden Markup-Sprachen der Fall ist.

Das Zugangssystem für Nutzer des frühen World Wide Web ist seit dem frühen HTML-basierten bis heute ein sogenannter Browser. Ein Web-Browser ist eine Softwarekomponente, die HTML-Dokumente aus dem Web laden und darstellen kann und welche die Möglichkeit zur Navigation entlang der eingebetteten HTML-Links bietet. Dazu muss der Browser die in das aktuelle Dokument eingebetteten HTML-Linkanker kenntlich machen und eine Möglichkeit zum Auswählen und Aktivieren eines HTML-Hypertextlinks anbieten. Es sind verschiedene HTML-Link-Semantiken denkbar, von denen die Webbrower in der Regel jedoch nur zwei unterstützen: der Zielknoten ersetzt den aktuellen Knoten im Browserfenster oder für den Zielknoten wird ein neues Browserfenster geöffnet. Webbrower bieten seitdem bis heute typischerweise die folgenden Navigationsunterstützungen an:

- Backtracking: Die Möglichkeit, während einer Browsersitzung die bisher besuchten Hypertextknoten in umgekehrter Reihenfolge wieder zurückzuverfolgen.
- History: Die Möglichkeit, auf die während einer Browsersitzung bisher besuchten Hypertextknoten direkt und wahlfrei zuzugreifen, meistens über eine Listenschnittstelle.
- Bookmarks: Die Möglichkeit, auf Hypertextknoten während des Browsens explizit Lesezeichen zu setzen, die bei Sitzungsende gespeichert und in späteren Sitzungen wieder aufgerufen werden können.

URI
http
TP/IP

Die Navigationsunterstützung, die Webbrowser bieten, ist nur ein kleiner Ausschnitt aus dem Spektrum, das in der Literatur und in anderen Hypertextsystemen zu finden ist. Um auf die drei grundlegenden Konzepte des frühen, rein HTML-basierten World Wide Web zurückzukommen, so ist HTML die originäre Sprache, in der Hypertexte im Web erstmals repräsentiert werden. Insbesondere enthält HTML Sprachmittel zur Beschreibung von in Hypertexten eingebetteten Hypertextlinks. Per HTML ist es möglich einen Textbereich als Hypertextanker zu markieren. Dieser wird mit der Adresse eines Zieldokuments und des Ankers – dem sogenannten uniformen Quellenidentifizierer (engl. *uniform resource identifier, URI*) – assoziiert. Die Kommunikation zwischen dem Browser, der im Nutzerauftrag ein Dokument bei einem Webserver anfordert und dem angesprochenen Webserver, der das Dokument ausliefert, läuft über das Hypertext-Übertragungsprotokoll (engl. *hypertext transfer protocol, http*), das auf den Internetprotokollen TCP/IP aufsetzt.

Das frappierend Neue am frühen Web war die Austauschbarkeit von Dokumenten rund um den Globus auf Knopfdruck. In den frühen 80er Jahren des letzten Jahrhunderts war der Austausch von Dokumenten zwischen verschiedenen Rechnerplattformen noch ein kleines Abenteuer: Der Austausch eines einfachen Textdokuments z. B. zwischen einem IBM-Mainframe und einer UNIX-Workstation über das Internet erforderte z. B. eine explizite Konvertierung zwischen den Zeichencodierungen EBCDIC und ASCII.

Der Austausch eines einfachen Textdokuments zwischen beispielsweise einem IBM-Personal-Computer und einem Apple-Macintosh erforderte damals z. B. den Transport speziell formatierter Disketten über das sogenannte Sneakernet. Und selbst von einer Siemens-Anlage BS2000 zur anderen gelangten die Daten nur über Magnetbänder. Diese Hardware-Probleme mit der Datenübertragung zwischen verschiedenen Plattformen waren jedoch bald gelöst: über FTP, HTTP, MIME, einheitliche Diskettenformate und die standardisierte CD-R war die Datei-Übertragung nun schneller möglich. Kritisch bleiben bis heute die Probleme, die sich aus der Vielfalt der Daten-, Dokumenten-, Informations- und Wissensrepräsentationsformate ergeben. Dokumentenformate sind typischerweise systemspezifisch: Ein Microsoft-Word-Dokument lässt sich z. B. weder mit LaTeX noch mit Adobe-FrameMaker, und oft genug noch nicht einmal mit einer früheren Word-Version, bearbeiten. Dokumentenformate sind darüber hinaus, trotz der überschaubaren Zahl gängiger Textverarbeitungssysteme, wegen der Vielzahl von Versionen und Plattformen erstaunlich zahlreich: Beim Springer-Verlag in Heidelberg beispielsweise gehen digitale Manuskripte in über 300 verschiedenen Formaten ein. Mitte der neunziger Jahre ging noch das Wort um, die einzigen universell brauchbaren TextCodierungen seien ASCII und PostScript, zwei Formate, die die wenig attraktive Alternative bieten zwischen editierbarem, aber unformatiertem und formatierbarem, aber nicht editierbarem Text. Gemeinsamer Vorteil beider Formate ist lediglich die Austauschbarkeit durch standardisierte und Plattform-übergreifend verfügbare Werkzeuge. Vor diesem Hintergrund erscheint die erfolgreiche Einführung der Hypertext Markup Lan-

guage in den frühen 90er Jahren des letzten Jahrhunderts als weltweit einheitliche Sprache für Dokumente im Web wie das Zerschlagen eines gordischen Knotens.

Selbsttestaufgabe 2.4:

Was sind die drei grundlegenden Konzepte des frühen HTML-basierten Webs?

Selbsttestaufgabe 2.5:

Charakterisieren Sie das frühe HTML-basierte Web als Hypertextsystem. Wodurch hebt es sich von anderen klassischen Hypertextsystemen ab?

Die HTML-Syntax

Um einen ersten Eindruck von der HTML-Syntax zu bekommen, wird zunächst ein aus Kurseinheit 1 bereits bekanntes Beispieldokument im HTML-Format eingeführt, welches sich im Anhang unter 7.a befindet.

Wird ein HTML-Dokument in den Webbrowser geladen, so wird die bereits formulierte Seite angezeigt. Webbrowser verfügen jedoch auch über eine Funktion, mit der sich der Nutzer den Quelltext anzeigen lassen kann, in der Regel ist das ein sogenannter (Dokumenten-Objektmodell-Inspektor (engl. *document object model Inspector, DOM Inspector*). Alternativ dazu ist es auch möglich, das Dokument lokal zu speichern und dann mit einem Texteditor – wie z. B. Microsoft Notepad, Vi, Edit oder Emacs – zu öffnen.

Aus dem Quelltext unter 7.a wird ersichtlich, dass ein HTML-Dokument einen Textstrom mit eingebetteten Auszeichnungen (Markierungen, engl. *markup*) darstellt und HTML damit eine Auszeichnungssprache (engl. *markup language*) ist. Eingebettetes Markup beschreibt das Dokument als eine hierarchische Struktur von Elementen und Text mit einem Wurzelement; Elemente können weitere Elemente, Text oder auch beides in gemischter Form enthalten.

Die Art und Weise, Formatierungsbefehle für die Elemente eines Dokumentes mit spitzen Klammern zu umgeben, könnte noch aus der ersten Kurseinheit bekannt sein. Dort wurde auch der Begriff Markierung (engl. *tag*) für die Codierung von Formatierungen mithilfe von öffnenden und schließenden Tags, welche den Namen der Formatierungsfunktion in eckigen Klammern beinhalten, eingeführt.

Markup, Tags und
Wohlgeformtheit

Bei der HTML-Syntax lässt sich feststellen, dass z. B. der Inhalt eines Elements XXX an seinem Anfang durch das öffnende Tag `<xxx>` und an seinem Ende durch das schließende Tag `</xxx>` markiert ist. Die Klammerstruktur der öffnenden und schließenden Tags sollte somit wohlgeformt sein. Wird ein Element in dem übergeordneten Element, wo es geöffnet wurde, auch wieder geschlossen, so wird dies ebenfalls als wohlgeformt bezeichnet. Dies ergibt eine hierarchische Strukturierung wie sie bei Umsetzungen des Modells der strukturierten Dokumente gängig ist. Es gibt jedoch auch Situationen in HTML, in denen Tags wie z. B. `<xxx>` oder `</xxx>` wegfallen dürfen. Voraussetzung ist, dass ein fehlendes Tag aus dem Zusammen-

hang des HTML-Codes erschlossen werden kann, wie beispielsweise beim HTML-Element IMG, das Einfügepositionen für Bilder markiert und keinen HTML-Inhalt enthalten kann. Wenn also z. B. direkt nach dem öffnenden Tag kein schließendes Tag folgt, kann HTML-fähige Software den fehlenden Tag ergänzen. Ähnlich liegt der Fall z. B. bei den Aufzählungspunkten einer Liste, die in HTML durch das Element LI (engl. *list item*, Listeneintrag) repräsentiert wird. Das schließende Tag kann wegfallen, da es immer von einem öffnenden Tag für den nächsten Aufzählungspunkt oder von dem schließenden Tag für die ganze Liste (z. B.) gefolgt wird. Bei dem Element BODY können z. B. sogar das öffnende und das schließende Tag wegfallen. Kann ein HTML-Element keinen Textinhalt haben, wie es z. B. bei dem Element IMG der Fall ist, so muss das schließende Tag sogar wegfallen. Die Sprachdefinition von HTML legt genau fest, welche Tags wegfallen dürfen oder wegfallen müssen und an welchen Stellen sie dann von HTML-fähiger Software zu rekonstruieren sind. Die Optionalität einiger Tags ändert nichts an dem Prinzip der hierarchischen Schachtelung von Elementen. HTML-Elemente haben immer einen Namen und können zusätzlich über Attribute verfügen. Attribute haben jeweils einen Namen und einen Wert. Der Name des HTML-Elements erscheint in dem öffnenden und in dem schließenden Tag für das Element. Eventuelle Attribute erscheinen nur in dem öffnenden Tag, und zwar nach dem Elementnamen in der Form <<AName>>="<<AWert>>" bzw. <<AName>>='<<AWert>>' . Dabei darf der Typ von Anführungszeichen (einfach oder doppelt), der den Attributwert begrenzt, im Attributwert nicht im Klartext auftreten. Ein Attributname darf pro Element nur einmal vorkommen; die Reihenfolge der Attribute ist ohne Bedeutung. Das Beispieldokument enthält z. B. ein Element IMG mit Attribut SRC, dessen Wert eine URL für die Bilddatei angibt:

```
IMG SRC="fbLogo.gif"
ALT="Das Logo der Frauenbeauftragten der TUM">
```

Zusammenfassend folgen also öffnende und schließende Tags in HTML den Mustern

```
<<<ENAME>><<AName1>>="<<AWert1>>">
...
<<ANameN>>="<<AWertN>>">
</<<ENAME>>>,
```

wobei die doppelten Anführungszeichen paarweise durch einfache Anführungszeichen ersetzt werden dürfen. Zusätzlich dürfen Attributspezifikationen verkürzt werden. Die Anführungszeichen um die Attributwerte dürfen in Fällen, in denen die Begrenzung des Wertes klar ist, ganz wegfallen. In bestimmten, in der Sprachdefinition von HTML genau angegebenen Fällen, dürfen zusätzlich der Attributname und das Gleichheitszeichen wegfallen, sodass der bloße Attributwert ohne Anführungszeichen das Attribut repräsentiert. Beispiele für Verkürzungen in diesem Sinne sind

```
<TABLE frame=border>
```

und

```
<TABLE border>.
```

Diese stehen für

```
<TABLE frame="border">
```

Die Namen von Elementen und Attributen sowie eine Reihe von Attributwerten können mit Kleinbuchstaben, Großbuchstaben oder einer beliebigen Mischung aus beiden gegeben werden. HTML-Dokumente können außer den Tags noch drei weitere Arten von Markups enthalten: eine Dokumenttypdeklaration, Kommentare und Referenzen.

Die Dokumenttypdeklaration steht immer am Anfang in einem HTML-Dokument, noch vor dem `<html>`-Tag. Es handelt sich hier lediglich um eine Anweisung für den Webbrowser, in welcher HTML-Version die Webseite erstellt wurde. Im alten HTML 4.01 oder XHTML 1.0 benötigte diese `<!doctype>`-Deklaration noch eine Dokumenttypdefinition (kurz DTD), die auf SGML basierte und die Regeln für die Markup-Sprache spezifizierte.

Das folgende Beispieldokument beginnt noch mit der alten Dokumenttypdeklaration:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN"  
      "http://www.w3.org/TR/REC-html40/strict.dtd">.
```

Diese Dokumenttypdeklaration legt fest, welche Version und Variante von HTML verwendet wird. Sie referenziert eine Strukturvorgabe, auch Dokumentengrammatik oder Dokumenttypdefinition genannt, für HTML. Die DTD für die strikte Variante von HTML 4.0 findet sich unter <http://www.w3c.org/> des W3C.

HTML5 wird aber im Gegensatz zu seinen Vorgängerversionen nicht mehr mithilfe des Rahmenwerks von SGML definiert. Somit könnte die `<!doctype>`-Deklaration eigentlich wegfallen, aber Webbrowser benötigen diese Angabe, um zu entscheiden, ob sie in den rückwärtskompatiblen Modus (Quirks-Modus) oder in den standardkompatiblen Modus (Standards-Modus) schalten müssen, um den Inhalt korrekt darzustellen.

Die Deklaration lautet einfach nur: `<!doctype html>`

Praxistipp:

Der neue Dokumenttyp sollte künftig immer verwendet werden, auch wenn noch kein neueres HTML5 verwendet wird. Diese Angabe wird von allen Webbrowsern verstanden – auch von denen, die HTML5 nicht beherrschen.

Kommentare und Referenzen

Kommentare folgen in HTML dem Muster `<!-<<Kommentartext>>->`, wobei in der Zeichenfolge `<<Kommentartext>>` die Zeichenfolge „--“ nicht vorkommen darf. Referenzen schließlich können Zeichenreferenzen oder Entity References sein. Eine Zeichenreferenz folgt dem Muster `&#x<<Codeposition>>;` wobei `<<Codeposition>>` bis zu vier Hexadezimalziffern enthält. Eine Zeichenreferenz referenziert ein Unicode-Zeichen über seine Codeposition. Ein HTML-Dokument darf beliebige Unicode-Zeichen aus dem Bereich von 0 bis FFFF referenzieren.

Praxistipp:

Kommentare sind hilfreich und sinnvoll. Kommentare werden zwar nicht im Webbrowser angezeigt, sind aber im Quelltext enthalten. Da der Quelltext eingesehen werden kann, sollte man beachten, was für Kommentare man verfasst.

Die Bedeutung von Zeichenentitäten in HTML hat speziell mit der Verwendung von UTF-8 erheblich abgenommen, aber es gibt immer noch Gründe für ihre Verwendung. Entity References folgen dem Muster `&<<EntName>>;` und dienen in HTML lediglich als mnemonicische Alternative zu Zeichenreferenzen. HTML definiert eine Reihe solcher mnemonicischen Namen, beispielsweise `Ä` für `Ä;`, um das Zeichen „Ä“ zu referenzieren. Die Entity References `<;>;&;';` und `";` für die Zeichen „<“, „>“, „&“, „““ und „“““ dienen als Escape-Mechanismus, damit diese Funktionszeichen im Inhalt oder in einem Attributwert verwendet werden können. Das syntaktisch unzulässige Tag `<xxx YYY='''>` muss also in der Form `<xxx YYY='';'>` geschrieben werden.

Ein HTML-Dokument besteht aus einer Folge von Unicode-Zeichen mit Code-Positionen im Bereich von hexadezimal 0000 bis FFFF. Nach den oben dargestellten Syntaxregeln wird dieser Strom strukturiert in Markup und Inhaltstext. In bestimmten Kontexten dienen Referenzen als Makros oder Abkürzungen für Unicode-Zeichen. Das HTML-Dokument selbst kann in einem bei der IANA als Charset registrierten Codierungsschema gespeichert und über das Internet übertragen werden.

Praxistipp:

Bis vor kurzem war das IANA-Register das Nachschlagewerk für Bezeichner von Zeichencodierungen. Die neue Spezifikation [Encoding](#)¹ enthält eine Liste, die gegen aktuelle Browserimplementierungen getestet wurde. Es wird dann vor der weiteren Bearbeitung zumindest konzeptionell in eine Folge von Unicode-Zeichen umgewandelt. Das Codierungsschema kann bei der Übertragung über HTTP in einem Headerfeld der übertragenen Nachricht oder in einem Element META im Dokument selbst deklariert werden. Die Deklaration in der HTTP-Nachricht hat dabei Priorität.

¹ <https://encoding.spec.whatwg.org/>

Der Vollständigkeit halber sei gesagt, dass es neben den zwei Möglichkeiten der Angaben zur ZeichenCodierung – HTTP und META – noch einen dritten Weg gibt: Wenn ein Unicode Byte Order Mark (BOM) am Dateianfang steht, hat diese höhere Priorität als jede andere Angabe, was aber in der Praxis nicht empfohlen wird.

Nach dieser ersten ausführlicheren Diskussion der Syntax von HTML-Dokumenten geht es nun weiter mit dem HTML- Dokumentenmodell und der HTML-Semantik.

Ein HTML-Steckbrief

Konzeptionell gesehen ist ein HTML-Dokument eine hierarchische Struktur von logischen Elementen. Es gibt genau ein Wurzelement auf der obersten Hierarchiestufe. Ein Element besteht aus einer eventuell leeren Folge von untergeordneten Elementen und Zeichen, die den Inhalt darstellen. Jedes Element hat einen Namen oder Typ und eventuell Attribute. Ein Attribut hat einen Namen und einen Wert von einem gewissen Typ oder aus einem gewissen Wertebereich. Zeichen können direkt über eine Codeposition oder indirekt über Zeichenreferenzen oder EntitätsReferenzen gegeben sein. Zeichenvorrat ist der gesamte Unicode-Zeichensatz. Diese konzeptionelle Sicht auf HTML kann auch mathematisch formalisiert werden.

Sei DocSort die Menge aller Namen für HTML-Elemente, AttSort die Menge aller Namen für HTML-Attribute, AttValue die Menge aller Werte für HTML-Attribute und Character der HTML-Zeichensatz. Dann bilden die HTML-Dokumente eine Teilmenge von DocInstance := DocSort × Attributes × Content, wobei Attributes definiert ist als die Menge aller partiellen Abbildungen von AttSort nach AttValue vereinigt mit Character* und Content als (DocInstance vereinigt mit Character)*.

Die Formalisierung der zu einem HTML-Strukturelement gehörigen Attribute als partielle Abbildung von AttSort legt in einer Formel fest, was der HTML-Standard umgangssprachlich beschreibt: Attributnamen dürfen in einem Element nicht mehrfach vorkommen. Die Reihenfolge der Attribute, die in der syntaktischen Codierung eines HTML-Elements gegeben ist, ist für die Bedeutung des Elements irrelevant. Die Formalisierung zeigt, dass der HTML-Standard bei Attributwerten zwischen beliebigen Zeichenketten, also Elementen aus Character*, und symbolischen Konstanten, also Elementen aus AttValue, unterscheidet. Wie zwischen diesen beiden Wertebereichen unterschieden werden kann, wird auf der Ebene des Datenmodells nicht thematisiert. Diese Themen werden auf der syntaktischen Ebene der HTML-Dokumente behandelt und sind Gegenstand des HTML-Standards.

Eine konzeptionelle Sicht auf HTML, wie sie hier vorgestellt wurde, nennt man auch ein Datenmodell für HTML. Das Datenmodell abstrahiert von der HTML-Syntax und stellt Begriffe auf einer höheren Abstraktionsstufe zur Verfügung. Weitere Diskussionen zu HTML können auf der bequemeren konzeptionellen Ebene des Datenmodells ablaufen und die Syntax nicht weiter berücksichtigen.

HTML-
Datenmodell

HTML-DOM

Wie bereits erwähnt, ist HTML nicht mehr als eine Anwendung von SGML definiert, sondern als eigene generalisierte Sprache. Die komplette HTML5-Spezifikation ist vom DOM-Thema geprägt. Das HTML-DOM ist somit das Objektmodell und

die Schnittstelle für HTML. Im HTML-DOM wird jedes HTML-Element als Objekt mit allen Eigenschaften, Methoden und Ereignissen definiert. So stellen nicht nur HTML-Elemente selbst Knoten dar, sondern auch die HTML-Attribute und die Inhalte der HTML-Elemente selbst sind Knoten eines DOM-Baumes. Da diese Knoten im Baum in einem ganz bestimmten Verwandtschaftsverhältnis (Eltern, Kind, Geschwister) zueinanderstehen, ist es z. B. mithilfe von JavaScript möglich, mit verschiedenen HTML-DOM-Methoden und HTML- DOM-Eigenschaften auf jeden dieser Knoten zuzugreifen. Somit bietet das HTML-DOM das standardisierte Modell, auf dessen Grundlage HTML-Elemente geändert, eingefügt oder gelöscht werden können.

Der größere Teil der Elemente und Attribute bei HTML beschreibt logische oder funktionale Aspekte eines Dokuments. Es gibt jedoch auch eine Reihe von Sprachmitteln in HTML, welche die Formatierung betreffen. Beispiele für logisch ausgerichtete Sprachmittel sind die Elemente P für Absätze und H1 für Überschriften ersten Grades sowie das Attribut HREF für URLs von Hypertextankern. Beispiele für auf das Format ausgerichtete Sprachmittel sind die Elemente I und B für Kursiv- und Fettchrift und das Attribut STYLE für Formatvorgaben.

Für logisch ausgerichtete Sprachmittel legt jeder Browsertyp individuell fest, wie er die Elemente formatiert. Auf das Format ausgerichtete Sprachmittel geben den Browsern Vorgaben zur Formatierung; sie können die Formatierung aber nicht vollständig bestimmen. Auch Formatvorgaben durch sogenannte kaskadierte Stilvorgaben (engl. *cascading stylesheets, CSS*), geben lediglich einen Rahmen oder gewisse Beschränkungen für die Formatierung an. Sie legen das Format aber nicht bis in das letzte Detail fest. Schließlich wird die Formatierung eines HTML-Dokuments noch durch konfigurierbare Voreinstellungen im Browser beeinflusst. Hierunter fallen etwa die Fenstergröße oder die konfigurierbare Größe der Grundschrift. Ein- und dasselbe HTML-Dokument kann folglich mit verschiedenen Browsern ganz unterschiedlich dargestellt werden.

An dieser Stelle sei noch einmal darauf hingewiesen, dass für die Textformatierung CSS zuständig ist. Der HTML5- Standard empfiehlt, das b- bzw. i-Element nur noch in Fällen einzusetzen, wenn kein anderes HTML-Element zur Auszeichnung mehr passt. Die Elemente in HTML, welche auf das Format ausgerichtet sind, dienen einer sauberen semantischen Textauszeichnung und legen so den Grundstein für eine spätere Textformatierung mit CSS.

Der Sprachschatz von HTML

Ein HTML-Dokument ist nach dem folgenden Muster aufgebaut:

```
<HTML>
```

```
<HEAD><<...>></HEAD>
<BODY><<...>></BODY>
</HTML>
```

Es hat also ein Wurzelement mit Namen HTML und zwei Kindelemente, HEAD für den sogenannten Kopfdatenbereich und BODY für den Inhaltbereich des Dokumentes.

Im Kopfdatenbereich des HTML-Dokumentes zwischen den Tags `<head>` und `</head>` können verschiedene HTML-Elemente eingefügt werden, mit denen der Inhalt und die Darstellung einer Webseite beeinflusst werden können. Das Element HEAD enthält z. B. Verwaltungsinformation zum Dokument, etwa seinen Titel, Schlüsselwörter für Suchmaschinen und andere Daten. Auch werden in diesem Bereich die Beziehungen zwischen dem Webbrowser und anderen Seiten oder Dokumenten definiert. Mit Ausnahme des Titel-Elements `TITLE` wird vom Webbrowser kein Inhalt aus dem Kopfdatenbereich angezeigt. Das Titel-Element kann zwar weg gelassen werden, aber dann handelt es sich um kein valides HTML und außerdem hat der Titel bei den Suchmaschinen eine hohe Bedeutung.

Kopfdatenbereich

Die weiteren Kindelemente von HEAD sind META, BASE, STYLE, LINK, SCRIPT und OBJECT. BASE darf höchstens einmal vorkommen; die übrigen Elemente dürfen beliebig häufig auftreten. Die Kindelemente von HEAD dürfen in beliebiger Reihenfolge auftreten.

Das Element META stellt zu einem HTML-Dokument Metainformation in Form von Eigenschaften zur Verfügung. Eine Eigenschaft hat einen Namen, gegeben durch das META-Attribut NAME, und einen Wert, gegeben durch das META-Attribut CONTENT. Die Namen und Wertebereiche der Eigenschaften sind frei. Wir könnten also die Versions- und Datumsinformation unseres Beispieldokuments mit META-Elementen folgendermaßen ausdrücken:

```
<META NAME="Version" CONTENT="1.3">
<META NAME="Date" CONTENT="12.10.1999">
```

In bestimmten Fällen sollte anstelle des Attributes NAME von META das Attribut HTTP-EQUIV eingesetzt werden, dann nämlich, wenn ein Webserver die Information für eine HTTP-Nachricht verwenden sollte oder wenn ein Webbrowser fehlende Information in der HTTP-Nachricht ergänzen muss. Ein Beispiel ist das für das Dokument verwendete Codierungsschema, also etwa

```
<META HTTP-EQUIV="Content-Type"
CONTENT="text/html; charset=ISO-8859-1">
```

Mit HTML5 wurde auch eine dritte Möglichkeit eingeführt, mit welcher die Zeichen-Codierung festgelegt werden kann und im sogenannten Meta-Tag notiert wird:

```
<meta charset="UTF-8">
```

Praxistipp:

Das Element TITLE und die Beschreibung (name="description") sind häufig das Erste, was von den Suchmaschinen zurückkommt, wenn die Website bei einer Suche aufgelistet wird.

Der Inhaltsbereich des Dokumentes

HTML-Inhaltsbereich

HTML-Dokumentkörper

Alle im Inhaltsbereich des HTML-Dokuments zwischen <body> und </body> befindlichen Elemente werden vom Webbrowser gerendert und angezeigt. Man spricht auch vom HTML-Dokumentkörper, in den alle HTML-Elemente wie z. B. Text, Hypertextlinks, Tabellen, Bilder etc. geschrieben werden, um die Struktur der Webseite festzulegen.

Die neuen Sektionselemente von HTML <section>, <article>, <aside> und <nav> dienen vorwiegend dazu, dem Inhalt eine Bedeutung zu geben, insbesondere ist es eine Hilfe für Suchmaschinen und auch Screenreader. Im Kapitel „Das neue semantische HTML“ wird auf diese Elemente nochmal eingegangen.

Zwei neue semantische HTML-Elemente sind auch für den Kopf- und Fußbereich neu hinzugekommen. Das <header>-Element kann für eine Seitenüberschrift, den Namen der Webseite oder eine Navigationsleiste verwendet werden. Im <footer>-Element können solche Angaben wie Impressum, rechtliche Informationen etc. platziert werden.

Ausführlichere Information zu den hier besprochenen Sprachelementen oder auch zum vollen Sprachumfang von HTML5 befinden sich in der weiterführenden Literatur. Ein geeigneter Ausgangspunkt für eigene Recherchen im Web ist außerdem die Web-Site des W3C zu HTML unter <http://www.w3.org/html/>.

Es gibt also zahlreiche, vor allem neue interessante HTML-Elemente und HTML-Attribute, wie z. B. das <time>-Element für gültige maschinenlesbare Datums- und Zeitangaben oder auch die neuen HTML5-Eingabefelder mit <input>, die im weiteren Verlauf der Kurseinheit nicht im Einzelnen besprochen werden können. Dazu kann man eine der vielen HTML-Einführungen heranziehen und auch im Internet nach HTML-Kursen recherchieren. Anstelle einer detaillierten Beschreibung einzelner HTML-Elemente wird im weiteren Verlauf vielmehr eine systematische Übersicht über den Sprachschatz von HTML gegeben.

Selbsttestaufgabe 2.6:

Öffnen Sie ein Dokument, das Sie in Ihrem Textsystem geschrieben haben. Übertragen Sie den Text des Dokumentes nach HTML. Repräsentieren Sie alle Strukturen Ihres Ausgangsdokuments im HTML-Markup.

Das neue semantische HTML

Meistens sind in HTML 4.01 die Elemente entweder Block-Elemente (z. B. address, div, h1-h6 etc.) oder Inline-Elemente (z. B. a, img, abbr etc.). Innerhalb des body-Elements wurde alles als Flow-Content bezeichnet. Elemente mit einer neuen Zeile oder Absatz im Textfluss gelten als Block-Elemente, welche neben Text wiederum Block- und Inline-Elemente haben können. Block-Elemente ähneln einem Absatzformat. Inline-Elemente sind reine Auszeichnungen innerhalb des Textes. Die wenigen Ausführungen sollen zum alten Content-Modell genügen.

In HTML5 existiert ein komplett anderes Content-Modell, weil bei der Strukturierung eines HTML-Dokumentes die semantische Bedeutung von HTML hervorgehoben wird. Das ist gerade mit Blick auf das Informations- und Wissensmanagement im Internet ein entscheidender Schritt in diese Richtung.

| semantisches HTML |

Es existieren sieben quasi semantische Content-Modelle: Flow-Content, Sectioning-Content, Heading-Content, Phrasing-Content, Embedded-Content, Interactive-Content und Metadata-Content.

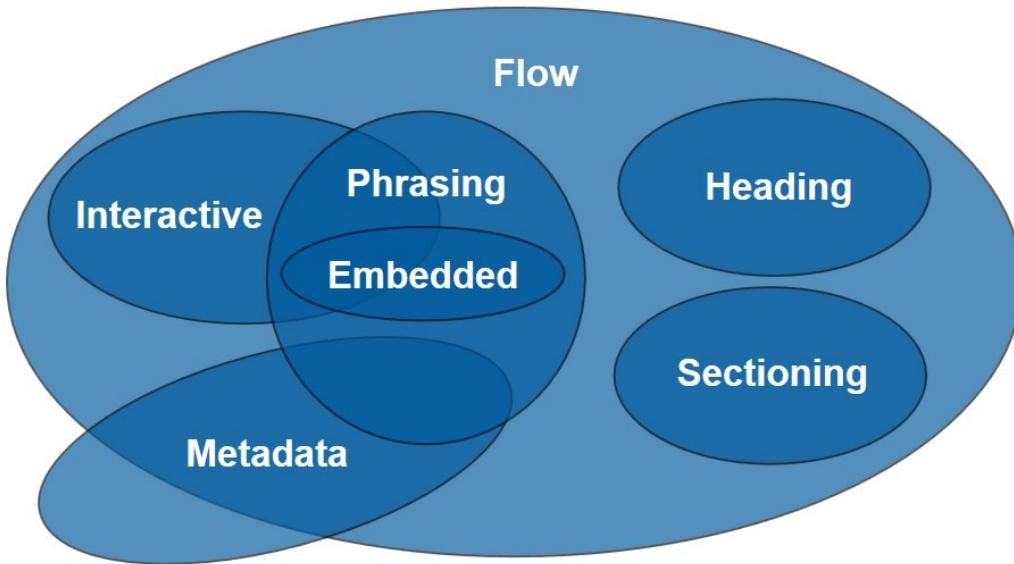


Abbildung 3.2 Das neue Content-Modell für HTML [W3Chtml5CM]

Die meisten body-Elemente gehören zum Flow-Content.

Mit dem Sectioning-Content lassen sich spezielle Sektionen im HTML-Dokument festlegen, um die semantische Struktur zu verbessern. Realisiert werden kann das mit den neuen Elementen article, aside, nav und section. Innerhalb von Sectioning-Elementen wird eine eigene Überschriftenstruktur verwendet.

Die sechs Überschreitebenen h1 bis h6 gehören zu den Heading-Elementen. Die Strukturierung ist von Sectioning-Elementen abhängig.

Die Phrasing-Elemente können verglichen werden mit den Inline-Elementen aus HTML 4.01, also neben normalem Text gehören Elemente zur Textauszeichnung innerhalb eines Absatzes bzw. des Dokumentenkörpers.

Mit Elementen, die zum Embedded-Content gehören (z. B. audio, canvas, svg, object etc.), können Grafiken, Videos und andere Markup-Inhalte etc. eingebunden werden.

Elemente aus der Interactive-Content-Kategorie bieten dem Benutzer die Möglichkeit zur Interaktion. Das sind beispielsweise button, details, input etc.

Die Elemente des Metadata-Content wurden bereits zu Ausführungen bzgl. des <head>-Tag im Kopf-Bereich eines Dokumentes erwähnt. Solche Metadata-Elemente können aber auch zwischen <body> und </body> platziert werden, häufig findet man dort z. B. das <script>-Element.

Empfehlungen zur Verwendung von HTML

Empfehlungen zum Semantischen HTML

Wie gerade erwähnt, wurde das neue semantische Content-Modell eingeführt, um HTML semantisch anzureichern. Hierzu soll es nun noch einige Empfehlungen geben.

Aber es soll auch Schluss sein mit dem ungehemmten Einsatz von div-Elementen und class-Attributnen. Div-Elemente und die Auszeichnung der Layoutbereiche mit ID- und Klassennamen sind HTML5-konform, also nicht verboten, dennoch sollten die neuen semantischen Elemente verwendet werden. Zum Beispiel ist es auch für Suchmaschinen, Screenreader, für Entwickler der Webseiten etc. besser, wenn sie z. B. wissen, was zum Hauptinhalt gehört.

An diesem Beispiel erkennt man, wie wichtig es ist, einem HTML-Dokument eine semantische und logische Struktur zu geben, auch wenn der „normale“ Nutzer im Webbrowser eigentlich keinen Unterschied bemerkt.

Auch sollten die neuen Sektionselemente von HTML <section>, <article>, <aside> und <nav> unbedingt genutzt werden.

Mit dem <section>-Element kann man den Inhalt des Dokuments in themenbezogene Abschnitte einteilen, z. B. kann eine Unterteilung in Kapitel und Unterkapitel erfolgen oder inhaltsbezogene Abschnitte erstellt werden. Das <article>-Element kann verwendet werden, um einen meist eigenständigen Inhalt zu einem Block zusammenzufassen. Um Inhalte mit zusätzlichen Informationen oder mit einer Seitenleiste zu versehen, dient das <aside>-Element. Das <nav>-Element eignet sich z. B. für das Zusammenfassen der Hauptnavigation einer Webseite.

Zum Schluss sollen noch die sogenannten ARIA Landmark Roles genannt werden. Der ausgeschriebene Begriff zur Abkürzung ARIA ist Accessible Rich Internet Application. Barrieararmut und Barrierefreiheit sollten heutzutage bei der Erstellung von

Webseiten keine Fremdwörter mehr sein. Es handelt sich hier also um eine technische Spezifikation, um behinderten und älteren Menschen Webseiten und Webanwendungen besser zugänglich zu machen. Für solche Orientierungspunkte wurden die HTML-Elemente um das role-Attribut erweitert. Weitere Informationen zu diesem Thema können unter http://www.w3.org/TR/wai-aria/roles#roles_categorization nachgelesen werden.

Praxistipp:

Natürlich ist es möglich mit alten HTML-Elementen und HTML-Attributen z. B. Texte ohne CSS-Angaben in fett oder kursiv darzustellen. In der Praxis sollte die Auszeichnungssprache HTML aber lediglich dazu dienen, ein HTML-Dokument mit den HTML-Elementen semantisch zu strukturieren. In HTML5 sind viele dieser Elemente als *obsolete* (bis HTML 4: *deprecated*), also veraltet, eingestuft; da sie sich auf das Format und nicht auf die logische Struktur von HTML-Elementen beziehen. Sie werden in Zukunft aus dem Standard entfernt. Die gleichen Effekte dieser Sprachelemente können auch mit Cascading Style Sheets erreicht werden, die in dieser Kurseinheit noch näher erklärt werden.

Empfohlene Syntaktische Konventionen

HTML bietet eine Reihe von syntaktischen Freiheiten, die XML aufhebt. Diese Freiheiten können in XHTML nicht mehr genutzt werden. Um einen möglichst glatten Übergang von HTML zu XHTML zu ermöglichen wird die Einhaltung der folgenden Konventionen für ein sogenanntes polyglottes Markup (in KE3 wird näher darauf eingegangen) empfohlen, die im Folgenden kurz dargestellt werden.

empfohlene syntaktische Konventionen

Auf Markupminimierung sollte verzichtet werden. Für alle Elemente sollten öffnende und schließende Klammern verwendet werden. Alleinstehende Elemente, leere Elemente und void-Elemente müssen in XHTML geschlossen werden, daher sollte die folgende Tagvariante verwendet werden:

```
<«ENAME» «ATTRIBUTE» />
```

Leerzeichen sollten vor dem Solidus stehen. In XHTML darf man bei einem alleinstehenden Element einfach noch ein Ende-Tag hinzufügen, wie z. B.
</br>, was in HTML nicht erlaubt ist. Aus diesem Grund sollten bei polyglottem Markup alleinstehende Elemente mit dem Schrägstrich schließen, wie z. B.
, weil ein HTML-Parser ein
 wie
 behandelt. Hier noch eine Auflistung aller leeren Elemente, die davon betroffen sind:

```
<area>; <base>; <br>; <col>; <embed>; <hr>; <img>;  
<input>; <keygen>; <link>; <meta>; <param>; <source>;  
<track>, <wbr>
```

Bei Attributen sollten immer Anführungszeichen und nie abkürzende Formen verwendet werden. Auch Zeilenumbrüche innerhalb von Wertzuweisungen von Attributen sollten vermieden werden. Leerzeichen sind aber erlaubt.

Alleinstehende Attribute dürfen in XHTML nicht minimiert notiert werden, sondern wie folgt geschrieben werden:

```
<input type="text" name="fernuni" disabled="disabled" />
```

In HTML können die Namen von Elementen und Attributen sowie bestimmte Arten von Attributwerten gleichermaßen mit Großbuchstaben oder mit Kleinbuchstaben geschrieben werden. Bei der XML-Syntax, und somit auch in XHTML, ist die Klein- und Großschreibung jedoch signifikant. XHTML-Namen sind dabei alle auf Klein-schreibung festgelegt. Somit ist es angebracht, auch in HTML Namen von Elementen und Attributen klein zu schreiben, obwohl viele Nutzer sich in HTML-Dokumenten mit großgeschriebenen Namen besser zurechtfinden.

Inhalte von Skript- und Style-Bereichen sollten in CDATA-Abschnitte eingeschlossen werden. Folgende Notation verursacht in HTML keine Probleme:

```
<script>
  Alert("Hier die Ungleichung: 1 < 2");
</script>
```

In XHTML bekommt man hier Probleme mit der spitzen Klammer, weil es ein HTML-eigenes Zeichen ist. Schließt man den Inhalt wie folgt in einen CDATA-Abschnitt ein, meldet der XML-Parser keinen Fehler:

```
<script>
  <! [CDATA[
    Alert("Hier die Ungleichung: 1 < 2");
  ]]>
</script>
```

Die Möglichkeiten von JavaScript bzw. CSS sind bei polyglottem Markup noch viel einfacher, als die Verwendung von CDATA.

Außerdem sollte in XHTML auf das `<noscript>`-Element verzichten werden.

Die letzte Empfehlung bezieht sich darauf, dass man Dokumente mit unterschiedlichen Browsern und Browerversionen auf verschiedenen Plattformen testen und sie auch formal auf ihre syntaktische Korrektheit überprüfen und validieren sollte. Validierer oder Syntax-Checker sind als lokal zu installierende Werkzeuge oder als Web-anwendungen erhältlich. Mit ihnen werden HTML-Syntaxregeln – soweit sie in SGML-Syntax formulierbar sind – überprüft.

Die meisten HTML-Browser sind fehlertolerant. Solange nur reine HTML-Dokumente mit Webbrowsern dargestellt werden sollen, ist das Validieren nicht so essentiell.

Es gibt jedoch gute Gründe, den HTML-Code zu validieren:

- Sobald weitere Sprachen mit HTML kombiniert werden sollen – etwa in Stylesheets – werden Syntaxfehler leicht zur Stolperfalle.
- Gerade bei mobilen Geräten sind weniger mächtige Fehlerkorrekturroutinen im Webbrowser integriert.
- Suchmaschinen müssen mit der Webseite umgehen können.
- Mit Blick auf die barrierefreie Webseite gilt, dass fehlerhafte Textauszeichnungen z. B. auch die Funktionalitäten bei Vorlesesoftware einschränken können.

Validieren ist also wichtig und zudem ohne großen Aufwand zu machen.

Um Fehler in Syntax oder Semantik von Dokumenten aufzudecken, sollte man die erstellten Dokumente mit unterschiedlichen Browsern anschauen.

URL, URN und URI

Jedes HTML-Dokument sowie auch andere Ressourcen im Web sind jeweils über eine individuelle Adresse ansprechbar. Das ursprüngliche Format für Webadressen heißt Uniform Resource Locator (URL). URLs codieren einen Zugriffsweg auf die Ressource. Dadurch ergeben sich zwei grundlegende Probleme. Erstens kann eine URL nicht als eindeutiger Name für die Ressource fungieren und zweitens wird die Adresse ungültig, wenn die Ressource ihren Speicherort ändert. Im W3C wird deshalb unter der Bezeichnung Uniform Resource Name (URN) ein weiteres Format diskutiert. Allerdings ist diese Diskussion noch nicht vollständig abgeschlossen, so dass URNs im Web derzeit hauptsächlich im Umfeld von Anwendungen digitaler Bibliotheken zur Verfügung stehen. Der Oberbegriff zu URLs und URNs ist Uniform Ressource Identifier (URI).

URL, URN und URI

Eine URL besteht aus zwei Teilen, einem Namen – auch Schema genannt – für das Protokoll, über das auf die Ressource zugegriffen werden soll, und einen schemaspezifischen Teil. Die beiden Teile sind durch einen Doppelpunkt getrennt. Das Muster für eine URL ist also

«Schema»:«SchemaSpezifisches» .

Von besonderem Interesse ist der Zugriff über das HTTP-Protokoll, also das Muster:

http: «SchemaSpezifisches»

Die schemaspezifische Information besteht aus dem Namen des Rechners, der die Ressource beheimatet, und dem Namen der Ressource selbst. Eine HTTP-URL kann absolut oder relativ sein.

Eine absolute HTTP-URL folgt dem Schema: http: «Server» oder http: «Server»/«Pfad», wobei «Server» der Domain-Name oder die IP-Adresse des Internetrechners ist, der den Webserver beheimatet, optional gefolgt von „:“ und der Port-Nummer des Webservers. Die Default-Port-Nummer für Webserver ist 80. Der Pfad der Ressource besteht aus einer Folge von Segmenten, die durch „/“ voneinander getrennt sind.

Eine absolute URL benennt eine Datei oder ein Verzeichnis im Dateisystem des Webservers. Dazu ist auf der Maschine des Webservers ein Verzeichnis als Webverzeichnis konfiguriert, beim Apache-Webserver unter Linux typischerweise /usr/local/etc/httpd/htdocs, von dem aus sich die Adresse der gewünschten Ressource aus den Pfadangaben in der URL lesen lässt.

Zwei Sonderfälle sind bei der Auflösung einer URL noch zu beachten:

Endet im ersten Fall eine URL mit dem Solidus („/“) und benennt der Teil vor dem Solidus ein Verzeichnis, so wird ein konfigurierbarer Dateiname – etwa index.html – als Default-Dateiname an die URL angehängt. Existiert keine solche Datei in dem Verzeichnis, so liefert der Webserver die Ansicht auf ein Dateiverzeichnis zurück. Es bietet sich an, die Datei index.html als Einstiegspunkt für ein aus mehreren Knoten bestehendes Hypertextdokument zu verwenden. Darüber hinaus könnte eine leere Datei index.html in allen Verzeichnissen angelegt werden. Dies schützt vor ungewollten Blicken auf die Dateistruktur. Das Verhalten des Browsers bezüglich der Datei index.html kann allerdings bei unterschiedlichen Webservern abweichen und individuell konfiguriert werden.

Beginnt im zweiten Fall das erste Segment eines Pfades mit einer Tilde ~, so wird der Rest des Segments als Nutzerkennung interpretiert und der Pfad hat seinen Ursprung in einem konfigurierbaren Verzeichnis, typischerweise HTML-Data, in dem zur Kennung gehörigen Home-Verzeichnis. Auf diese Weise können Webdokumente lokal in einem eigenen Verzeichnis abgelegt werden.

Eine relative URL besteht nur aus durch den Solidus / getrennten Pfadsegmenten. Sie wird durch Anhängen an eine absolute Basis-URL zu einer absoluten URL komplettiert. Eine relative URL darf Pfadsegmente „..“ und „...“ enthalten; die beiden Segmente bezeichnen das aktuelle Verzeichnis und das übergeordnete Verzeichnis eines hierarchischen Dateisystems. Relative URLs machen ganze Webbereiche portabel.

Sowohl einer absoluten als auch einer relativen URL kann ein sogenannter Fragmentidentifikator in der Form

#«FragmentID»

angehängt sein. Mit diesem Mechanismus lassen sich benannte Teilbereiche eines Dokuments ansprechen.

URLs sind Sequenzen mit Zeichen aus einem kleinen Zeichensatz, bestehend aus den Buchstaben „a“–„z“ und „A“–„Z“, den Ziffern „0“–„9“, den grafischen Symbolen „-“, „_“, „.“, „!“, „~“, „*“, „'“, („ und)“ und den Funktionszeichen

„;“ „/“ „?“ „:“ „@“ „&“ „=“ „+“ „\$“ und „.“. Die Verwendung der Funktionszeichen in Datenbereichen ist durch die Syntax von URLs eingeschränkt. Es gibt aber einen Escape-Mechanismus mit dem Prozentzeichen „%“ als Escape-Zeichen. Bei der Auflösung einer Escape-Codierung zu einer URL wird die Sequenz %XY ersetzt durch das Zeichen an Unicode Codeposition XY. Umgekehrt wird ein Funktionszeichen mit Codeposition XY im Namen einer Ressource, wenn es im entsprechenden Datenteil einer URL nicht erlaubt ist, in der URL durch %XY ersetzt.

Außer dem Schema http sind noch die folgenden URL-Schemavarianten in Gebrauch:

```
mailto: «eMailAdresse»  
mailto: «eMailAdresse»?subject=«eMailSubject»  
news: «newsGroup»  
ftp:// «Server»  
ftp:// «Server»/«Pfad»  
file:// «Pfad»  
telnet://«Server»
```

Die gängige Verwendung einer URL in einem HTML-Dokument ist als Wert des Attributs *HREF* in einem Hypertextanker A oder als Wert des Attributs SRC einer Bildreferenz IMG. Dabei verwenden relative URLs in erster Präferenz das Attribut HREF des Elements BASE im HEAD des aktuellen Dokuments oder in zweiter Präferenz die URL des aktuellen Dokuments als Basis, um sich zu einer absoluten URL aufzulösen.

HTML-Werkzeuge

Ein guter Editor ist zunächst einmal – egal für welchen Typ von Daten – auf die persönlichen Präferenzen und Arbeitsstile zugeschnitten. Bei HTML-Editoren ist das nicht anders. Es sollen deshalb an dieser Stelle nur einige Klassen von HTML-Editoren mit einigen typischen Vertretern genannt werden, sodass jeder die verschiedenen Typen selbst ausprobieren kann.

HTML-Werkzeuge

Eine einfache Methode ein HTML-Dokument zu erstellen ist es, einen Texteditor zu verwenden, also z. B. Notepad unter Windows, Edit unter MSDOS, vi oder emacs unter UNIX oder Textedit bei Mac-OS. Mit einem solchen Werkzeug werden sowohl der eigentliche Inhalt als auch die Markups direkt als Text eingegeben. Mittlerweile gibt es eine Menge kostenloser und kommerzieller HTML-Editoren. Zahlreiche HTML-Editoren bieten WYSIWYG-Oberflächen um HTML-Dokumente zu erstellen und zu bearbeiten. Dazu gehören auch die in den Mozilla-Browser und Chrome integrierten Editoren. Der derzeit bekannteste WYSIWYG-Editor ist Dreamweaver von Adobe.

Es sind auch Werkzeuge verfügbar, mit denen sich kleine oder große Web-Sites verwalten lassen. Die Funktionalität solcher Werkzeuge umfasst die grafische und editierbare Darstellung einer gesamten Web-Site mit Verzeichnis und Linkstruktur. Auch die Aktualisierung von Links bei Verschiebungen von Webseiten oder Verzeichnissen wird gewährleistet. Zudem existiert ein Content Management über Templates für Import aus Datenbanken und Formatvorgaben sowie die Konfiguration des Webservers als auch die Analyse der Zugriffe.

Praxistipp:

Zum Erlernen von HTML sind WYSIWYG-Editoren weniger gut geeignet, zumal man auch nicht beurteilen kann, wie gut der von diesen Programmen erzeugte HTML-Code wirklich ist. Verfügt man über mehr Erfahrung, dann kann man diese Editoren nutzen. Zum Erlernen von HTML sind Editoren mit Syntaxunterstützung und -überprüfung die bessere Wahl, und dazu gehören z. B. solche Vertreter wie Notepad++ oder HTMLPad unter Windows, Bluefish unter Linux oder Netbeans IDE unter Windows, Linux und Mac.

Weitverbreitete HTML-Browser sind der Firefox-Browser und der Microsoft-Internet-Explorer. Das frühere „Sorgenkind“ Microsoft-Internet-Explorer gehört schon fast der Vergangenheit an. In Windows 10 ist Microsoft-Edge (früher unter dem Codenamen „Spartan“ bekannt) ein fester Bestandteil, will den Internet Explorer vergessen machen und ist eine leichtgewichtige Alternative dazu. Microsoft-Edge wirkt frischer als der Internet-Explorer, auch wenn der Browser in der aktuellen Version in Sachen Funktionsumfang noch längst nicht an Firefox oder Chrome heranreicht.

Alle gängigen Textverarbeitungssysteme bieten heutzutage den Export nach HTML an. Typischerweise verwendet der dabei erzeugte HTML-Code in erheblichem Umfang die nicht empfohlenen typografischen Gestaltungsmöglichkeiten von HTML, um ein dem Original möglichst ähnliches Erscheinungsbild zu erzeugen. In der Regel ist jedoch auch die Information über im Original verwendete Druckformatvorlagen in der HTML-Version repräsentiert. Es ist also prinzipiell möglich, eine solche HTML-Version von unerwünschten Sprachkonstrukten zu bereinigen.

HTML-Browser sind, wie bereits betont wurde, typischerweise sehr liberal mit der Syntax von HTML. In vielen Fällen interpretieren sie ein HTML-Dokument trotz syntaktischer Fehler korrekt. Die falsche Schachtelung

`Fettschrift, durchsetzt mit <I>Kursivschrift.</I>`

beispielsweise wird vom Internet-Explorer korrekt umgesetzt. Andere Werkzeuge sind jedoch nicht so liberal. Fehlerhaft codierte CSS-Stylesheets können unerwartete Effekte haben, wenn z. B. auf der HTML-Ebene Syntaxfehler vorliegen. HTML-Dokumente sollten deswegen formal validiert werden. Dazu stehen im Netz sogenannte Online-Validierer zur Verfügung, von denen zunächst zwei empfohlen werden können. Der Validierer der Web-Design-Group unter <https://www.htmlhelp.com/tools/> ist sehr zuverlässig und bietet eine gute Oberflä-

che. Der Validierer unter <http://validator.w3.org> entspricht den aktuellen Regularien des W3C

4 Cascading Stylesheets (CSS)

Tim Berners-Lee hat HTML für den Austausch von Information konzipiert. Es ist zu diesem Zweck nicht notwendig, die Präsentation bis ins Detail festzulegen. Ein und dasselbe Dokument kann sogar in ganz unterschiedlichen Formaten daherkommen, ohne dass sich sein Informationsgehalt ändert. Dementsprechend legt die Definition der HTML-Elemente eine bestimmte Art der grafischen Formatierung zwar nahe, spezifiziert sie aber nicht vollständig. Das Format ein und desselben HTML-Dokuments kann auf verschiedenen Plattformen unter verschiedenen Browsern und bei verschiedenen Einstellungen der Nutzerpräferenzen unterschiedlich ausfallen. Ganz im Sinne des modernen Dokumentenbegriffs kommt es lediglich darauf an, dass der Informationsgehalt des Dokuments vermittelt wird. Dies ist für so manch einen Designer zwar ein Ärgernis, entspricht aber der Philosophie der Textgestaltung.

Stylesheets für HTML

Es hat sich schnell herausgestellt, dass das pure Konzept von HTML zur Codierung von Information die Anforderungen der Webanwenderinnen und -anwender nicht erfüllt. Von Anbeginn der Geschichte des Webs wurde daher der Ruf nach grafischen Gestaltungsmöglichkeiten laut. Beispielsweise ist für Firmen ein Webauftritt nur dann sinnvoll, wenn sie ihre sogenannte Corporate-Identity vermitteln können. Sobald es in die kommerziellen Bereiche geht, ist die individuelle Darstellung von Webseiten nicht mehr gewünscht. Auch im Modell der strukturierten Dokumente wird lediglich beschrieben, dass die Vorgaben, wie das Format aussehen soll, vom eigentlichen Dokument getrennt zu halten sind.

Es etablierten sich zunächst drei unterschiedliche Techniken, um das ursprüngliche Konzept von HTML zu umgehen, und die Gestaltung von Webseiten zu beeinflussen.

Die erste Technik macht sich Fremdformate zunutze, in denen formatierte Dokumente codiert werden können, beispielsweise PostScript und PDF oder Grafikformate wie GIF und JPEG. Diese Daten werden, mit entsprechenden MIME-Kennzeichnungen versehen, über das HTTP-Protokoll ausgeliefert. Sie wurden zunächst in sogenannten externen Helper-Applikationen angezeigt, die von Webbrowsern selbstständig gestartet werden konnten. Die meistgebrauchten Helper-Applikationen sind inzwischen als Browser-Plug-in umgesetzt, die ihre Daten sogar innerhalb des Browserfensters darstellen können. Die Firma Macromedia z. B. entwickelte für ihr multimediafähiges Autorenwerkzeug Director ein eigenes Datenformat Flash, das für die serielle Datenübertragung über das HTTP-Protokoll oder andere Internetprotokolle optimiert ist und das auf der PC-Plattform mit Browser-Plug-in dargestellt werden

kann. Das „alte“ HTML hatte für die Einbettung von Fremddaten das Element OBJECT angeboten. Mit dieser Technik sind allerdings die Dokumentinhalte für weitergehende Webanwendungen, etwa Suchunterstützung (engl. *information retrieval, IR*) verloren gegangen. Die zweite Technik missbraucht ursprünglich zur Beschreibung von Strukturen gedachte Sprachmittel in HTML für grafische Zwecke. Die dritte Technik, die eine Zeit lang von den Browserentwicklern angewendet wurde, erweiterte HTML um neue, browserspezifische Sprachmittel, mit denen sich Formatvorgaben ausdrücken lassen. Beispiele sind das Element FONT, das von Microsoft eingeführt wurde, und das Element LAYER, das auf Netscape zurückgeht. Viele dieser Erweiterungen fanden schließlich Eingang in die HTML-Version 3.2. Dieser sogenannte „Browser-Krieg“ hat lange Zeit die universelle Nutzbarkeit von Webdokumenten gefährdet und verstößt gegen das Modell der strukturierten Dokumente, das Formatvorgaben von Inhalt und Struktur trennt.

|css

Als Reaktion auf diese Entwicklungen verabschiedete das W3C im Dezember 1996 eine eigene Sprache für Formatvorgaben für HTML, nämlich die Cascading Style Sheets. Hand in Hand damit ging die Verabschiedung von HTML in der Version 4, in der formatspezifische Sprachelemente als deprecated eingestuft sind und in der die Anbindung von Stylesheets an HTML-Dokumente standardisiert ist.

An dieser Stelle muss kurz zum neuen HTML5 zurückgekehrt werden, bevor auf CSS weiter ausführlich eingegangen wird. Seit HTML5 ist eine Trennung von Strukturierung und Layout durch die Definition mit CSS gegeben. Alle Attribute zur Formatierung aus „altem“ HTML, außer dem Element <border> für Rahmen, werden in HTML5 nicht mehr unterstützt, auch für Tabellen werden keinerlei Formatierungsmöglichkeiten mehr angeboten. Sogar die Verwendung von multimedialen Inhalten und Grafiken ist in HTML5 sehr viel einfacher geworden:

- Mithilfe von MathML (Mathematical Markup Language) lassen sich mathematische Formeln direkt integrieren.
- Das neue <canvas>-Element ermöglicht Zeichnungen, Animationen usw. aber mit <canvas> kann man noch viel mehr machen, wie Transformationen, Schatten, Kurven u v. m.
- Das neu in HTML5 eingeführte video-Element ermöglicht das Abspielen von Videos ohne Plug-in wie z. B. Flash im Webbrowser.
- Das neue Element <audio> funktioniert nach demselben Muster wie <video>, nur eben für Audiodateien.
- Um Objekte in HTML einzubetten gibt es <embed>, <object> oder auch <iframe>. Welches Element man in der Praxis bevorzugt, ist eher eine Frage des persönlichen Geschmacks.

Es wurde nun deutlich, dass man für die Erstellung einer modernen Webseite heutzutage mehrere grundlegende Komponenten benötigt: Für den Inhalt verwendet man HTML, Mediadateien können in das HTML-Dokument eingebettet werden, für spezielle Aktionen verwendet man Skripte und für das Erscheinungsbild CSS (auch Stylesheets). Mit CSS lassen sich für HTML-Dokumente separate und auswechselbare Formatvorgaben definieren, ohne Inhalt und Struktur der Dokumente zu verwässern. HTML und CSS können sich nun die Vorteile des Modells der strukturierten Dokumente zunutze machen:

- Das Quellformat enthält Strukturelemente, die für eine automatisierte Verarbeitung des Dokuments nutzbar sind.
- Formatvorgaben für die einzelnen Elemententypen unterstützen eine konsistente Formatierung innerhalb von Dokumenten.
- Die Anwendung einer separaten Formatvorgabe für eine Gruppe von Dokumenten unterstützt eine konsistente Formatierung dieser Dokumente.
- Die Anwendung verschiedener Formatvorgaben auf ein Dokument ermöglicht die flexible Darstellung und Nutzung des Dokuments je nach Anwendungssituation.

Im weiteren Verlauf der Kurseinheit wird nun das Thema CSS vom Konkreten zum Allgemeinen eingeführt. Beginnend mit einem Beispiel und einer Einführung in die Syntax von CSS werden dann der Sprachschatz und zuletzt die Verarbeitung von CSS-Stylesheets beschrieben.

CSS am Beispiel

Um einen ersten Eindruck von dem CSS-Funktionsbereich zu bekommen, werden wieder zunächst das bereits bekannte Beispieldokument und ein Stylesheet im CSS-Format betrachtet. Das Stylesheet findet sich im Anhang 7.c. Das HTML-Dokument, das das Stylesheet einbindet, findet sich auf der Web-Site dieses Kurses. Basis ist die mit CLASS-Attributen angereicherte HTML-Version des Memos, die bereits vorgestellt wurde. Der Anhang 7.d enthält das mit dem Microsoft-Internet-Explorer-Version 5 gemäß dem Stylesheet formatierte Dokument. Die Zeilen im HTML-Code, die das Stylesheet einbinden, lauten

```
<LINK REL="stylesheet" TYPE="text/css"  
      HREF="compProp.css">
```

Das Stylesheet gibt für die einzelnen HTML-Elemente die Rahmenbedingungen für die Formatierung an. Es werden Parameter für Schriften, Farben, Ausrichtungen und Abstände gesetzt. Im Rahmen dieser Vorgaben berechnen CSS-fähige Browser eine Darstellung eines HTML-Dokuments. An den beiden ähnlichen, aber nicht identischen Formatierungen unseres Beispieldokuments unter (dem mittlerweile nahezu historischen) Netscape-Navigator und (dem demnächst wahrscheinlich ebenfalls

historischen) Internet-Explorer lässt sich ersehen, dass die Vorgaben noch Gestaltungsspielraum lassen, z. B. bezüglich der Zeilenumbrüche oder der tatsächlich ausgewählten Schriften.

Ein CSS Steckbrief

CSS ist eine Sprache, in der sich Formatvorgaben oder Stylesheets für HTML-Dokumente formulieren lassen. CSS-Formatvorgaben steuern die Präsentation von HTML-Dokumenten durch Regeln, die bestimmte Parametersetzungen für bestimmte HTML-Elemente festzschreiben. Das Binden von Regeln an HTML-Elemente erfolgt über den Namen, die Attribute oder den strukturellen Kontext der HTML-Elemente.

Vielseitigkeit von CSS

Damit ist CSS im Prinzip anwendbar auf alle Dokumentensprachen, deren Dokumente Hierarchien von Strukturelementen und Text bilden und deren Strukturelemente über einen Namen und optional über Attribute verfügen. CSS-Stylesheets lassen sich auch auf XML-Dokumente anwenden. CSS-Strukturvorgaben können außerhalb von HTML-Dokumenten gehalten werden. CSS-fähige Browser präsentieren ein HTML-Dokument in dem durch das Stylesheet vorgegebenen Format. Damit wird durch HTML und CSS zusammen das Modell der strukturierten Dokumente umgesetzt.

Es kann vorkommen, dass CSS-Parameter für ein HTML-Element im Dokument unterspezifiziert oder überspezifiziert sind. Es können also auch gar keine Regeln und explizite Werte für die Parameter gesetzt bzw. mehrere und widersprüchliche Wertesetzungen definiert sein. Beide Fälle wurden bei CSS vorgesehen. Im Fall der Unterspezifizierung greift ein Vererbungs-Mechanismus, im Fall der Überspezifizierung die sogenannte Kaskade, die konkurrierende Quellen für Stylesheets (z. B. intern oder extern definierte Stylesheets) in einer Reihenfolge bringt – also kaskadiert – und zusätzlich mit Prioritäten arbeitet. Die Kaskade hat den Cascading Style Sheets ihren Namen gegeben. Bei CSS sind verschiedene Medientypen berücksichtigt, unter anderem visuelle etwa für die Ausgabe am Bildschirm und Drucker oder taktile Medientypen wie für die Braille-Schrift. Für jeden CSS-Parameter ist festgelegt für welche Medientypen er gültig ist. Jede Präsentation eines HTML-Dokuments gemäß einer CSS-Formatvorgabe erfolgt für ein spezifisches Medium; nur die für dieses Medium einschlägigen Parameter finden bei der Präsentation Berücksichtigung. Grundlegende visuelle Parameter betreffen Vordergrund- und Hintergrundfarben sowie Hintergrundbilder. Auch die verschiedenen Aspekte von Schriften, die Textformatierung sowie die Präsentation von Textblöcken gehören dazu.

Das W3C hat inzwischen zwei Versionen von CSS verabschiedet: CSS1 am 17. Dezember 1996 und CSS2 am 12. Mai 1998. Die dritte Version ist bereits seit 2000 in Arbeit. Das Designziel von CSS3 war und ist, unterschiedliche Aspekte von CSS in Module zu separieren und so das Verständnis zu erleichtern sowie gezielte lokale Erweiterungen ohne „Seiteneffekte“ zu ermöglichen. Aus diesem Grund hat man bei CSS3 keine einzelnen Versionen mehr verwendet. Es sind noch längst nicht alle Techniken implementiert und befindet sich in ständiger Weiterentwicklung. Aus-

fürdliche Informationen dazu kann man unter <http://www.w3.org/Style/CSS/current-work> finden.

Anbindung von Stylesheets an HTML-Dokumente

HTML4 eröffnet drei Möglichkeiten, CSS-Formatvorgaben mit einem HTML-Dokument zu verknüpfen. Als erste Möglichkeit kann ein als externe Datei vorliegendes Stylesheet über das HTML-Element LINK zugebunden werden; das Element LINK erscheint immer innerhalb des Elements HEAD in einem HTML-Dokument. Wie in Abschnitt 7.b bereits beispielhaft gezeigt, entspricht die Definition dem folgenden Muster:

```
<LINK REL="stylesheet" TYPE="text/css" HREF="«URL»">,
```

wobei «URL» die URL des Stylesheets repräsentiert. Das HTML-Element LINK dient allgemein dazu, eine Beziehung zwischen dem Quelldokument, in dem das Element enthalten ist, und externen Daten herzustellen. Das Attribut REL gibt dabei die Art der Beziehung wieder und das Attribut TYPE den MIME-Typ der externen Daten.

HTML ermöglicht es auch, ein CSS-Stylesheet intern in einem HTML-Dokument unterzubringen. Als Container dient ein HTML-Element STYLE in dem HTML-Element HEAD. Die Definition muss dem folgenden Muster entsprechen:

```
<STYLE TYPE="text/css"><!--«StyleSheet»--></STYLE> .
```

| Steuerdirektive Funktion | Funktionalität |
|--------------------------|--|
| import | importiert Stylesheet |
| media | definiert Medientyp für eine Gruppe von Regeln |
| page | setzt die Seitendimensionen bei Medientyp paged |
| charset | definiert das Kodierungsschema des Stylesheets |
| font-face | definiert Schriftarten |
| document | beschränkt die Stilregeln basierend auf der URL des Dokuments |
| keyframes | legt die einzelnen Wegepunkte einer CCS-Animationssequenz fest |

| | |
|------------------------|---|
| <code>namespace</code> | definiert XML-Namensräume |
| <code>supports</code> | zur Unterstützung bestimmter Webbrowser |

Tabelle 4-1 Steuerdirektiven in CSS

Ein internes Stylesheet sollte immer mit HTML-Mitteln auskommen, die das Element STYLE noch nicht kennen, das Stylesheet überlesen und nicht etwa im Klartext darstellen.

Als dritte Möglichkeit sieht der HTML-Standard für die meisten HTML-Elemente das Attribut STYLE vor, als dessen Wert CSS-Wertzuweisungen erscheinen können. Externe und mithilfe des Elements LINK zugebundene Stylesheets bieten die Option, Webdokumente nach dem Modell der strukturierten Dokumente zu gestalten. Interne Stylesheets können eventuell zugebundene externe Stylesheets überschreiben oder dokumentspezifisch ergänzen. Mithilfe des Attributs STYLE kann für ein einzelnes Element in einem HTML-Dokument eine individuelle Formatierung vorgegeben werden.

Die CSS-Syntax

Ein CSS-Stylesheet besteht aus einer Folge von Regeln. Jede Regel beginnt mit Selektoren, mit denen bestimmt wird, auf welche HTML-Elemente die Regel anwendbar ist. Selektoren sind durch Kommata getrennt. Unser Beispiel enthält pro Regel nur einen Selektor.

CSS-Parameter

Auf die Selektoren folgen, in geschweifte Klammern eingeschlossen, Zuweisungen, die angeben, wie die CSS-Parameter für die selektierten HTML-Elemente zu setzen sind. Die Parameter beschreiben das äußere Erscheinungsbild des HTML-Elements, etwa seine Farbe oder seine Schrift.

Die Parameterzuweisungen in einer CSS-Regel folgen dem aus C und Java bekannten Muster

```
«PName»: «PWert»
```

wobei eine Regel mehrere durch Semikolon getrennte Parameterzuweisungen enthalten kann. Hinter «PWert» in einer Parameterzuweisung kann noch eine Gewichtung durch das Schlüsselwort !important stehen, das in Zusammenhang mit der Kaskade ausgewertet wird. Ein CSS-Stylesheet kann Steuerdirektiven enthalten – sogenannte At-Regeln – die folgendem Muster entsprechen:

```
@«SName» «SIinhalt»
```

In Tabelle 4-1 werden die bisher definierten Steuerdirektiven zusammengefasst und somit auch die möglichen Werte für «SName». Das Format für den Inhalt «SI-

halt» einer Steuerdirektive hängt von ihrem Typ «SName» ab. Je nach Typ sind auch Einschränkungen zur Position einer Steuerdirektive zu beachten. Beispielsweise muss eine Charset-Direktive immer ganz am Anfang eines externen Stylesheets stehen, direkt gefolgt von den Importdirektiven.

Kommentare in CSS folgen dem Muster

```
/*«Kommentartext»*/ .
```

Hinweis: Kommentare dürfen nicht geschachtelt auftreten.

Mittlerweile gibt es auch CSSDoc, um die CSS-Kommentare zu standardisieren. Das Prinzip ist schon bekannt von solchen Lösungen wie Javadoc oder Scriptdoc, um daraus eine Dokumentation zu generieren. Der Kommentarabschnitt wird mit `/**` eingeleitet und mit `*/` beendet.

CSS Selektoren

Einfache CSS-Selektoren geben einen Namen für ein HTML-Element an. Im Beispiel sind u. a. die Selektoren `BODY`, `H1` und `IMG` von diesem Typ. Selektoren können sich auch auf die Werte von CLASS-Attributen im HTML-Dokument beziehen. Sie folgen dann dem Muster: `.«CWert»`. Das Beispiel enthält für diesen Typ die Fälle `.Autorin` und `.Briefkopf`. Schließlich kann sich ein Selektor auf den Zustand des Browsers beziehen.

CSS Selektoren

Dies ist der Fall bei den sogenannten Pseudoclass-Selektoren `A:link` und `A:visited`, die HTML-Anker A selektieren, wenn ihre Ziele vom aktuellen Browser aus noch nie bzw. wenigstens schon einmal besucht worden sind. Für das Styling der HTML-Formulare gibt es sogenannte Userinterface-Pseudoklassen für Formularlemente: `:enabled`, `:disabled` und `:checked`. Um auf Interaktionen des Anwenders zu reagieren stehen die Pseudoklassen: `:hover`, `:active` und `:focus` zur Verfügung. Eine weitere Pseudoklasse heißt `:target` und ist für Elemente mit Verweisziel gedacht.

Sehr komfortabel sind die Struktur-Pseudoklassen von CSS. Mit `:root` kann die Wurzel des Dokuments und mit `:empty` können leere Elemente angesprochen werden. Um Kind-Elemente auszuwählen, gibt es `:first-child`, `:last-child`, `:nth-child()`, `:nth-last-child()` und `:only-child`. Will man ganz bestimmte Kind-Elemente ansprechen, nutzt man die Struktur-Selektoren `:first-of-type`, `:last-of-type`, `:nth-of-type()`, `:nth-last-of-type()` und `:only-of-type`.

Mit der Sprach-Pseudoklasse `:lang()` werden Elemente selektiert, die mit dem Attribut lang versehen sind. Mit der Negations-Pseudoklasse können Elemente ausgewählt werden, mit denen ein Selektor nicht übereinstimmt.

Die folgenden tabellarischen Aufstellungen geben einen genauen Überblick über die Selektoren von CSS; dabei ist es nützlich, sich die hierarchische Struktur eines HTML-Dokuments als Baum vorzustellen.

| Selektor | Passt auf | Art des Selektors |
|-------------------|---|-----------------------------|
| * | alle Elemente | Universalselektor |
| element{...} | Element mit dem Namen element | Typselektor |
| #elementid | Element mit der ID elementid | ID-Selktor |
| .klassenname | Elemente mit der Klasse klassenname | Klassenselektor |
| [attr] | Elemente mit dem angegebenen Attribut | Attributselektor |
| [attr="value"] | Elemente, deren Attribute den angegebenen Wert haben. | Attributselektor |
| [attr~= "value"] | Elemente, deren Attribute unter anderen exakt den angegebenen Wert enthält | Attributselektor |
| [attr^= "value"] | Elemente, deren Attributwerte mit der Zeichenkette des angegebenen Wertes anfangen | Attributselektor - Teilwert |
| [attr\$= "value"] | Elemente, deren Attributwerte mit der Zeichenkette des angegebenen Wertes enden | Attributselektor - Teilwert |
| [attr*= "value"] | Elemente, bei denen die Zeichenkette des angegebenen Wertes im Attributwert vorkommt | Attributselektor - Teilwert |
| [attr = "value"] | Elemente, bei denen im Attribut der Wert am Anfang als eine durch Bindestrich (Minuszeichen) getrennte Zeichenfolge steht | Attributselektor |

Tabelle 4-2 Übersicht über einfache Selektoren

| Selektor | Passt auf |
|-------------------|---|
| :root | das Wurzelement |
| :empty | leere Elemente |
| :first-child | das erste Kindelement innerhalb eines Elements |
| :last-child | das letzte Kindelement innerhalb eines Elements |
| :nth-child | jedes n-te Element innerhalb eines Elternelements |
| :nth-last-child | jedes n-te Kindelement in einem Element, dabei werden die Kindelemente von hinten durchlaufen |
| :nth-of-type | jedes n-te Element vom gleichen HTML-Typ auf gleicher Ebene (Geschwisterelemente) |
| :nth-last-of-type | jedes n-te Element auf gleicher Ebene (Geschwisterelemente), dabei werden die Elemente von hinten durchlaufen |

| | |
|----------------|--|
| :first-of-type | das erste Element, welches das erste Kindelement eines bestimmten Typs ist |
| :last-of-type | das letzte Element des gleichen HTML-Elementtyps innerhalb eines Elternelements |
| :only-child | ein Element, das keine Geschwisterelemente hat und damit das einzige Kindelement im übergeordneten Element ist |
| :only-of-type | ein Element, wenn es das einzige Kind dieses Typs im Elternelement ist |

Tabelle 4-3 Übersicht über Struktur-Pseudoklassen

| Selektor | Bedeutung | Art des Selektors |
|---------------|---|--|
| :link | Hyperlink, der noch nicht angeklickt wurde | Link-Pseudoklasse (Verweise) |
| :visited | Hyperlink, der bereits angeklickt wurde | Link-Pseudoklasse (Verweise) |
| :hover | für Elemente, die mit dem Mauszeiger berührt werden | Pseudoklasse für Maus- und Tastaturinteraktionen |
| :focus | für Elemente, die den Fokus erhalten | Pseudoklasse für Maus- und Tastaturinteraktionen |
| :active | für Elemente, die aktuell angeklickt sind | Pseudoklasse für Maus- und Tastaturinteraktionen |
| :target | für Elemente, die Ziel eines Verweises sind | Zielpseudoklasse |
| :lang(de) | für Elemente mit einer bestimmten Sprachauszeichnung | Sprachpseudoklasse |
| :enabled | für Formularfelder, die editierbar sind | Userinterface-Pseudoklasse |
| :disabled | für Formularfelder, die nicht editierbar sind | Userinterface-Pseudoklasse |
| :checked | für aktivierte Checkboxen oder Radiobuttons | Userinterface-Pseudoklasse |
| :not(Element) | Für alle Elemente, außer das Element, welches in Klammern steht | Negationspseudoklasse |

Tabelle 4-4 Übersicht über dynamische und weitere Pseudoklasse

| Selektor | Bedeutung |
|----------------|---|
| ::first-letter | dass das erste Zeichen in einer Zeile wird angesprochen |
| ::first-line | die erste Zeile eines Absatzes wird angesprochen |
| ::before | Inhalt wird vor einem Element eingefügt und formatiert |
| ::after | Inhalt wird nach einem Element eingefügt und formatiert |

Tabelle 4-5 Übersicht über Pseudoelemente

| Kombinator | Bedeutung | Art des Selektors |
|------------|--|--------------------|
| E F | Element F wird nur dann angesprochen, wenn es Nachfahre eines E-Elements ist | Nachfahrenselektor |

| | ments ist | |
|-------|---|---------------------|
| E > F | Element F wird nur dann angesprochen, wenn es Kindelement eines E-Elements ist | Kindselektor |
| E + F | Element F wird nur dann angesprochen, wenn es im Elementbaum direkt auf ein E-Element folgt, also der direkte Nachbar ist | Nachbarselektor |
| E ~ F | alle F-Elemente werden angesprochen, die im Elementbaum in derselben Ebene auf ein E-Element folgen | Geschwisterselektor |

Tabelle 4-6 Übersicht über Kombinationsselektoren

Mit den oben vorgestellten ein- und zweistelligen Operatoren können beliebige ungeklammerte Ausdrücke gebildet werden. An jeder Stelle, an der ein primitiver Selektor «Name» stehen darf, kann auch der primitive Selektor * verwendet werden. Der Selektor * darf, wenn er mit einem einstelligen Operator verknüpft wird, auch entfallen. Der Selektor H1 < .quote:first-child selektiert also alle Elemente, die erstes Kind eines Elements H1 sind und das Attribut CLASS="quote" gesetzt haben.

Ein Selektor wird immer in Bezug auf ein HTML-Dokument ausgewertet, und zwar zu einer Menge von Dokumentenknoten oder Vorkommen von Elementen im Dokument. Ein einstelliger Operator bewirkt dabei eine Einschränkung der Ergebnismenge. Ein zweistelliger Operator berechnet zu jedem Dokumentknoten, den der linke Operand bezeichnet, alle Knoten mit passendem Namen, die zu diesem Knoten in der durch den Operator angegebenen Beziehung stehen.

Der Selektor DIV > P:firstChild EM wird also folgendermaßen ausgewertet: Der Selektor DIV bezeichnet alle Elemente des Ausgangsdokuments mit Namen DIV. Der Selektor DIV > P bezeichnet alle Elemente P des Ausgangsdokuments, deren direkt übergeordnetes Element im Schritt davor ausgewählt wurde. Der Selektor DIV > P:firstChild wählt aus den im Schritt davor selektierten Elementen diejenigen aus, die in der Dokumentenhierarchie keinen linken Nachbarn haben. Der Selektor DIV > P:firstChild EM schließlich selektiert alle Elemente EM des Dokuments, die in der Dokumentenhierarchie direkt oder indirekt einem im vorigen Schritt selektierten Element untergeordnet sind.

Die Auswertung von Selektoren erfolgt hier von links nach rechts. Für jedes selektierte Element existiert ein Pfad durch den Dokumentenbaum, der aus sogenannten Zeugen oder Belegen für die einzelnen Teilausdrücke des Selektors besteht. Die Teilausdrücke sind die Zwischenergebnisse der Berechnung. Das jeweils selektierte Element steht am Ende des Pfades. Die Idee der Pfadausdrücke wird außerdem in Zusammenhang mit XPath vertieft. Hier bleibt erst einmal festzuhalten, dass CSS-Pfade

nur von oben nach unten durch einen Dokumentenbaum führen können und dass sie nur Elemente, nicht aber Attribute oder inhaltlichen Text selektieren können. Beide Einschränkungen werden in XPath aufgehoben.

Der Sprachschatz von CSS

Der CSS-Funktionsumfang von CSS2.1 zu CSS3 hat sich schon mehr als nur verdoppelt, und es sind bei weitem noch nicht alle Techniken implementiert. Vor allem die Erweiterungen der Gestaltungsmöglichkeiten u. a. durch die Aufstockung der beeinflussbaren Eigenschaften sind so umfangreich, dass im Rahmen dieser Kurseinheit nur auf einige Teilbereiche und Funktionalitäten eingegangen werden kann.

Die CSS-Parameter für visuelle Medien lassen sich in die folgenden Gruppen einteilen, die jeweils unterschiedliche Aspekte der Formatierung betreffen:

Schrift-Parameter, die die grafische Darstellung von Text durch Glyphenbilder steuern, wie Schriftart, Gewicht oder Größe.

Farbe und **Hintergrund**-Parameter, die Vorder- und Hintergrundfarben sowie Auswahl und Positionierung von Hintergrundbildern steuern.

Text-Parameter, die die Aufeinanderfolge von Glyphen und den Zeilenumbruch steuern, wie Wortzwischenraum, Ausrichtung, Zeileneinzug oder Zeilenhöhe.

Boxdarstellung-Parameter, die die Abmessungen und Positionierung von Textbereichen und Rändern steuern, wie Höhe, Breite oder Rahmenbreite. Parameter, die das Clipping von Textbereichen und das Verhalten bei Überlappungen steuern.

Reihenfolge-Parameter, die Abweichungen in der Reihenfolge der Darstellung gegenüber der Texteingabe steuern, z. B. für Fließtexte oder für die Einteilung in separat formatierte Textflüsse.

Markierung-Parameter, die Aussehen und Positionierung von Markierungen steuern, die etwa Listeneinträge oder Änderungen kennzeichnen.

Seitenlayout-Parameter, die den Satzspiegel bestimmen.

Tabellensatz-Parameter, die die Formatierung von Tabellen steuern.

Generierung-Parameter, die die Generierung von Inhalten betreffen.

Die Werte der Parameter für visuelle Medien können die folgenden Datentypen haben:

- Aufzählungstypen
- Numerische Typen (ganze und gebrochene Zahlen)

- Länge
- Schriftname
- URL
- Zähler
- Farbe
- Inhalt

In einem CSS-Stylesheet erfolgen Wertzuweisungen an Parameter über Ausdrücke passenden Typs, die teilweise relativ zu anderen Parameterwerten evaluiert werden. Ein Ausdruck kann für jeden Parameter verwendet werden, nämlich das Schlüsselwort `inherit`, das ohne Anführungszeichen einzugeben ist. Der Ausdruck besagt, dass der Wert geerbt wird.

Aufzählungstypen: Aufzählungstypen bestehen aus Schlüsselwörtern, die als *Literale* wie das Schlüsselwort `inherit` ohne Anführungszeichen einzugeben sind. Operatoren gibt es für Aufzählungstypen nicht.

Numerische Typen: Literale für die beiden numerischen Typen sind ganze oder gebrochene Zahlen mit Vorzeichen in Dezimalnotation.

Länge: Literale für Längenangaben bestehen aus einem numerischen Wert, gefolgt von einer der folgenden Längeneinheiten:

- in steht für Inch oder Zoll wobei 1 Zoll 2,54 Zentimetern entspricht
- cm steht für Zentimeter
- mm bedeutet Millimeter
- pt heißt Point oder Punkt und 1 Punkt entspricht 1/72 Zoll
- pc bedeutet Pica und 1 Pica entspricht 12 Punkten

Ein Typ von Ausdrücken für Längenangaben besteht aus einem numerischen Wert, gefolgt von einer der folgenden relativen Längeneinheiten: em korrespondiert mit der Größe der aktuellen Schrift und entspricht ungefähr der Breite des Buchstabens „M“ in dieser Schrift, ex korrespondiert mit der Größe der aktuellen Schrift und entspricht ungefähr der Höhe des Buchstabens „x“ in dieser Schrift oder px heißt Pixel und ist eine auflösungs- und medienabhängige Größe, ein ganzzahliges Vielfaches des Durchmessers eines Bildpunkts für das Ausgabemedium, bei Bildschirmen in der Größenordnung von 0,28 Millimeter.

Ein weiterer Typ von Längenausdrücken besteht aus einer Prozentangabe. Prozentangaben werden durch einen numerischen Wert, gefolgt von einem Prozentzeichen angegeben. Sie berechnen sich in der Regel, abhängig vom Parameter, für den sie gesetzt sind, relativ zu dem entsprechenden Parameterwert des übergeordneten Elements.

Der Datentyp Länge hat noch einen speziellen Wert, nämlich `auto`, den jedoch nur die Parameter für die Höhe, die Breite und die Randausdehnungen einer Box an-

nehmen dürfen. Dieser Wert wird über das Schlüsselwort `auto` spezifiziert. Die tatsächliche Länge eines auf `auto` gesetzten Parameters wird dann nicht bei der Auswertung der Parameterspezifikation, sondern erst bei der Formatierung bestimmt. Es handelt sich also dann nicht, wie sonst bei allen Werten, um Eingabewerte für die Formatierung, sondern um Ausgabewerte.

Schriftnamen: Schriftnamen können als Schlüsselwörter ohne Anführungszeichen angegeben werden. Sie können jedoch auch in einfachen oder doppelten Anführungszeichen stehen und müssen dies sogar, wenn sie Leerzeichen enthalten. Generische Schriftnamen wie `serif` oder `sans-serif` müssen ohne Anführungszeichen stehen, wenn sie nicht als konkrete Namen interpretiert werden sollen. Mithilfe des Kommaoperators können Ausdrücke aus Schriftnamen-Literale gebildet werden. Eine solche Liste von Schriftnamen gibt eine Auswahl an Schriften in absteigender Präferenz an. Da es von der Rechnerplattform und vom Browser abhängt, welche Schriften in jedem Einzelfall zur Anzeige eines Dokuments zur Verfügung stehen, sollte jede Schriftnamenspezifikation als letztes Element in ihrer Liste einen der generischen Schriftnamen `serif`, `sans-serif` oder `monospace` vorsehen, da diese drei generischen Schriften von praktisch jeder Plattform unterstützt werden

URL: URLs werden in der Form `url(«quotedURL»)` gegeben, wobei der Ausdruck `«quotedURL»` eine optional in (einfache oder doppelte) Anführungszeichen eingeschlossene URL ist, in der CSS-Funktionszeichen mit dem Escape-Zeichen `\` geschützt sind.

Zähler: Zählernamen in CSS sind benutzerdefiniert. Sie werden als Schlüsselwörter ohne Anführungszeichen gegeben. Zähler werden über die Parameter `counter-increment` und `counter-reset` fortgezählt bzw. zurückgesetzt. Sie sind automatisch mit 0 initialisiert.

Farbe: Farben können in CSS über vordefinierte Namen oder über ihre Rot-, Grün- und Blauanteile im RGB-Farbraum angegeben werden. Ein RGB-Wert hat als eine mögliche Syntax `#«RHex»«GHex»«BHex»` mit Paaren von Hexadezimalziffern `#«XHex»`. Dabei ist nur bei Paaren aus identischen Hexadezimalziffern XX und Werten 0, 3, 6, 9, C und F für X garantiert, dass die Farben auf allen Browsern gleich dargestellt werden. Dies sind die sogenannten browsersicheren Farben. Neben konkreten vordefinierten Farbnamen wie `aqua` oder `gray` gibt es noch generische Farbnamen wie `button-text`, `button-highlight` oder `active-border`, deren tatsächliche Farbwerte plattformabhängig sind.

Inhalt: Ein Typ Inhalt-Literale sind Zeichenkettenliterale mit einfachen oder doppelten Anführungszeichen. Bestimmte Zeichen müssen dabei indirekt, über den Escape-Mechanismus `\ «DezASCIIICode»` eingegeben werden.

Ausdrücke vom Typ Inhalt werden durch einfaches Hintereinanderschalten von Inhaltsliteralen, Zählerreferenzen und URLs auf Bilder gebildet, was als Verkettung von Inhalten interpretiert wird. Eine Referenz auf einen Zähler hat dabei die Form `cou-`

ter («ZName») oder counter («ZName», «Format»); die durch URLs referenzierten Bilder werden wie Glyphenbilder im laufenden Text behandelt.

Die folgenden CSS-Regeln enthalten einige Beispiele für Parametersetzungen:

```
H1:before
content: "Chapter" counter(chapter) ".";
counter-increment: chapter;
counter-reset: section;
H2:before
content: counter(chapter) "." counter(section) " ";
counter-increment(section);
LI
list-style-image: url("http://www.xxx.com/redCircle.gif");
list-style-position: inside;
list-style-type: disc;
color: #ff0000;
```

Selbsttestaufgabe 2.7:

Arbeiten Sie weiter mit dem HTML-Dokument aus Selbsttestaufgabe 2.6.

Definieren Sie ein CSS-Stylesheet, das die Formatierung Ihrer ursprünglichen Vorlage aus einem Textsystem möglichst originalgetreu reproduziert.

Erkennen Sie an Ihrem Fallbeispiel Grenzen von CSS?

In diesem Abschnitt wurde zunächst eine systematische Darstellung des Sprachschatzes von CSS gegeben. Für die Entwicklung von CSS-Stylesheets werden jedoch detailliertere Informationen über die CSS-Parameter und ihre Semantik benötigt. Diese Informationen finden sich im Web. Alternativ sollen hier noch einige Bücher empfohlen werden. Eine sehr gute Darstellung findet sich in dem Buch von Meyer [M00]. Das ebenfalls umfassende Buch von Niederst [N99] nähert sich dem Thema CSS aus der Perspektive einer Grafikdesignerin. Eines der ersten Bücher überhaupt zu CSS war [LB97], geschrieben von zwei Mitgliedern der W3C-Arbeitsgruppe zu CSS, Lie und Bos [LB97]. Das Buch ist zwar technisch nicht mehr auf dem neuesten Stand, bietet aber durch seinen klaren Stil einen einfachen Einstieg und entwickelt sehr schön die auch heute noch gültigen Designziele von CSS.

Der CSS-Prozessor

CSS-Prozessor

Ein CSS-Prozessor liest zunächst einmal ein HTML-Dokument und erzeugt daraus eine interne Datenstruktur. Diese Datenstruktur setzt das Datenmodell für HTML um. CSS-Prozessoren sind prinzipiell auch auf andere Dokumentensprachen anwendbar als HTML, vorausgesetzt die Sprachen lassen sich mit dem Datenmodell von HTML modellieren. Wie bereits erwähnt, trifft das auf XML zu. Der CSS-Prozessor identifiziert dann den Medientyp für die zu berechnende Präsentation. Dazu zieht er Information aus der Umgebung heran, beispielsweise den Browsetyp.

Wie diese Information aussieht und in welcher Form sie an den CSS-Prozessor weitergereicht wird, ist nicht Gegenstand der CSS-Spezifikation.

Im nächsten Schritt werden per CSS-Prozessor alle dem Dokument zugeordneten Stylesheets und darin alle Regeln selektiert, die für den Medientyp einschlägig sind. Ausschlaggebend für diese Auswahl sind die Medientypen der Parameter, die in einer Regel gesetzt werden, aber auch explizite Markierungen von Regeln in den Stylesheets auf CSS-Ebene oder von ganzen Stylesheets auf HTML-Ebene.

Selektion

Konzeptionell gesehen wird anschließend jedes Element in der Datenstruktur des HTML-Dokuments mit allen für den Medientypen einschlägigen Parametern und den für das Element berechneten Werten annotiert. Die Werte ergeben sich aus den Regeln der Stylesheets sowie bei Unterspezifikation aus den Vererbungsregeln und bei Überspezifikation aus den Prioritätsbestimmungen, also der Kaskade. Diese Struktur ist Grundlage der Formatierung. Das Ergebnis einer Formatierung für visuelle Medien ist in der CSS-Spezifikation als eine Struktur aus Boxen definiert. Diese Formatierungsstruktur enthält Daten über die Ausdehnung und Platzierung von Boxen und die typografischen Attribute von Inhalten. Der CSS-Prozessor berechnet die Formatierungsstruktur aus der mit Parametern annotierten Datenstruktur für das HTML-Dokument. Dieser Vorgang heißt auch Formatieren. Als letzter Schritt erfolgt dann die eigentliche Präsentation des Dokuments, also die Übertragung der Formatierungsstruktur auf das Präsentationsmedium.

Formatierung und Ergebnis einer Struktur

Es ist gute Praxis im Bereich der Dokumentensysteme und ein weiteres Beispiel für das Informatikprinzip der sogenannten Späten Bindung (engl. *late binding*), dass der Formatierer eine geräteunabhängige Zwischenstruktur, die Formatierungsstruktur, erzeugt, anstatt direkt ein spezielles Ausgabemedium zu bedienen. Der typischerweise durch das Late Binding erzielte Effektivitätsgewinn bedeutet in diesem konkreten Fall, dass die von einem Formatierer berechneten Ergebnisse auf verschiedenen Ausgabegeräten präsentiert werden können. Es ist lediglich erforderlich, für jeden Typ von Ausgabegerät einen sogenannten Treiber zur Verfügung zu haben, der die Formatierungsstruktur für das Ausgabegerät übersetzt.

[Donald E. Knuth²](#) hat in den achtziger Jahren die Technik des geräteunabhängigen Formatierformats in seinem Textsystem TeX zum ersten Mal angewendet. Das Formatierformat für TeX heißt DVI und die Abkürzung steht für device independent. Da LaTeX als Makropaket zu TeX realisiert ist, ist DVI auch das Formatierformat für LaTeX. Es gibt DVI-Treiber für alle bekannten Ausgabemedien. Am weitesten verbreitet ist der PostScript-Treiber dvips und der X-Treiber xdvi.

Prozessmodell in CSS

Ein CSS-Prozessor muss die einzelnen Schritte des Prozessmodells nicht wirklich implementieren. Er muss nur zu einer korrekten Präsentation kommen. Das Prozess-

² <http://www-cs-faculty.stanford.edu/~uno/>

modell in CSS dient hauptsächlich als Erklärungsmodell dafür, wie ein CSS-Stylesheet zu interpretieren ist. Deshalb gibt es auch keine verbindliche Definition für ein Formatierformat für CSS, das etwa mit DVI vergleichbar ist.

Zuweisung von Parameterwerten in CSS

Mit dem Prozessmodell von CSS ist vorgesehen, dass jedes Element im HTML-Dokument mit dem vollen Satz an für das aktuelle Medium einschlägigen CSS-Parametern annotiert wird. Die Parametersetzungen erfolgen jeweils gemäß der für ein Element gültigen Regeln. Diese entstammen den für das Dokument einschlägigen Stylesheets. In Vorbereitung der Parametersetzungen müssen also zunächst die einschlägigen Stylesheets identifiziert und ihre Regeln in eine Reihenfolge gebracht werden. Dieser Vorgang wird auch *Kaskadierung* genannt.

Es sind die folgenden Stylesheets zu berücksichtigen: Die von Autor oder Autorin im HTML-Dokument vorgegebenen Author-Stylesheets, die lokal vorgegebenen User-Stylesheets und die im Browser vorgegebenen Browser-Stylesheets für die Grundeinstellungen.

Für jeden der drei Stylesheet-Typen werden die Importsteuerdirektiven rekursiv ausgeführt; die referenzierten Stylesheets werden jeweils am Anfang des übergeordneten Stylesheets in der Reihenfolge der Importdirektiven inkludiert. Wie im weiteren Verlauf erläutert wird, erhalten die importierten Regeln damit eine niedrigere Priorität als die Regeln des importierenden Stylesheets. Ein im HTML-Dokument mit dem Element STYLE definiertes Stylesheet wird hinten an das Author-Stylesheet angehängt, gefolgt von Regeln für Attribute vom Typ STYLE im Dokument; diese Regeln erhalten damit höhere Priorität.

In den resultierenden drei Stylesheets werden dann jeweils die komplexen Regeln aufgelöst, sodass jede Regel nur noch für eine Kombination von Element und Parameter den Wert festlegt; dabei ist jede Auflösung akzeptabel, die die Reihenfolge der Parameterzuweisungen pro Selektor beibehält. Schließlich werden aus jedem der drei Stylesheets X die mit !important markierten Regeln in ein Stylesheet Important X extrahiert.

Innerhalb eines jeden der so entstandenen sechs Stylesheets werden nun die Regeln nach Spezifität ihres Selektors aufsteigend geordnet, wobei für Regeln mit gleicher Spezifität die ursprüngliche Reihenfolge erhalten bleiben muss. Die Spezifität von Selektoren errechnet sich als numerischer Wert aus der Zahl der Attribute und Elemente in einem Selektor.

Die sechs so sortierten Stylesheets werden nun in die folgende Sequenz kaskadiert:

- Browser-Stylesheet
- User-Stylesheet
- Author-Stylesheet
- Important Browser-Stylesheet

- Important Author-Stylesheet
- Important User-Stylesheet

Während also im Normalfall Regeln aus dem Author-Stylesheet lokal definierte Regeln überschreiben, ist es bei den mit !important markierten Regeln genau umgekehrt. Intention ist es, dass Nutzer mit speziellen Anforderungen an die Präsentation, was etwa die Schriftgröße oder das Farbschema angeht, ihre eigenen Präferenzen einstellen können.

Zusammenfassend lässt sich also sagen, dass die Kaskadenregeln die Regeln der verschiedenen für ein HTML-Dokument relevanten Stylesheets in eine Ordnung bringen. Erstes Ordnungskriterium ist dabei die Herkunft der Regeln, kombiniert mit ihrem Gewicht. Zweites Kriterium ist die Spezifität ihrer Selektoren. Und letztes Kriterium ist die Reihenfolge, in der die Regeln spezifiziert sind.

Soll nun für ein konkretes HTML-Element ein bestimmter CSS-Parameter gesetzt werden, so können zwei Ausnahmefälle eintreten:

- Der Parameter kann für das Element unterspezifiziert sein; d. h. keine Regel in den zum Dokument gehörigen Stylesheets definiert einen Wert für diesen Parameter und dieses Element.
- Der Parameter kann für das Element überspezifiziert sein; d. h. der Parameter ist für dieses Element in den zum Dokument gehörigen Stylesheets mehrfach und mit unterschiedlichen Werten definiert.

Im ersten Fall tritt, abhängig von der Erbberechtigung des Parameters, die Vererbungsregel in Kraft: Parameter mit Erbberechtigung erben ihren Wert von dem übergeordneten Element in der Dokumentenhierarchie; Parameter ohne Erbberechtigung erhalten einen in der CSS-Spezifikation festgelegten Grundeinstellungswert. Beispielsweise hat der Parameter WIDTH keine Erbberechtigung, der Parameter FONT-SIZE jedoch schon.

Vererbt werden Parameterwerte und nicht die Formeln zu ihrer Berechnung. Ist also etwa die Schriftgröße FONT-SIZE für das Element BLOCKQUOTE auf 80 % gesetzt in einer Umgebung mit einer 10-Punkt-Schriftgröße, so wird der Text innerhalb des Elements in einer 8-Punkt-Schrift gesetzt. Enthält BLOCKQUOTE ein Element em, für das die Schriftgröße nicht explizit gesetzt ist, so erbt em die Schriftgröße 8 Punkt und nicht etwa 80 % davon.

Ist ein Parameter für ein Element überspezifiziert, so wird die hinten stehende Regel aus den kaskadierenden Stylesheets angewendet, die auf das Element passt.

Parameter bei CSS

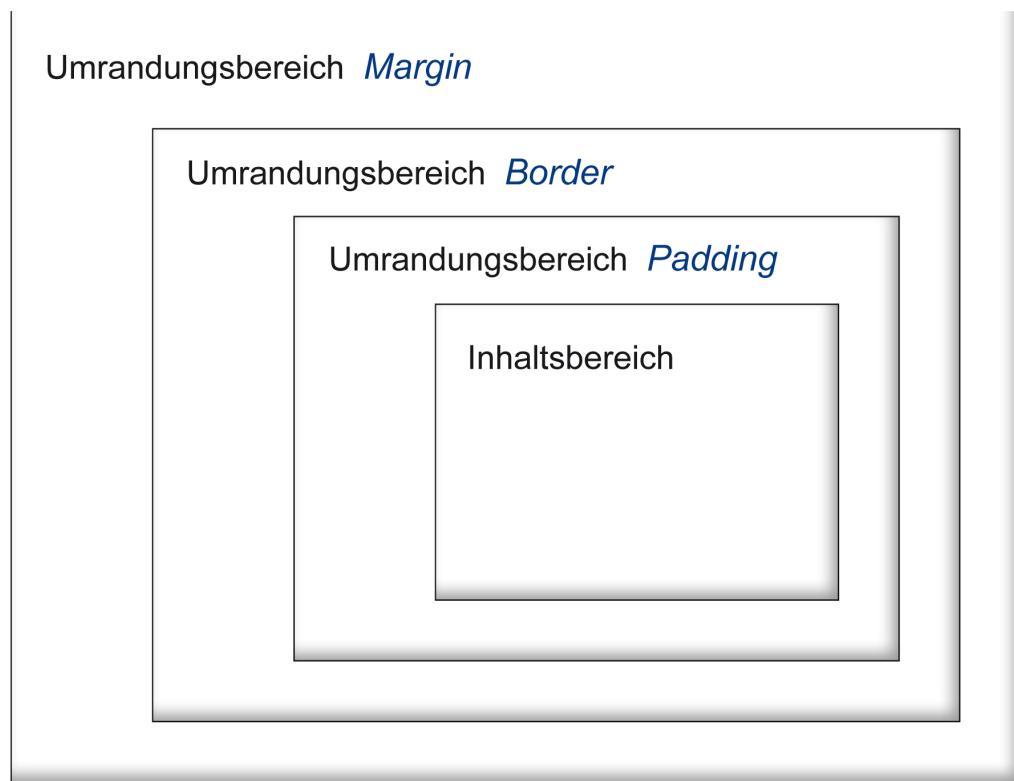
Als Ergebnis der Formatierung für ein visuelles Medium berechnet ein CSS-Prozessor eine Struktur von sogenannten Boxen. Eine Box ist, wie in Abb. 2.5

Boxen in CSS

dargestellt, ein rechteckiger Inhaltsbereich mit drei Umrandungsbereichen von innen nach außen: Padding, Border und Margin.

Dem Inhaltsbereich einer Box sind entweder weitere Boxen oder reiner Text zugeordnet. Der Inhalt einer Box darf dabei rein geometrisch gesehen über den Rand des Inhaltsbereichs hinausragen. Der Inhaltsbereich einer Box stellt für seinen Inhalt lediglich ein Koordinatensystem zur Verfügung, relativ zu dem sich der Inhalt positionieren kann.

Jede Box in der Boxstruktur eines Dokuments ist von einem HTML-Element oder einem Pseudoelement des Dokuments generiert worden. Die CSS-Parameter des erzeugenden Elements bestimmen einige Aspekte der Erscheinungsform einer Box. Die BACKGROUND-Parameter des erzeugenden Elements bestimmen die Hintergrundfarbe und das Hintergrundbild für den Inhaltsbereich und den Bereich Padding der generierten Box. Die BORDER-Parameter des erzeugenden Elements bestimmen die Farbe und das Muster des Rand-Bereichs der generierten Box.



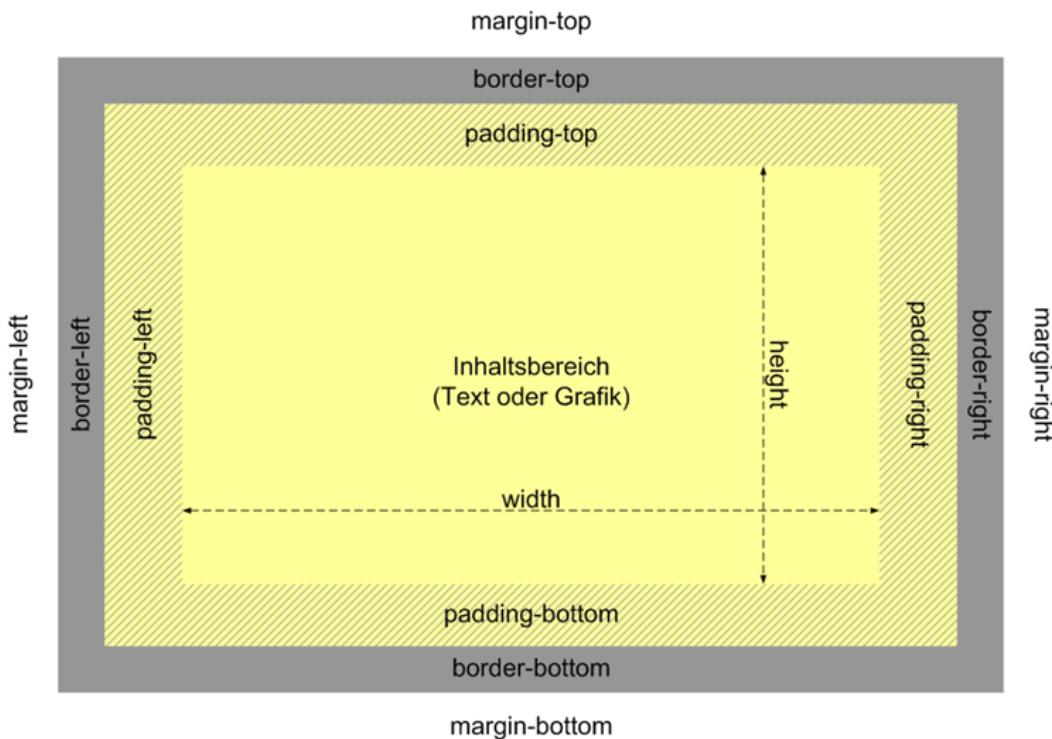
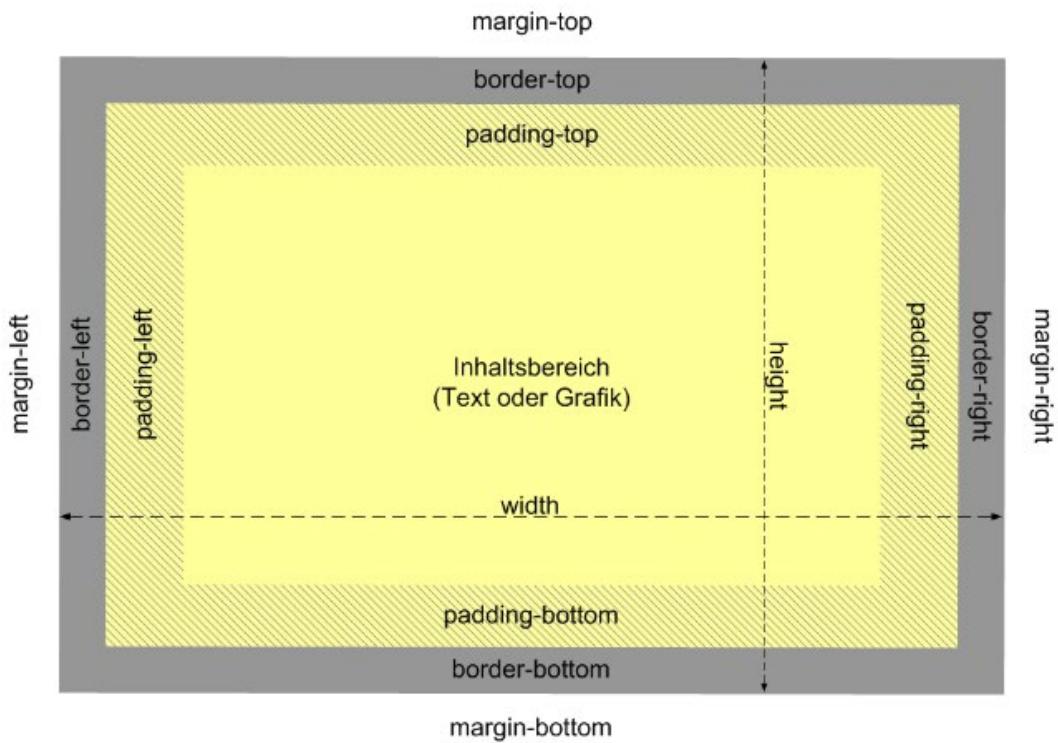


Abbildung 4.1 Das klassische Boxmodell für CSS

Die grafische Gestalt des Bereichs Margin ist über das generierende Element nicht beeinflussbar; der Bereich ist immer transparent. Die Ausdehnungen der jeweils vier Segmente der drei Umrandungsbereiche sowie des Inhaltsbereichs werden ebenfalls über Parameter des erzeugenden Elements vorgegeben. Die Ausdehnungen des Inhaltsbereichs und der vier Segmente des Randbereichs Margin können mit dem Wert auto spezifiziert sein; die tatsächlichen Werte werden dann im Laufe der Formatierung vom CSS-Prozessor bestimmt.

Das neue Boxmodell von CSS

Das klassische Box-Modell wird dann problematisch, wenn innerhalb einer Box für die Angaben von width, padding, border oder margin verschiedene Einheiten verwendet werden. In CSS3 wurde das neue Box-Modell eingeführt. In diesem Modell wird die Breite width und die Höhe height nicht mehr nur für den Inhaltsbereich festgelegt, sondern der Innenabstand (padding) und der Rahmen (border) gleich mitberücksichtigt werden. Das Modell heißt border-box, weil die Breite der Box von border bis border gemessen wird. Die CSS-Eigenschaften width und height gelten von border-left bis border-right bzw. border-top bis border-bottom. Wird eine Breitenangabe mit width gemacht, dann haben padding und border keinen Einfluss mehr auf die gemachte Angabe und werden von dieser Breite abgezogen.



CSS3-Box-Modell für Block-Elemente – { box-sizing: border-box } – <http://little-boxes.de/>

Abbildung 4.2 Das neue Boxmodell für CSS

Mit der Eigenschaft `box-sizing` kann man das neue Box-Modell anwenden. Das CSS sieht für `box-sizing` folgende Werte vor:

- `content-box`: klassisches Box-Modell,
- `padding-box`: neues Box-Modell – Innenabstände verbreitern ein Element nicht mehr,
- `border-box`: neues Box-Modell – Rahmen verbreitern ein Element nicht mehr,
- `inherit`: Wert von Elternelement übernehmen.

Alle modernen Webbrowser können mit dem neuen Box-Modell umgehen.

Das Formatiermodell für CSS

Das CSS-Prozessmodell besagt, dass bei visuellen Medien der Inhalt eines HTML-Elements in eine Box fließt, die von dem Element erzeugt wird. Neben dieser Hauptbox, in die sein Inhalt fließt, kann ein HTML-Element noch weitere Boxen generieren. Beispielsweise generiert ein Element `LI` zusätzlich zu einer Hauptbox, die seinen Inhalt aufnimmt, noch eine Nebenbox für die Markierung des Aufzählungspunkts wie etwa mit einem Spiegelstrich.

CSS unterscheidet zwei Typen von Boxen: Inlineboxen und Blockboxen. Inlineboxen können an einer Inlineformatierung teilnehmen. Sie können mit anderen Inlineboxen zusammen horizontal in Zeilen angeordnet und an Zeilengrenzen in mehrere Inline-

boxen umgebrochen werden. Blockboxen können an einer Blockformatierung teilnehmen. Sie können mit anderen Blockboxen zusammen vertikal angeordnet und bei einem Medientyp paged an Seitengrenzen in mehrere Blockboxen umgebrochen werden.

Der Typ einer Box ergibt sich aus dem CSS-Parameter DISPLAY des generierenden Elements, wie in der folgenden Aufstellung angegeben:

block Ein Element mit Parameter DISPLAY="block" generiert eine Blockbox als seine Hauptbox und keine weiteren Boxen.

inline Ein Element mit Parameter DISPLAY="inline" generiert eine Inlinebox als seine Hauptbox und keine weiteren Boxen.

list-item Ein Element mit Parameter DISPLAY="list-item" generiert zwei Boxen, nämlich eine Blockbox als seine Hauptbox und eine Inlinebox, die die grafische oder alphabetische Markierung des Elements aufnimmt. Weitere Attribute zum Listenstil bestimmen die Art und die Position der Markierung.

marker CSS-Regeln mit Selektoren der Form «Selektor»:before beziehungsweise «Selektor»:after können vor und hinter dem mit «Selektor» ausgewählten Element anonyme, sogenannte Pseudoelemente einfügen und für sie das Attribut DISPLAY = "marker" setzen. Das Pseudoelement generiert dann eine Inlinebox, die seinen Inhalt aufnimmt und die in Relation zur Hauptbox des mit dem Pseudoelement assoziierten „echten“ Elements positioniert wird. Nur Pseudoelemente können das Attribut DISPLAY = "marker" verwenden; ein echtes Element, das DISPLAY = "marker" gesetzt hat, wird als Inlineelement behandelt.

none Ein Element mit DISPLAY = "none" generiert keine Box; es wird so getan, als wären das Element selbst und sein Inhalt nicht vorhanden.

run-in Ein Element mit Attribut DISPLAY="run-in" generiert, je nach Kontext, eine Blockbox oder eine Inlinebox. Erscheint im Fluss (s. u.) der Boxen direkt nach der aus dem Run-in-Element zu generierenden Box eine (noch weiteren, hier vernachlässigten Anforderungen genügende) Blockbox, so generiert das Run-in-Element eine Inlinebox, die als erstes Element in die Blockbox aufgenommen wird. Im anderen Fall generiert das Run-in-Element eine Blockbox.

compact Ein Element mit Attribut DISPLAY="compact" generiert, je nach Kontext, eine Blockbox oder eine Inlinebox. Erscheint im Fluss (s. u.) der Boxen direkt nach der aus dem Compact-Element zu generierenden Box eine (noch weiteren, hier vernachlässigten Anforderungen genügende) Blockbox, so generiert das Compact-Element versuchsweise eine Inlinebox und berechnet deren Breite. Passt die generierte Inlinebox in den Rand der Blockbox, so wird sie

dort positioniert. Im anderen Fall generiert das Compact-Element eine Blockbox.

Der Parameter `DISPLAY` kann auch den Wert `table` und weitere, mit Tabellen zusammenhängende Werte enthalten, die Elementen eine Rolle bei der Tabellenformatierung zuweisen. Letztendlich generiert ein Element mit Parametersetzung `DISPLAY="table"` eine Blockbox. Die interne Struktur dieser Box bestimmt sich aus den speziellen Regeln der Tabellenformatierung, die wir hier außer Acht lassen.

Zusammenfassend lässt sich sagen, dass ein CSS-Prozessor als ersten Schritt der Formatierung eine Struktur aus Boxen generiert, wobei Boxen Text und weitere Boxen enthalten können. Elemente und Pseudoelemente, für die nicht `DISPLAY="none"` gesetzt ist, generieren eine Hauptbox, die ihre Inhalte aufnimmt. Daraus resultiert eine hierarchische Struktur aus Hauptboxen und Text, die der Datenstruktur des HTML-Dokuments entspricht, modulo der unterdrückten Elemente mit der Setzung `DISPLAY="none"`. Zusätzlich zur Hauptbox können Elemente Nebenboxen generieren, die zu ihrer Hauptbox assoziiert sind und relativ zu ihr positioniert werden. Einen Sonderfall bilden Pseudoelemente, für die der Parameter `DISPLAY="marker"` gesetzt ist und die damit keine Hauptbox generieren, sondern eine zu einer anderen Hauptbox assoziierte Nebenbox. Einen weiteren Sonderfall bilden Elemente, für die das Attribut `FLOAT` gesetzt ist. Also etwa `FLOAT="left"` oder `FLOAT="right"`. Die von einem solchen Element generierte Box wird aus der Hierarchie der Hauptboxen herausgenommen und zu der übergeordneten Blockbox assoziiert.

Die generierten Boxen sind entweder Blockboxen oder Inlineboxen; die Entscheidung über den Boxentyp bleibt in einigen Fällen noch offen, bis die Formatierung weiter fortgeschritten ist. Jede Box ist mit einem Element oder Pseudoelement assoziiert, das sie generiert und ihren Inhalt bestimmt hat.

Im nächsten Schritt wird die bisher erzeugte Boxenstruktur bereinigt. Zunächst einmal soll erreicht werden, dass jede Box entweder nur Text oder nur Inlineboxen oder nur Blockboxen enthält – also keine gemischte Struktur. Dazu werden maximale Sequenzen von Text und Inlineboxen in unerwünschten Mischkontexten mit Blockboxen in anonyme mit keinem Element oder Pseudoelement assoziierte Blockboxen eingeschlossen. Anschließend werden maximale Textstücke in unerwünschten Mischkontexten mit Inlineboxen in anonyme Inlineboxen eingeschlossen.

Blockboxen innerhalb von Inlineboxen

In der CSS-Spezifikation gibt es keine Aussage dazu, wie CSS-Prozessoren Blockboxen innerhalb von Inlineboxen behandeln sollen. Die Implementierungen in Browsern scheinen sich nach folgender Regel zu verhalten: Eine Blockbox innerhalb einer Inlinebox bricht die Inlinebox in drei auf gleicher Hierarchiestufe stehende Boxen auf, nämlich eine Inlinebox mit den Inhalten vor der Blockbox, die Blockbox selbst und eine Inlinebox mit den Inhalten nach der Blockbox. Dies zieht dann weitere Bereinigungen nach sich. Die Abbildung 4.3, in der Rechtecke mit

durchgezogener Umrandung für Blockboxen und Rechtecke mit punktierter Umrandung für Inlineboxen stehen, illustriert die Schritte dieses Schemas.

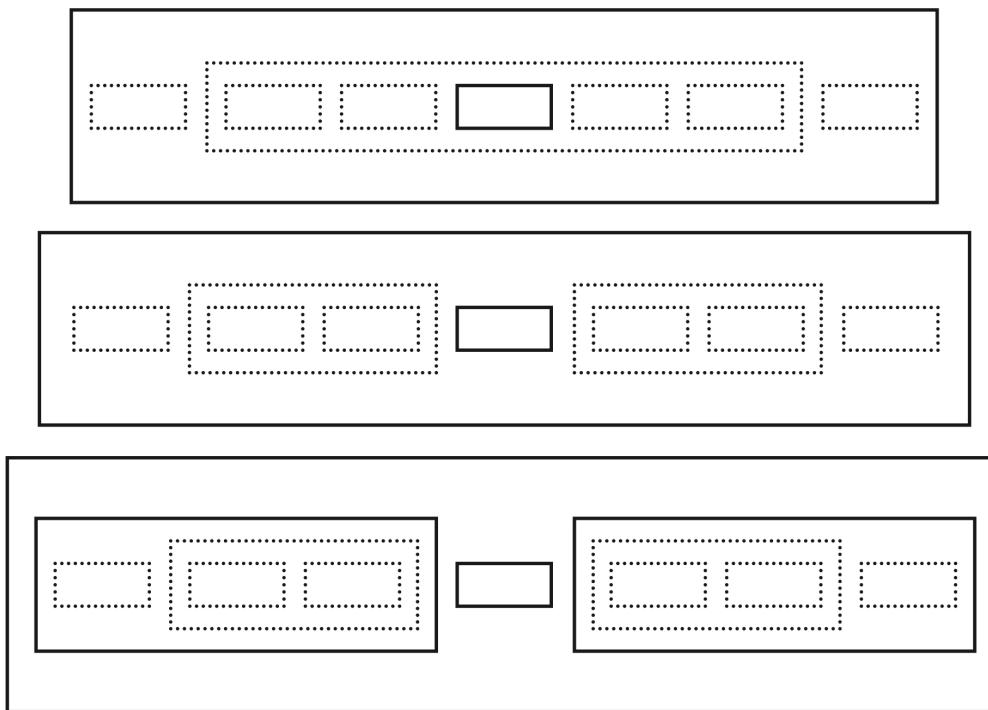


Abbildung 4.3 Behandlung von Blockboxen in Inlineboxen in Browsern

Es kann also für die weitere Formatierung von einer primär hierarchisch organisierten Struktur aus Boxen und Text ausgegangen werden. Zu den Boxen der Haupthierarchie können weitere, in der Regel flach strukturierte Boxen assoziiert sein. Blockboxen in dieser Boxstruktur enthalten entweder nur weitere Blockboxen oder nur Inlineboxen bzw. nur Text. Inlineboxen enthalten nur weitere Inlineboxen oder nur Text.

Jede Box ist mit einem vollständigen Satz von CSS-Parametern versehen, die sie von ihrem generierenden Element geerbt hat. Anonyme Boxen erhalten dabei die Werte der übergeordneten Boxen bzw. die Default-Werte entsprechend des Vererbungstyps. Von den Übergabe-Parametern ist DISPLAY bereits ausgewertet und somit im weiteren Verlauf zu ignorieren.

In der Dokumentenverarbeitung wird eine Hierarchie von Boxen, wie sie ein CSS-Prozessor erzeugt, Fluss (engl. *flow*) genannt. Ein Fluss bestimmt die Reihenfolge, in der die in ihm enthaltenen Objekte formatiert und positioniert werden. Bei CSS ist vorgesehen, dass Teilhierarchien von Boxen aus dem Fluss, in den sie von ihren generierenden Elementen platziert worden sind, herausgenommen werden und einen eigenständigen Fluss bilden. Der neue Fluss wird dann unabhängig von dem Kontext, dem er entnommen wurde, formatiert und positioniert.

Flüsse und Boxen

Durch den CSS-Parameter POSITION wird bestimmt, ob eine Box mitsamt untergeordneten Boxen und Text in ihrem Fluss verbleibt oder in neuem Kontext einen neuen Fluss begründet. Das Setzen von `POSITION = "static"` bzw. `"relative"` bestimmt, dass die Box in ihrem Fluss verbleibt. Per `POSITION="absolute"` bzw. `"fixed"` wird ein neuer Fluss begründet.

Eine Box mit Attribut `POSITION="relativ"` wird bezugnehmend zu ihrer normalen Position im aktuellen Fluss verschoben. Dadurch wird die Position anderer Boxen im Fluss nicht beeinflusst. Diese Funktionalität lässt sich etwa für Hoch- und Tiefstellungen verwenden.

Eine Box mit `POSITION="absolut"` wird aus ihrem aktuellen Fluss herausgenommen und formatiert. Die Box wird dann relativ zu ihrer *Referenzbox* positioniert. Dies ist eine Box, die ihr im ursprünglichen Fluss übergeordnet war. Referenzboxen für absolut positionierte Boxen sind sogenannte positionierte Boxen, also Boxen mit Attribut `POSITION = "relativ"`, `POSITION="absolut"` oder `POSITION="fixed"`.

Eine Box mit `POSITION="fixed"` wird genauso behandelt wie eine mit der Setzung `POSITION = "absolut"`, nur dass die Positionierung relativ zu dem Fensterausschnitt des Browsers, dem sogenannten Viewport auf das Dokument, erfolgt.

Nachdem ein CSS-Prozessor aus den Elementen des Dokuments eine Boxstruktur generiert, bereinigt und in Flüsse aufgeteilt hat, können nun die Flüsse unabhängig voneinander formatiert und anschließend die Ergebnisse relativ zueinander positioniert werden. CSS unterscheidet Blockformatierung für Sequenzen aus Blockboxen und Inlineformatierung für Sequenzen aus Inlineboxen bzw. für Text. Die Formatierung erfolgt rekursiv entlang der Haupthierarchie der Boxenstruktur.

Die Blockformatierung veranlasst für eine Sequenz von Blockboxen zunächst die Formatierung jeder einzelnen Box in der Sequenz. Als Ergebnis stehen für jede Box die Ausdehnungen der Inhalts- und Umrandungsbereiche gemäß dem Boxmodell fest. Die Blockformatierung ordnet dann diese Boxen untereinander an, wobei die Umrandungsbereiche Margin von zwei aufeinanderfolgenden Boxen zu deren Maximum kollabieren. Das Ergebnis bildet den Inhalt der übergeordneten Box und bestimmt, wenn die Höhen- oder Tiefenparameter der übergeordneten Box auf „auto“ gesetzt sind, die Höhe oder Tiefe ihres Inhaltsbereichs.

Die Inlineformatierung formatiert Sequenzen von Inlineboxen und deren untergeordnete Inlineboxen oder Textbereiche in Zeilen, die dann als Blockboxen wieder an einer Blockformatierung teilnehmen. Zeilengrenzen können dabei Inlineboxen auf-splitten; die Umrandungsbereiche an den Schnittstellen haben dabei alle eine Nullausdehnung.

Sowohl Block- als auch Inlineformatierung müssen zu den Hauptboxen assoziierte Boxen, insbesondere auch Floats, berücksichtigen. Die weitere Formatierung der

einzelnen Flüsse richtet sich nach den entsprechenden CSS-Parametern. Als Ergebnis der Formatierung berechnet ein CSS-Prozessor für alle primären und assoziierten Boxen eines Flusses Ausdehnungen und Positionen relativ zu den Referenzboxen. Es werden Farben, Hintergrundbilder und Randgestaltung für Boxen berechnet. Weiter werden die typografischen Attribute und die Positionierung für die Textzeichen bestimmt. Die Inhalte, die über den Inhaltsbereich ihrer Box hinausragen, werden ebenso verarbeitet wie auch Boxen über Zeilen- und Seitengrenzen nach Bedarf gesplittet werden. Das Ergebnis dieser Formatierung ist eine Formatierstruktur aus Boxen und Text, die direkt auf das Ausgabemedium übertragen werden kann.

Selbsttestaufgabe 2.8:

Ein HTML-Dokument repräsentiere Aufgaben mit folgendem Markup:

```
<DIV CLASS="Aufgabe">
<P>«Der Text der Aufgabe»</P>
</DIV>
<DIV CLASS="Aufgabe">
<P>«Der etwas längere,
über mehrere Zeilen gehende
Text einer weiteren Aufgabe»
</P>
</DIV>
```

Entwerfen Sie eine CSS-Regel mit folgender Wirkung: Der erste Absatz einer Aufgabe enthält vor dem eigentlichen Aufgabentext in Fettschrift das Schlüsselwort „Aufgabe“, gefolgt von der Nummer der Aufgabe, also:

Aufgabe 1«Der Text der Aufgabe»

Aufgabe 2«Der etwas längere, über mehrere Zeilen gehende Text einer weiteren Aufgabe»

Selbsttestaufgabe 2.9:

Entwerfen Sie eine CSS-Regel, die H2-Elemente fett setzt und in den darunter liegenden Text/Absatz laufen lässt.

Auf den Internet-Seiten des W3C befinden sich aktuelle Informationen zur Gestaltungsmöglichkeit mit CSS. Hier folgt nun ein zusammenfassender Überblick.

Medientypen

CSS unterstützt eine Reihe von Medientypen, die auf die unterschiedlichsten Präsentationsmedien oder Gerätetypen zugeschnitten sind. Die Medientypen sind jeweils in Gruppen klassifiziert (<http://www.w3.org/TR/CSS2/media.html>).

CSS-Werkzeuge

Jeder Webbrowser enthält mittlerweile Entwickler-Tools, die auch in Verbindung mit CSS außerordentlich hilfreich sind. Man kann mit diesen Tools auch gleich Änderun-

gen am CSS vornehmen und die Auswirkungen sofort sehen. Neben den Entwickler-Tools der Browser gibt es viele weitere Programme, die noch wesentlich mehr Möglichkeiten bieten.

Um CSS-Dateien komfortabel und valide zu erstellen, gibt es verschiedene Möglichkeiten. Günstig ist es, sich einen geeigneten Editor zu suchen. Die Konformität mit dem aktuellen Standard zu CSS kann auf den Webseiten des W3C entsprechend nachgeschaut werden. Wichtig ist es, dass die CSS-Seiten auch valide sind. Einige HTML-Editoren stellen eine WYSIWYG-Oberfläche zur Verfügung und generieren daraus HTML- und CSS-Dateien. Generelles Problem bei diesen Werkzeugen ist, dass sie in der Regel nicht die volle Funktionalität von CSS nutzen. Professionelle Software, wie sie etwa in der [Adobe-Creative-Suite](#)³ enthalten ist, bietet da zwar eine hinreichend gute Konformität, ist aber in der Regel nicht kostenfrei. Linux-Besitzer könnten sich die Open Source Varianten unter <http://cssed.sourceforge.net/> einmal anschauen. Das W3C stellt unter <http://jigsaw.w3.org/css-validator/> einen Validierer für CSS zur Verfügung. Der Validierer arbeitet als Service über das Web, kann aber auch als Java-Klasse implementiert werden. Wer Aufschluss zu den Selektoren braucht, bekommt hier <http://www.w3.org/TR/CSS2/selector.html> mehr Informationen dazu.

Praxis-Tipps:

- Für den Entwurf eines CSS-Stylesheets empfiehlt es sich, von einer Skizze zu arbeiten, in der Ausdehnungen, Farben und Schriften für die einzelnen HTML-Elemente festgehalten sind. Das erleichtert die Orientierung bei der Codierung in CSS.
- CSS3 befindet sich in ständiger Entwicklung und nicht jeder Browser beherrscht alle CSS-Anweisungen
- Tests für unterschiedliche Geräte und Bildschirmgrößen sind unumgänglich

Weiterführende Links

Die folgenden Links weisen zu den relevanten W3C-Spezifikationen bezüglich CSS und HTML:

- <http://www.w3.org/Style/CSS/current-work> – CSS-Spezifikationen
- <http://www.css3.info/selectors-test> – Selektoren-Implementierung im verwendeten Webbrowser
- <http://userstyles.org> – fertige Benutzerstylesheets für Google, Facebook etc.
- <http://www.w3.org/TR/css3-values/> – CSS-Wertangaben

³ <http://www.adobe.com/de/products/creativesuite/design.html>

- <http://caniuse.com> – Informationen zu Webbrowsern und deren Unterstützung von HTML5- und CSS3-Features
- <http://css3test.com> – CSS3-Webbrowsers-test – diese Adresse im aktuell verwen-deten Webbrowser eingeben

5 JavaScript

Vor ein paar Jahren hätte es noch gereicht, in dieser Kurseinheit HTML und CSS zu behandeln. Viele Neuerungen von HTML5 werden jedoch über JavaScript angesprochen, und wenn man sich ernsthaft mit HTML5 und CSS in der Praxis befasst, kommt man um JavaScript nicht mehr herum.

So ist es nicht verwunderlich, dass hier kurz auf JavaScript als Webprogrammiersprache eingegangen werden muss. Der Umfang von JavaScript und den neuen JavaScript-APIs ist so immens, dass hier lediglich eine rudimentäre Einführung gegeben werden kann, wobei vor allem die Neuerungen von HTML5 mit den JavaScript-APIs und DOM-Manipulationen im Focus stehen. Es sei angemerkt, dass es neben HTML, CSS und JavaScript viele weitere Technologien wie jQuery, Ajax etc. gibt, die darauf aufsetzen und mit denen man in der Praxis konfrontiert wird, aber in diesem Rahmen kann hier nicht darauf eingegangen werden.

Allgemeine Ausführungen zu JavaScript

Ursprünglich wurde JavaScript von Netscape unter dem Namen LiveScript für den Netscape-Navigator 2.0 entwickelt und später in JavaScript umbenannt. Im Gegenzug entwarf Microsoft mit JScript eine ähnliche Skriptsprache. Sind mehrere Konkurrenzprodukte im Umlauf, dann wird es Zeit für eine Standardisierung – in diesem Fall ECMAScript. Die ECMA (European Computer Manufacturers Association) legt heute den Kern der JavaScript-Spezifikation fest. Die aktuelle Version 6 (ECMAScript Harmony) wurde im Juni 2015 veröffentlicht.

JavaScript hat nichts mit der Sprache Java zu tun. Im JDK 8 gibt es aber eine neue JavaScript-Engine mit Namen

„Nashorn“, welche zu hundert Prozent ECMAScript-kompatibel ist.

Einbinden von JavaScript in HTML

JavaScript wird mit dem script-Element in ein HTML-Dokument eingebunden, welches sowohl im head-Element, als auch im body-Element verwendet werden kann. Die Einbindung innerhalb des body-Elements ist besser, weil dann das gesamte DOM geladen wird.

Es gibt zwei Möglichkeiten:

1. Der JavaScript-Code wird direkt zwischen dem öffnenden <script> und dem schließenden </script>-Tags notiert:

```
<script>
    // JavaScript-Code
    function beispielFunktion() {
        document.getElementById("einBeispiel").innerHTML
= "Hier wird JavaScript ausgeführt!";
    }
</script>
```

2. Im Script steht lediglich ein Verweis mit dem src-Attribut auf eine externe Datei:

```
<script> src="beispielScript.js"></script>
```

Praxistipps:

In einigen Fällen ist es nicht egal, an welcher Stelle das JavaScript im HTML-Dokument platziert wird. Parst ein Webbrowser das HTML-Dokument, dann wird eine bestimmte Struktur, das DOM, erstellt. Trifft der Parser auf ein Script-Element, dann hält die Verarbeitung an und der JavaScript-Code innerhalb des script-Elements wird ausgeführt, danach fährt er mit der Verarbeitung des restlichen HTML-Dokuments fort. Beim Einlesen größerer externer Script-Dateien, sollte JavaScript möglichst vor dem Ende des schließenden body-Tags notiert werden, wobei aber wieder der Anwendungszweck mitbetrachtet werden muss. Genauso verhält es sich mit dem DOM-Baum, denn es kann nicht auf Elemente im DOM-Baum zugegriffen werden, die vom Parser noch nicht verarbeitet wurden.

In dieser Kurseinheit wird nur auf die allgemeinste Form von JavaScript eingegangen, in welcher diese Scriptsprache in HTML-Dokumenten integriert und für das Verhalten der Webseite, genauer für die Interaktion, verantwortlich ist. Mit JavaScript werden Interaktivitätsgrade für Webseiten geboten, die mit einfachen HTML-Dokumenten nicht erzielt werden können. Wie auch HTML und CSS ist JavaScript im Klartext lesbar und kann in einem Texteditor geschrieben werden. Man benötigt keine zusätzlichen Werkzeuge, weil der Interpreter für JavaScript in jedem Webbrower bereit integriert ist.

Selbsttestaufgabe 2.10:

Wie wird JS in eine HTML-Seite eingebunden?

Verwendung von JavaScript

Generell muss angemerkt werden, dass JavaScript immer als optionale und nützliche Erweiterung für Webseiten zu sehen ist. HTML-, CSS- und JavaScript-Code müssen sauber getrennt bleiben.

Praxistipps:

- Ist JavaScript im Webbrowser deaktiviert, kann mit dem noscript-Tag ein entsprechender Hinweis ausgegeben werden.
- Die wichtigsten Informationen auf der Webseite müssen auch ohne JavaScript zugänglich sein, d. h. der Hauptinhalt muss mit jedem Webbrowser verfügbar sein, egal ob JavaScript aktiviert ist oder nicht. Das gleiche gilt für CSS, wenn Befehle für ein bestimmtes Medium nicht geeignet sind.
- Einzeilige Kommentare werden mit // eingeleitet und alles bis zum Zeilenende wird als Kommentar verwendet. Mehrzeilige Kommentare beginnen mit /* und enden mit */.
- Anweisungen werden immer mit einem Semikolon abgeschlossen.

JavaScript hat wie jede Programmiersprache alle gängigen arithmetischen Operatoren und Zuweisungsoperatoren an Bord. In JavaScript gibt es keine direkten Standardausgaben, weil es in erster Linie zur DOM-Manipulation verwendet wird. Dennoch gibt es Möglichkeiten, um JavaScript auszugeben:

- Bei der Einbindung in HTML gab es aber schon die „beispielFunktion()“ mit der Zeile:

```
document.getElementById("einBeispiel").innerHTML =  
    "Hier wird JavaScript ausgeführt!";
```

So wird ein HTML-Element mit dem id-Attribut-Wert „einBeispiel“ gesucht und der Inhalt des gefundenen Elements wird durch den Text „Hier wird JavaScript ausgeführt!“ ersetzt.

- Mit „document.write()“ kann man direkt in das HTML-Dokument schreiben, aber in der Praxis sollte man das so nicht tun.
- Für Debugging-Zwecke kann man die (Web)konsole nutzen. Um Ausgaben direkt in eine Konsole zu schreiben, notiert man folgendes:

```
Console.log("Hallo Beispiel");
```

- Informationen können auch mit der Dialogbox „alert“ angezeigt werden.

So wird ein HTML-Element mit dem id-Attribut-Wert „einBeispiel“ gesucht und der Inhalt des gefundenen Elements wird durch den Text „Hier wird JavaScript ausgeführt!“ ersetzt.

- Mit „document.write()“ kann man direkt in das HTML-Dokument schreiben, aber in der Praxis sollte man das so nicht tun.

- Für Debugging-Zwecke kann man die (Web)konsole nutzen. Um Ausgaben direkt in eine Konsole zu schreiben, notiert man folgendes:

```
Console.log("Hallo Beispiel");
```

- Informationen können auch mit der Dialogbox „alert“ angezeigt werden.

Selbsttestaufgabe 2.11:

Wie kann man in Javascript eine Bildschirmausgabe erzeugen, ohne die Funktion `document.write()` zu verwenden?

Das ECMAScript enthält auch einen Strict-Modus, welcher die Fehlerüberprüfung verbessert. Dieser Modus zwingt durch striktere Analyse, sauberes JavaScript zu schreiben. Aktiviert wird der Modus durch: „use strict“; JavaScript-Funktionen beginnen mit dem Schlüsselwort `function`. Funktionen sind in JavaScript vollwertige Objekte mit Methoden und Eigenschaften. Verzweigungen sind mit `if()`, `else`, `else if()` möglich.

Mit Schleifen können Anweisungen wiederholt werden. Mit „`break`“ kann die Schleifenaus-führung verlassen werden. Es gibt verschiedene Arten von Schleifen:

- die kopfgesteuerte „`for`“-Schleife,
- Weitere Formen der „`for`“-Schleife sind die „`for – in`“ und „`for – of`“ Schleife. Die „`for – of`“-Schleife ist erst im neuen Standard ECMAScript 6 enthalten und sollte auch verwendet werden,
- Der Vorteil ist, dass diese Schleife nicht die Eigenschaftnamen, sondern die Eigenschaftswerte durchläuft.
- die kopfgesteuerte „`while`“-Schleife,
- die fußgesteuerte „`do while`“-Schleife.

Boolesche Wahrheitswerte werden in JavaScript mit „`true`“ oder „`false`“ angegeben. Neben dieser Möglichkeit existieren verschiedene Vergleichsoperatoren, die zu `true` oder `false` auswerten.

JavaScript ist eine schwach typisierte Programmiersprache, daher können Variablen von einem beliebigen Typ sein, z. B. Zeichenkette, Nummer. Es gibt aber auch komplexere Typen wie Arrays oder Objekte. Variablennamen müssen mit einem Buchstaben beginnen und dürfen weder Leer- und Sonderzeichen noch deutsche Umlaute enthalten. JavaScript-Schlüsselwörter sind reserviert und dürfen nicht als Variablenname verwendet werden.

Eine lokale Variable ist so lange gültig, wie die Funktion ausgeführt wird. Eine globale Variable erlischt erst nach Verlassen der Webseite.

Praxistipp:

Man sollte unbedingt das Schlüsselwort „var“ vor den Variablennamen setzen, wenn es keinen Grund gibt eine globale Variable zu verwenden. Wird es weggelassen, erzeugt man eine solche globale Variable, die außerhalb einer Funktion gültig ist. Im ungünstigsten Fall werden bereits existierende globale Variablen überschrieben. Wie in jeder anderen Programmiersprache auch, sollten Variablen so lokal wie möglich vereinbart werden.

Mithilfe von Arrays können mehrere Werte in einer Variablen gespeichert werden. Das erste Element in einem Array hat immer den Index[0].

Die Haupttypen in JavaScript sind Objekte, auch die Webseite selbst ist für JavaScript ein Objekt. JavaScript selbst ist auch eine objektorientierte Sprache. Es gibt in JavaScript eigene Objekte, JavaScript-Objekte und HTML-Objekte.

Objekte entsprechen in etwa einem komplexen Datentyp wie etwa in der prozeduralen Programmierung, besitzen darüber hinaus auch Eigenschaften (Attributes) und Methoden, welche in das Objekt verkapselt sind und Implementierungsdetails verbergen. Jede Eigenschaft hat einen Namen und einen Wert. Die Methoden sind die Funktionen, welche nur mit dem Objekt ausgeführt werden können. Jedes Objekt erbt vom Prototyp des globalen Objekt-Konstruktors. Das Konzept der objektorientierten Programmierung soll an dieser Stelle aber nicht weiter vertieft werden. Es geht im Anschluss vielmehr darum, die fertigen Objekte von JavaScript und die darunterliegenden HTML-Dokumente zu verwenden.

In JavaScript wird eine Vielzahl von Objekten für das Browserfenster, das HTML-Dokument, zur Ereignisbehandlung, Zeichenketten, mathematische Operationen, das screen-Objekt für Bildschirmangaben, das location-Objekt für den Zugriff auf URI etc. bereitgestellt. Diese fertigen Objekte haben, wie oben beschrieben, verschiedene Eigenschaften und Methoden, welche man nutzen kann; z. B. hat ein String-Objekt die Eigenschaft length für die Länge der Zeichenkette oder auch solche Methoden wie substr() zur Ermittlung der Teilzeichenkette ab Position usw.

Die HTML-Objekte sind unterhalb des JavaScript-Objektes window und deren untergeordnetem Objekt document enthalten. Das window-Objekt entspricht dem im aktuellen Fenster enthaltenen DOM-Dokument. Das document-Objekt bezieht sich auf den Inhalt, der im Browser-Fenster angezeigt wird. Das document-Objekt macht es möglich, dass mit JavaScript auf alle Elemente des HTML-Dokuments zugriffen werden kann und ggf. zu ändern, was im nächsten Abschnitt erläutert wird.

Selbsttestaufgabe 2.12:

Warum ist es sinnvoll, das Schlüsselwort „var“ vor einen Variablenamen in JavaScript zu setzen?

Selbsttestaufgabe 2.13:

Inwiefern folgt JavaScript dem Konzept der objektorientierten Programmierung?

DOM-Manipulationen

HTML-DOM- Manipulation

Das HTML-DOM ist das Objektmodell und die Schnittstelle für HTML und damit der Standard, mit dem HTML-Elemente geändert, hinzugefügt oder gelöscht werden können. In den Ausführungen zu HTML wurde bereits erwähnt, dass HTML-Elemente eines Dokumentes eine hierarchische Baumstruktur bilden. Ganz oben ist das sogenannte document-Objekt, danach kommt das html-Element und anschließend head- und body-Element, gefolgt von weiteren Kind-Elementen. Alle Elemente dieses Baumes sind Knoten (engl. *nodes*) und miteinander verwandt durch Eltern-, Kind- oder Geschwisterbeziehungen. Aber auch die HTML-Attribute und die Inhalte der HTML-Elemente selbst sind Knoten des DOM-Baumes.

Im folgenden Konstrukt sind alle drei wichtigen Knotentypen enthalten:

```
<p lang="de">Schon wieder ein Beispiel ...</p>
```

Zuerst kommt der HTML-Elementknoten mit dem p-Element, dann der Attributknoten mit lang="de" und der Textknoten „Schon wieder ein Beispiel ...“

JavaScript kann auf jeden dieser Knoten mithilfe der verschiedenen HTML-DOM-Methoden und HTML-DOM- Eigenschaften zugreifen.

Um HTML-Elemente manipulieren zu können, müssen diese vorher gefunden werden:

- `document.getElemenetsByClassName()` – alle Knoten mit einem bestimmten Klassennamen werden gefunden,
- `document.getElementsByName()` - alle Knoten mit einem bestimmten Namen werden gefunden.

Mit `querySelector()` und `querySelectorAll()` kann auf komplexere Strukturen, z. B. wenn ein article-Element enthalten ist, zugegriffen werden.

Zugriff auf das komplette Body-Element hat man mit `document.body` und mit `document.documentElement` greift man auf den head- und body-Teil zu.

Zusätzlich stehen fertige Objektsammlungen und Eigenschaften zur Verfügung, um HTML-Elemente anzusprechen, z. B. `document.title`, welches den Inhalt des title-Elements liefert.

Außerdem ist eine gezielte Navigation durch den DOM-Baum möglich. Dafür existieren verschiedene Eigenschaften wie `parentNode`, `childNodes[n]`, `firstChild`, `lastChild`,

nextSibling, previousSibling. Einer ganz ähnlichen Navigation durch den Dokumentenbaum wird man in Kurseinheit 3 im Abschnitt XPath und XPointer bei den location steps wieder begegnen.

Hat man die Elemente ausfindig gemacht, kann man diese auch ändern:

- element.innerHTML= ändert den Inhalt eines Elements,
- element.attribute=, element.setAttribute(attribut, wert) ändern den Wert eines Attributs,
- element.style.property= - ändert den CSS-Style eines Elements.

Ein neues HTML-Element fügt man mit der Methode createElement() hinzu. Textinhalt wird mit createTextNode() erzeugt. In den DOM-Baum wird dieser Knoten mit appendChild() eingehängt.

Zum Entfernen eines HTML-Elements vom DOM-Baum gibt es die Methode removeChild(). Mit replaceChild() ersetzt man einen Knoten durch einen anderen. Will man ganze Fragmente vervielfältigen greift man auf die Methode cloneNode() zurück.

Um tatsächlich interaktiv sein zu können, muss mit JavaScript-Ereignissen gearbeitet werden. Diese Ereignisse sind in JavaScript schon enthalten. Seit dem DOM-Level3 sind die Ereignisse so umfangreich geworden, dass auch hier nur eine kurze Übersicht der gängigsten Ereignisse vorgestellt wird:

- Ereignisse der Benutzeroberfläche: z. B. onload, onresize, onscroll,
- Maus-Ereignisse: z. B. onclick, onmousemove, onmouseover,
- Formular-Ereignisse: z. B. onfocus, onselect, onsubmit,
- Touch-Ereignisse: z. B. touchstart, touchend, touchmove,
- Tastatur-Ereignisse: z. B. onkeydown, onkeyup, onkeypress,
- Video- und Audio-Ereignisse: z. B. onplay, onpause, onended.

Eine Ereignisbehandlung (engl. *event handler*) wird ausgeführt, wenn ein Event ausgelöst wird, auch hier gibt es mehrere Möglichkeiten, dieses einzurichten.

Diese Ausführungen zu JavaScript sollen genügen.

In der Praxis wird man bei der Erstellung von Webdokumenten unausweichlich mit dem Konzept der asynchronen Datenübertragung mit Ajax oder der JavaScript-Bibliothek jQuery konfrontiert werden.

Ajax steht für Asynchronous JavaScript and XML. Wenn eine http-Anfrage gestellt, läuft der Vorgang ohne Ajax synchron ab, d. h. die Scriptausführung wird angehalten bis die Daten vom Webserver geladen sind. Asynchron bedeutet im Zusammenhang mit Ajax, dass die Scriptausführung bei einer HTTP-Anfrage weitergeht, weil die Anfrage im Hintergrund ausgeführt wird. Mit Ajax kann man aber durchaus auch synchrone Vorgänge verwenden, wenn die Reihenfolge wichtig ist. In Ajax gibt es aber nicht nur das XML-Datenformat, um Informationen zwischen Client und Server auszutauschen. Das in der Praxis Bekannteste ist JSON (JavaScript Object Notation).

Mit jQuery steht eine umfangreiche JavaScript-Bibliothek zur Verfügung, in der immer wiederkehrende Funktionen angeboten werden, ob es nun um HTML-Manipulationen, CSS-Manipulationen o. ä. geht. jQuery hat komfortable Selektoren integriert oder auch elegante Mittel für komplexere Funktionen von JavaScript wie z. B. Effekte und Animationen. Unabhängig davon gibt es zahlreiche weitere Plugins.

Selbsttestaufgabe 2.14

Wie greift man per JavaScript auf ein DOM Element zu? Geben Sie ein Beispiel.

Selbsttestaufgabe 2.15

Gegeben sei die Datei test.html:



The screenshot shows a code editor window with the file 'test.html' open. The code is as follows:

```
1  <!doctype html>
2  <html>
3  <head>
4  <title>Test</title>
5
6  <script language="JavaScript">
7  function FensterAuf() {
8      setup = "toolbar=no,location=no,scrollbar=no,width=600,height=400,left=200,top=200";
9
10     fenster = window.open('', 'fenster', setup);
11     fenster.document.location.href = "thumb.png";
12 }
13 </script>
14
15 <style>
16     a:link {text-decoration: underline;}
17     a:hover {color: red;}
18 </style>
19 </head>
20
21 <body bgcolor="#eeeeff">
22
23 <A HREF="javascript:FensterAuf()"> Hier klicken</A>
24
25 </body>
26 </html>
```

Erläutern Sie stichpunktartig anhand der Zeilennummern, wie per JavaScript das HTML-DOM beeinflusst wird.

6 Zusammenfassung

In dieser Kurseinheit wurde eine kleine Lösung erläutert, wie sich das Modell der strukturierten Dokumente im Web mit HTML, CSS, und JavaScript umsetzen lässt. In den Spezialisierungen der Formate HTML für Text und Struktur, CSS für Formatvorgaben und über Plug-ins umgesetzte Grafikformate für Bilddaten finden wir das allgemeine Informatikprinzip Separation of Concerns realisiert.

Auch für einige andere Prinzipien und Paradigmen der Informatik finden sich Beispiele im Stoff dieser Kurseinheit: Die Kommunikation zwischen den verschiedenen Komponenten im Web ist nach einem bekannten und etablierten Schema strukturiert, der Client-Server-Architektur. Die Datenmodellierung von HTML-Dokumenten erfolgt nach dem wohlbekannten Schema der Baumstruktur. Die Prinzipien, nach denen in CSS3 die Spezifikation von CSS in Module gegliedert wird, finden sich im objektorientierten Design als schwache Kopplung zwischen Komponenten und als sogenannte Kohäsion innerhalb von Komponenten wieder. Schließlich wird bei HTML unterschieden zwischen dem konzeptionellen Modell, das hinter HTML-Dokumenten steht, und seiner syntaktischen Repräsentation als Zeichenkette. Hier zeigt sich die in der Informatik fundamentale Unterscheidung zwischen abstrakt formulierten Dingen und ihrer konkreten Repräsentation. Darüber hinaus zeigt sich der Ansatz, die Modellierung formal durchzuführen und auf diese Weise Eigenschaften des Modells genau formulieren und sogar im mathematischen Sinne beweisen zu können.

7 Anhang: Materialien zu HTML und CSS

a. Dokument im HTML-Format

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html40/strict.dtd">
<!-- Alternativen zu "strict" sind "loose" und "frameset". -->

<HTML>
  <HEAD>
    <TITLE>Konzept</TITLE>
  </HEAD>
  <BODY>
    <P>
      <IMG SRC="fbLogo.gif" ALT="Das Logo der
Frauenbeauftragten der TUM">
    </P>
    <H4>
      Prof. Dr. Anne Brüggemann-Klein<BR>
      Dr. Angelika Reiser<BR>
      Prof. Dr. Doris Schmitt-Landsiedel
    </H4>
    <H1>Ein Computer-Prädikat
von Studentinnen für Studentinnen
    </H1>
    <H2><B>12.10.1999</B>, Version <B>1.3</B></H2>
    <H3>Hintergrund und Ziele</H3>
    <P>
      Gerade in technischen Studiengängen spielen Computer
als alltägliches Arbeitsmittel eine wichtige Rolle.
      Es ist wesentlich, dass die Studierenden
      gleich zu Beginn des Studiums einen vertrauten und
      sicheren Umgang mit diesem Handwerkszeug haben. Eventuell
      vorhandene Hemmschwellen müssen sobald wie
      möglich abgebaut werden; davon ist eine
      studienzeitverkürzende Wirkung zu erwarten.
    </P>
    <P>
      Es ist bekannt, dass Schülerinnen weniger
      Zugang zu einem Computer haben als Schüler und
      Weniger häufig einen eigenen Computer besitzen.
      Das Computer-Prädikat richtet sich deshalb
      primär an Studentinnen, die dadurch eventuell
      fehlende Vorerfahrungen ausgleichen können.
    </P>
    <H3>Das Kursangebot des Computer-Prädikats</H3>
    <P>
      Das Computer-Prädikat bietet
      Kurzeinführungen mit praktischen Übungen in
      für das Studium an der TU München relevante
      Themen rund um den Computer. Jeder Kurs dauert als Block
      angeboten-drei Stunden und beinhaltet eine tutorielle
      Einführung in das Thema, praktische Übungen am
      Rechner und eine Abschlussbesprechung zu den
      Übungen. Kursthemen können sein:
    </P>
    <UL>
      <LI>Textverarbeitung mit Word</LI>
      <LI>Textverarbeitung mit LaTeX</LI>
      <LI>Textverarbeitung mit Star Office</LI>
      <LI>E-Mail</LI>
      <LI>
```

Browsen und Informationsnutzung im World-Wide Web

Erstellen von Informationsangeboten im World-Wide Web
(HTML)

Betriebssystem, Dateiorganisation
Gestaltung des Desktops (Window-Manager)
ASCII-Editoren (vi, emacs)
Kauf eines PCs
Datenverbindungen von zu Hause (analog, ISDN)

Fakultätsspezifische
Informationsinfrastrukturen

<P>
Jeder Kurs sollte sich auf ein enges Thema
konzentrieren. Die Themenliste ist offen
für Ergänzungen. Kriterium ist, daß der
Kursinhalt propädeutischer Natur für das
Studium an der TU ist und eine Form der Computernutzung
zum Inhalt hat. Ein Kurs kann
fakultätsübergreifend oder
fakultätspezifisch sein. Die Kurse sind in der
Regel auf eine spezielle Rechnerplattform (PC/Windows,
Unix) zugeschnitten.
</P>
<H3>Die Zielgruppe des Computer-Propädeutikums</H3>
<P>
Das Kursangebot richtet sich an Studentinnen im ersten
Semester. Wird ein Angebot von dieser Zielgruppe nicht
ausgeschöpft, steht es auch Studentinnen
höherer Semester und, in dritter Priorität,
männlichen Studierenden offen.
</P>
<H3>
Die Leiterinnen und Leiter des Computer-
Propädeutikums
</H3>
<P>
Die Kurse werden vorzugsweise von Studentinnen
höherer Semester konzipiert und durchgeführt.
Zur Ergänzung des Kursangebots können auch
Mitarbeiterinnen und Mitarbeiter oder Studenten
höherer Semester Kurse anbieten. Nur von
Studierenden angebotene Kurse können im Rahmen des
Programms finanziert werden.
</P>
<P>
Die Kursleiterinnen und -leiter sind selbst für die
Organisation von Seminar- und Rechnerräumen und von
Zugangsmöglichkeiten zu den Räumen am
Wochenende für ihren Kurs verantwortlich. Wir gehen
davon aus, daß die Studiendekaninnen und -dekanen
sie bei der
Organisation unterstützen.
</P>
<H3>Durchführung</H3>
<P>
Das Computer-Propädeutikum ist eine Initiative der
Frauenbeauftragten der TU München und wird vom
Frauenbüro (Kerstin Hansen, Anja Quindeau)
organisiert. Die TU München finanziert das Programm

aus dem Tutorienprogramm zur Verkürzung der Studiendauer. Für von Studierenden angebotene Kurse wird ein Werksvertrag abgeschlossen, der die Konzeption, Vorbereitung und Durchführung eines dreistündigen Kurses beinhaltet.

Konzeption und Vorbereitung eines Kurses werden mit 200 DM und jede Durchführung mit 100 DM honoriert. Das Frauenbüro erhät Hilfskraftmittel aus dem Tutorienprogrammfür die Organisation des Computer-Propädeutikums. Wir rechnen mit folgenden Kosten: 20 Kurse, jeder zweimal durchgeführt, zu 20 mal 200 DM plus 20 mal 2 mal 100 DM, also 8000 DM, plus 1000 DM für die Organisation im Frauenbüro.

</P>

<P>

Ausschreibung des Programms ist bereits erfolgt. Interessierte Kursleiterinnen und -leiter können ihr Kursangebot bis zum 5.11.1999 abgeben. Bitte benutzen Sie dazu unser Formular.

Die ersten Kurse sollen noch im November 1999 stattfinden. Bei Bedarf erfolgt eine zweite Ausschreibung, zu der neue Kursangebote abgegeben werden können, Anfang Dezember 1999. Die Ausschreibung des Programms, die Gestaltung des Kursangebots sowie die Ankündigung der Kurse erfolgt durch das Frauenbüro in Zusammenarbeit mit den Studierendenvertretungen, den Studiendekanen und den Frauenbeauftragten. Die Verträge werden nach Absprache mit dem Frauenbüro von der Hochschulverwaltung abgeschlossen.

</P>

<P>

Die Kurse sollen zu Randzeiten (Abends, Freitags Nachmittags oder am Wochenende) stattfinden.

</P>

</BODY>

</HTML>

b. Semantisch angereichertes Dokument im HTML-Format

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/REC-html40/strict.dtd">
<!-- Alternativen zu "Strict" sind "Transitional" und "Frameset".
--&gt;

&lt;html&gt;
  &lt;head&gt;&lt;title&gt;Konzept&lt;/title&gt;
    &lt;LINK REL="stylesheet" TYPE="text/css" HREF="style.css"&gt;
  &lt;/head&gt;

  &lt;body class="Konzept"&gt;
    &lt;div class="Briefkopf"&gt;
      &lt;div class="Institution"&gt;
        &lt;center&gt;&lt;img src="fbLogo.gif"/&gt;&lt;/center&gt;
      &lt;/div&gt;
      &lt;div class="Autorinnen"&gt;
        &lt;h4 class="Autorin"&gt;
          Prof. Dr. Anne Br&amp;uuml;ggemann-Klein&lt;br /&gt;
          Dr. Angelika Reiser&lt;br /&gt;
        &lt;/h4&gt;
      &lt;/div&gt;
    &lt;/div&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre>

```

```
Prof. Dr. Doris Schmitt-Landsiedel
</h4>
</div>
</div>
<h1 class="Hauptueberschrift">
  Ein Computer-Prop&auml;deutikum von Studentinnen f&uuml;r
  Studentinnen
</h1>
<h2 class="Nebenueberschrift">
  Konzept vom
  <b class="Datum">12.10.1999</b>
  , Version
  <b class="Version">1.3</b>
</h2>
<div class="Abschnitt">
<h3 class="Abschnittueberschrift">
  Hintergrund und Ziele
</h3>
<p class="P">
  Gerade in technischen Studieng&auml;ngen spielen
  Computer als allt&auml;gliches Arbeitsmittel eine
  wichtige Rolle. Es ist wesentlich, da&szlig; die
  Studierenden gleich zu Beginn des Studiums einen
  vertrauten und sicheren Umgang mit diesem Handwerkszeug
  haben. Eventuell vorhandene Hemmschwellen m&uuml;ssen
  sobald wie m&ouml;glich abgebaut werden;
  davon ist eine studienzeitverk&uuml;rzende Wirkung zu
  erwarten.
</p>
<p class="P">
  Es ist bekannt, da&szlig; Sch&uuml;lerinnen weniger
  Zugang zu einem Computer haben als Sch&uuml;ler und
  Weniger h&auuml;ufig einen eigenen Computer besitzen.
  Das Computer-Prop&auml;deutikum richtet sich deshalb
  prim&auml;r an Studentinnen, die dadurch eventuell
  fehlende Vorerfahrungen ausgleichen k&ouml;nnen.
</p>
</div>

<div class="Abschnitt">
<h3 class="Abschnittueberschrift">
  Das Kursangebot des Computer-Prop&auml;deutikums
</h3>
<p class="P">
  Das Computer-Prop&auml;deutikum bietet
  Kurzeinf&uuml;hrungen mit praktischen &Uuml;bungen in
  f&uuml;r das Studium an der TU M&uuml;nchen relevante
  Themen rund um den Computer.
  Jeder Kurs dauert-als Block angeboten-drei Stunden
  und beinhaltet eine tutorielle Einf&uuml;hrung in das
  Thema, praktische &Uuml;bungen am Rechner und eine
  Abschlu&szlig;besprechung zu den &Uuml;bungen.
  Kursthemen k&ouml;nnen sein:
</p>
<ul class="UL">
  <li class="LI">Textverarbeitung mit Word</li>
  <li class="LI">Textverarbeitung mit LaTeX</li>
  <li class="LI">Textverarbeitung mit Star Office</li>
  <li class="LI">E-Mail</li>
  <li class="LI">Browsen und Informationsnutzung im
    World-Wide Web
  </li>
  <li class="LI">Erstellen von Informationsangeboten im
    World-Wide Web (HTML)
```

```
</li>
<li class="LI">Betriebssystem, Dateiorganisation</li>
<li class="LI">Gestaltung des Desktops (Window-
    Manager)
</li>
<li class="LI">ASCII-Editoren (vi, emacs)</li>
<li class="LI">Kauf eines PCs</li>
<li class="LI">Datenverbindungen von zu Hause
    (analog, ISDN)
</li>
<li class="LI">Fakult&auml;tsspezifische
    Informationsinfrastrukturen</li>
</ul>
<p class="P">
    Jeder Kurs sollte sich auf ein enges Thema konzentrieren.
    Die Themenliste ist offen f&uuml;r Erg&auml;nzungen.
    Kriterium ist, da&szlig; der Kursinhalt
    prop&auml;deutischer Natur f&uuml;r das Studium an der
    TU ist und eine Form der Computernutzung zum Inhalt hat.
    Ein Kurs kann fakult&auml;ts&uuml;bergreifend
    oder fakult&auml;tsspezifisch sein. Die Kurse sind in der
    Regel auf eine spezielle Rechnerplattform (PC/Windows,
    Unix) zugeschnitten.
</p>
</div>

<div class="Abschnitt">
<h3 class="Abschnittueberschrift">
    Die Zielgruppe des Computer-Prop&auml;deutikums
</h3>
<p class="P">
    Das Kursangebot richtet sich an Studentinnen im ersten
    Semester. Wird ein Angebot von dieser Zielgruppe nicht
    ausgesch&ouml;pft, steht es auch Studentinnen
    h&ouml;herer Semester und, in dritter Priorit&auml;t,
    m&auml;nnlichen Studierenden offen.
</p>
</div>

<div class="Abschnitt">
<h3 class="Abschnittueberschrift">
    Die Leiterinnen und Leiter des
    Computer-Prop&auml;deutikums
</h3>
<p class="P">
    Die Kurse werden vorzugsweise von Studentinnen
    h&ouml;herer Semester konzipiert und durchgef&uuml;hrt.
    Zur Erf&auml;nzung des Kursangebots
    k&ouml;nnen auch Mitarbeiterinnen und Mitarbeiter
    oder Studenten h&ouml;herer Semester Kurse anbieten.
    Nur von Studierenden angebotene Kurse k&ouml;nnen im
    Rahmen des Programms finanziert werden.
</p>
<p class="P">
    Die Kursleiterinnen und -leiter sind selbst f&uuml;r die
    Organisation von Seminar- und Rechner&auml;umen und von
    Zugangsm&ouml;glichkeiten zu den R&auml;umen am
    Wochenende f&uuml;r ihren Kurs verantwortlich. Wir gehen
    davon aus, da&szlig; die Studiendekaninnen und -dekanen
    sie bei der Organisation unterst&uuml;tzen.
</p>
</div>

<div class="Abschnitt">
```

```

<h3 class="Abschnittueberschrift">
Durchf&uuml;hrung
</h3>
<p class="P">
Das Computer-Prop&auml;deutikum ist eine Initiative der
Frauenbeauftragten der TU M&uuml;nchen und wird vom
Frauenb&uuml;ro (Kerstin Hansen, Anja Quindeau)
organisiert.
Die TU M&uuml;nchen finanziert das Programm aus dem
Tutorienprogramm zur Verk&uuml;rzung der Studiendauer.
F&uuml;r von Studierenden angebotene Kurse
wird ein Werksvertrag abgeschlossen, der
die Konzeption, Vorbereitung und Durchf&uuml;hrung
eines dreist&uuml;ndigen Kurses beinhaltet.
Konzeption und Vorbereitung eines Kurses werden mit 200
DM und jede Durchf&uuml;hrung mit 100 DM honoriert.
Das Frauenb&uuml;roerh&auml;lt Hilfskraftmittel
aus dem Tutorienprogrammf&uuml;r die
Organisation des Computer-Prop&auml;deutikums.
Wir rechnen mit folgenden Kosten: 20 Kurse,
jeder zweimal durchgef&uuml;hrt,
zu 20 mal 200 DM plus 20 mal 2 mal 100 DM, also 8000 DM,
plus 1000 DM f&uuml;r die Organisation im
Frauenb&uuml;ro.
</p>
<p class="P">
Eine erste <a class="A">Ausschreibung</a>
des Programms ist bereits erfolgt. Interessierte
Kursleiterinnen und -leiter k&ouml;nnen ihr Kursangebot
bis zum 5.11.1999 abgeben.
Bitte benutzen Sie dazu unser
<a class="A">Formular</a>. Die ersten Kurse sollen noch
im November 1999 stattfinden. Bei Bedarf erfolgt eine
zweite Ausschreibung, zu der neue Kursangebote abgegeben
werden k&ouml;nnen, Anfang Dezember 1999.
Die Ausschreibung des Programms, die Gestaltung des
Kursangebots sowie die Ank&uuml;ndigung der Kurse erfolgt
durch das Frauenb&uuml;ro
in Zusammenarbeit mit den Studierendenvertretungen,
den Studiendekanen und den Frauenbeauftragten.
Die Vertr&auml;ge werden nach Absprache mit dem
Frauenb&uuml;ro von der Hochschulverwaltung
abgeschlossen.
</p>
<p class="P">
Die Kurse sollen zu Randzeiten (Abends, Freitags
Nachmittags oder am Wochenende) stattfinden.
</p>
</div>
</body>
</html>

```

c. Stylesheet im CSS-Format

```

body {
font-family: Helvetica, sans-serif;
font-size: small;
font-weight: normal;
color: blue;
background-color: lightyellow;
margin: 15% 10%;
}

```

```
h1 {  
    font-size: large;  
    font-weight: bold;  
    color: lightyellow;  
    background-color: blue;  
    text-align: center;  
}  
  
h2 {  
    font-size: normal;  
    font-weight: bold;  
    text-align: center;  
}  
  
h3 {  
    font-size: normal;  
    font-weight: bold;  
    color: lightyellow;  
    background-color: blue;  
}  
  
.Autorin {  
    color: black;  
    background-color: white;  
    font-size: small;  
    font-weight: bold;  
    text-align: right;  
    margin-top: 0pt;  
    margin-bottom: 0pt;  
}  
  
.Briefkopf {  
    margin-bottom: 2em;  
}  
  
img {  
    width: 100%;  
}  
  
p {  
    margin-top: 0pt;  
    margin-bottom: 0pt;  
}  
  
a:link {  
    color: red;  
    text-decoration: none;  
}  
  
a:active {  
    color: yellow;  
    text-decoration: none;  
}  
  
a:visited {  
    color: green;  
    text-decoration: none;  
}
```

d. Beispiel in HTML5 Semantik

```
<!DOCTYPE html>

<html lang="de">
<head>
    <meta charset="utf-8">
    <title>Konzept</title>
    <link rel="stylesheet" href="styles.css">
    <link rel="Shortcut Icon" href="favicon.ico" type="image/x-
icon">
</head>

<body>

    <!-- Einen page-wrapper macht man gerne, um etwa Maximal-
        Breiten oder Abstände zu realisieren. Den BODY kann
        Man auf diese Weise in Ruhe lassen: -->
    <div class="page-wrapper">

        <!-- Der Kopfbereich der Seite, sauber als header
            ausgezeichnet -->
        <header>

            <!-- Das obligatorische Logo oder Bild -->
            

            <!-- Die Liste der Autorinnen -->
            <ul class="autorinnen">
                <li>Prof. Dr. Anne Brügmann-Klein</li>
                <li>Dr. Angelika Reiser</li>
                <li>Prof. Dr. Doris Schmitt-Landsiedel</li>
            </ul>

        </header>

        <!-- Der Hauptbereich der Seite mit zwei Sektionen -->
        <main>

            <!-- Hauptüberschrift und Meta-Informationen -->
            <section>
                <h1>Ein Computer-Propädeutikum von Studentinnen
für
                Studentinnen</h1>
                <p class="version">Konzept vom 12.10.1999 , Version
                1.3</p>
            </section>

            <!-- Ordentlich strukturierter Inhalt dieser Seite -->
            <section>

                <h2>Hintergrund und Ziele</h2>
                <p>Gerade in technischen Studiengängen spielen
                Computer als alltägliches Arbeitsmittel eine
                wichtige Rolle. Es ist wesentlich, daß die
                Studierenden gleich zu Beginn des Studiums einen
                vertrauten und sicheren Umgang mit diesem
                Handwerkszeug haben. Eventuell vorhandene
                Hemmschwellen müssen sobald wie möglich abgebaut
                werden; davon ist eine studienzeitverkürzende

```

Wirkung zu erwarten.</p>

zu einem Computer haben als Schüler und Weniger häufig einen eigenen Computer besitzen. Das Computer-Propädeutikum richtet sich deshalb primär an Studentinnen, die dadurch eventuell fehlende Vorerfahrungen ausgleichen können.</p>

<h2>Das Kursangebot des Computer-Propädeutikums</h2>

Das Computer-Propädeutikum bietet Kurzeinführungen mit praktischen Übungen in für das Studium an der TU München relevante Themen rund um den Computer. Jeder Kurs dauert als Block angeboten-drei Stunden und beinhaltet eine tutorielle Einführung in das Thema, praktische Übungen am Rechner und eine Abschlußbesprechung zu den Übungen. Kursthemen können sein:</p>

Textverarbeitung mit Word

Textverarbeitung mit LaTeX

Textverarbeitung mit Star Office

E-Mail

Browsen und Informationsnutzung im World-Wide Web

Erstellen von Informationsangeboten im World-Wide Web (HTML)

Betriebssystem, Dateiorganisation

Gestaltung des Desktops (Window-Manager)

ASCII-Editoren (vi, emacs)

Kauf eines PCs

Datenverbindungen von zu Hause (analog, ISDN)

Fakultätsspezifische Informationsinfrastrukturen

Jeder Kurs sollte sich auf ein enges Thema konzentrieren. Die Themenliste ist offen für Ergänzungen. Kriterium ist, dass der Kursinhalt propädeutischer Natur für das Studium an der TU ist und eine Form der Computernutzung zum Inhalt hat. Ein Kurs kann fakultätsübergreifend oder fakultätsspezifisch sein. Die Kurse sind in der Regel auf eine spezielle Rechnerplattform (PC/Windows, Unix) zugeschnitten.</p>

<h2>Die Zielgruppe des Computer-Propädeutikums</h2>

Das Kursangebot richtet sich an Studentinnen im ersten Semester. Wird ein Angebot von dieser Zielgruppe nicht ausgeschöpft, steht es auch Studentinnen höherer Semester und, in dritter Priorität, männlichen Studierenden offen.</p>

```
        <h2>Die Leiterinnen und Leiter des Computer-  
Propädeutikums</h2>  
        <p>Die Kurse werden vorzugsweise von Studentinnen  
höherer Semester konzipiert und durchgeführt. Zur  
Ergänzung des Kursangebots können auch  
Mitarbeiterinnen und Mitarbeiter oder Studenten  
höherer Semester Kurse anbieten. Nur von  
Studierenden angebotene Kurse können im Rahmen des  
Programms finanziert werden.</p>  
        <p>Die Kursleiterinnen und -leiter sind selbst für  
die  
Organisation von Seminar- und Rechnerräumen und von  
Zugangsmöglichkeiten zu den Räumen am Wochenende für  
ihren Kurs verantwortlich. Wir gehen davon aus, daß  
die Studiendekaninnen und -dekanen sie bei der  
Organisation unterstützen.</p>  
  
        <h2>Durchführung</h2>  
        <p>Das Computer-Propädeutikum ist eine Initiative  
der  
Frauenbeauftragten der TU München und wird vom  
Frauenbüro (Kerstin Hansen, Anja Quindeau)  
organisiert. Die TU München finanziert das Programm  
aus dem Tutorienprogramm zur Verkürzung der  
Studiendauer. Für von Studierenden angebotene Kurse  
wird ein Werksvertrag abgeschlossen, der die  
Konzeption, Vorbereitung und Durchführung eines  
dreistündigen Kurses beinhaltet. Konzeption und  
Vorbereitung eines Kurses werden mit 200 DM und jede  
Durchführung mit 100 DM honoriert. Das  
Frauenbüro erhält Hilfskraftmittel aus dem  
Tutorienprogramm für die Organisation des Computer-  
Propädeutikums. Wir rechnen mit folgenden Kosten: 20  
Kurse, jeder zweimal durchgeführt, zu 20 mal 200 DM  
plus 20 mal 2 mal 100 DM, also 8000 DM, plus 1000 DM  
für die Organisation im Frauenbüro.</p>  
        <p>Eine erste Ausschreibung des Programms ist be-  
reits  
erfolgt. Interessierte Kursleiterinnen und -leiter  
können ihr Kursangebot bis zum 5.11.1999 abgeben.  
Bitte benutzen Sie dazu unser Formular. Die ersten  
Kurse sollen noch im November 1999 stattfinden. Bei  
Bedarf erfolgt eine zweite Ausschreibung, zu der  
neue Kursangebote abgegeben werden können, Anfang  
Dezember 1999. Die Ausschreibung des Programms, die  
Gestaltung des Kursangebots sowie die Ankündigung  
der Kurse erfolgt durch das Frauenbüro in  
Zusammenarbeit mit den Studierendenvertretungen, den  
Studiendekanen und den Frauenbeauftragten. Die  
Verträge werden nach Absprache mit dem Frauenbüro  
von der Hochschulverwaltung abgeschlossen.  
        Die Kurse sollen zu Randzeiten (Abends, Freitags  
Nachmittags oder am Wochenende) stattfinden.</p>  
    </section>  
  </main>  
</div>  
</body>  
</html>
```

e. Stylesheet in CSS3 Semantik

```
/*
styles.css
Version: 2016-04-21 - 15:45
*/

/* Erstmal alle Browser "resetten", damit alle ungefähr
gleich rendern, alle Elemente erstmal KEINE nativen
Abstände etc haben: */

html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}

/* HTML5 display-role reset for older browsers (IE8 etc) */
article, aside, details, figcaption, figure, footer, header,
hgroup, menu, nav, section {
    display: block;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}

/* Hier die spezifischen Styles: */
body {
    font-family: 'Trebuchet MS', 'Lucida Grande', 'Lucida Sans
    Unicode', 'Lucida Sans', Tahoma, sans-serif;
    line-height: 1.6;
    color: blue;
    background-color: lightyellow;
}

.page-wrapper {
    /* Lesefluss optimieren durch Zeilenbreite begrenzen! */
    max-width: 60em;
    margin: 0 auto;
    padding: 5% 10%;
}

h1,
```

```
h2 {
    font-weight: bold;
    color: lightyellow;
    background-color: blue;
    text-align: center;
}

h1 {
    margin: 1em 0;
    font-size: 1.1em;
}
h2 {
    margin: 3em 0 1em 0;
}

p {
    margin-bottom: 0.5em;
}
ul {
    margin: 0 0 1em 3em;
}

/* im header: */
p.version {
    font-weight: bold;
    text-align: center;
}
ul.autorinnen {
    font-size: 0.75em;
    text-align: right;
    list-style: none;
    color: black;
    background-color: white;
    font-weight: bold;
    margin: 0;
    padding: 0.5em;
}
header {
    text-align: center;
}

a:link {
color: red;
text-decoration: underline;
}

a:focus,
a:hover,
a:active {
color: yellow;
text-decoration: none;
}

a:visited {
color: green;
text-decoration: none;
}
```

f. Mit CSS-Stylesheet formatiertes Dokument


Technische
Universität
München

 Die Frauenbeauftragte

Prof. Dr. Anne Brüggemann-Klein
Dr. Angelika Reiser
Prof. Dr. Doris Schmitt-Landsiegel

Ein Computer-Propädeutikum von Studentinnen für Studentinnen

Konzept vom 12. Oktober 1999, Version 1.3

Hintergrund und Ziele

Gerade in technischen Studiengängen spielen Computer als alltägliches Arbeitsmittel eine wichtige Rolle. Es ist wesentlich, daß die Studierenden gleich zu Beginn des Studiums einen vertrauten und sicheren Umgang mit diesem Handwerkzeug haben. Eventuell vorhandene Hemmschwellen müssen sobald wie möglich abgebaut werden; davon ist eine studienzeitverkürzende Wirkung zu erwarten.

Es ist bekannt, daß Schülerinnen weniger Zugang zu einem Computer haben als Schüler und weniger häufig einen eigenen Computer besitzen. Das Computer-Propädeutikum richtet sich deshalb primär an Studentinnen, die dadurch eventuell fehlende Vorerfahrungen ausgleichen können.

Das Kursangebot des Computer-Propädeutikums

Das Computer-Propädeutikum bietet Kurzeinführungen mit praktischen Übungen in für das Studium an der TU München relevante Themen rund um den Computer. Jeder Kurs dauert als Block angeboten drei Stunden und beinhaltet eine tutorielle Einführung in das

8 Literaturverzeichnis

- [F98] F. Feizabadi. History of the World Wide Web. In M. Abrams, editor, World Wide Web: Beyond the Basics. Prentice Hall, Upper Saddle River, New Jersey, 1998.
- [M00] E. A. Meyer. Cascading Style Sheets: The Definite Guide. O'Reilly, Sebastopol, California, 2000.
- [N99] J. Niederst. Web Design in a Nutshell. O'Reilly, Sebastopol, California, 1999.
- [LB97] H. W. Lie and B. Bos. Cascading Style Sheets: Designing for the Web. Addison Wesley Longman, Essex, England, 1997.
- [W3Crdf14] W3C Ressource Description Framework. URL: <https://www.w3.org/TR/rdf11-primer/>

Letzter Zugriff 26.10.2016

- [W3Chtml5CM] W3C Working Draft HTML5 Content Models
<https://www.w3.org/TR/2011/WD-html5-20110525/content-models> Letzter Zugriff 06.03.2023

9 Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 2.1 SGML als Basis..... | 5 |
| Abbildung 2.2 Werdegang von HTML zum heutigen HTML5 [F98]..... | 6 |
| Abbildung 3.1 Weltweite Informationsverteilung durch das Web | 9 |
| Abbildung 3.2 Das neue Content-Modell für HTML [W3Chtml5CM] | 19 |
| Abbildung 4.1 Das klassische Boxmodell für CSS | 45 |
| Abbildung 4.2 Das neue Boxmodell für CSS | 46 |
| Abbildung 4.3 Behandlung von Blockboxen in Inlineboxen in Browsern | 49 |

10 Tabellenverzeichnis

| | |
|---|---|
| Tabelle 4.1 Beschriftung fehlt | Fehler! Textmarke nicht definiert. |
| Tabelle 4.2 Übersicht über einfache Selektoren | 34 |
| Tabelle 4.3 Übersicht über Struktur-Pseudoklassen | 35 |
| Tabelle 4.4 Übersicht über dynamische und weitere Pseudoklasse..... | 35 |
| Tabelle 4.5 Übersicht über Pseudoelemente | 35 |
| Tabelle 4.6 Übersicht über Kombinationsselektoren..... | 36 |

11 Lösung der Selbsttestaufgaben

Selbsttestaufgabe 2.1

Auf welcher Auszeichnungssprache basierte HTML ursprünglich?

HTML basierte auf der standardisierten generalisierten Auszeichnungssprache (engl. *standard generalized markup language, SGML*)

Selbsttestaufgabe 2.2

Welches Gremium hat es seit 1996 zur Aufgabe, über die Entwicklungsstandards von HTML zu entscheiden?

Das World-Wide Web Consortium (W3C)

Selbsttestaufgabe 2.3

Warum ist die Verwendung von Metadaten in Dokumenten unverzichtbar geworden?

Die Menge an Informationen, die über das Internet zur Verfügung stehen, ist mittlerweile nahezu unüberschaubar geworden.

Das Auffinden benötigter Dokumente über Suchanfragen wäre nicht möglich, wenn jedes im Web bereitgestellte Dokument Wort für Wort durchsucht werden müsste. Daten zum Inhalt von Dokumenten werden als beschreibende Meta-Information, und ihre maschinenlesbare Repräsentation als Metadaten, also als Daten, die Information über Daten beinhalten, bezeichnet.

Metadaten, die Inhalte beschreiben und schnell durchsucht werden können, sind daher eine wichtige Voraussetzung.

Selbsttestaufgabe 2.4

Was sind die drei grundlegenden Konzepte des frühen HTML-basierten Webs?

- weltweite Adressierung von Ressourcen über URLs,
- einfaches Protokoll zum Anfordern und Ausliefern von Ressourcen (HTTP),
- Hypertextparadigma zur Navigation zwischen den Ressourcen, z. B. über Anker in HTML-Dokumenten.

Selbsttestaufgabe 2.5

Charakterisieren Sie das frühe HTML-basierte Web als Hypertextsystem.

Wodurch hebt es sich von anderen klassischen Hypertextsystemen ab?

- weltweit verteilter Informationsbestand

- offener, von überall her zugänglicher Informationsbestand
- keine direkten technischen Möglichkeiten, einzelne Hypertextbereiche nach außen als Einheit kenntlich zu machen
- zweistellige, unidirektionale, in das Quelldokument eingebettete Hypertextlinks
- Hypertextlinks als einzige Gliederungsmöglichkeit für Hypertexte, keine direkte technische Unterstützung anderer Organisationsprinzipien wie beispielsweise hierarchische Gliederung
- einfache, standardisierte und mit verbreiteten Werkzeugen bearbeitbare Formate (HTML) zur Textdarstellung
- keine Nutzermodellierung

Selbsttestaufgabe 2.6

Öffnen Sie ein Dokument, das Sie in Ihrem Textsystem geschrieben haben. Übertragen Sie den Text des Dokumentes nach HTML. Repräsentieren Sie alle Strukturen Ihres Ausgangsdokuments im HTML-Markup.

--> Keine Musterlösung

Selbsttestaufgabe 2.7

Arbeiten Sie weiter mit dem HTML-Dokument aus Selbsttestaufgabe 2.6. Definieren Sie ein CSS-Stylesheet, das die Formatierung Ihrer ursprünglichen Vorlage aus einem Textsystem möglichst originalgetreu reproduziert.

Erkennen Sie an Ihrem Fallbeispiel Grenzen von CSS?

Selbsttestaufgabe 2.8

Ein HTML-Dokument repräsentiere Aufgaben mit folgendem Markup:

```
<DIV CLASS="Aufgabe">  
  <P><Der Text der Aufgabe></P>  
</DIV>  
  
<DIV CLASS="Aufgabe">  
  <P><Der etwas längere,  
  über mehrere Zeilen gehende  
  Text einer weiteren Aufgabe></P>  
</DIV>
```

Entwerfen Sie eine CSS-Regel mit folgender Wirkung: Der erste Absatz einer Aufgabe enthält vor dem eigentlichen Aufgabentext in Fettschrift das Schlüsselwort „Aufgabe“, gefolgt von der Nummer der Aufgabe, also:

Aufgabe 1«Der Text der Aufgabe»

Aufgabe 2«Der etwas längere, über mehrere Zeilen gehende Text einer weiteren Aufgabe»

```
body {  
    counter-reset: AufgC;  
}  
.Aufgabe > P:first-child:before {  
    content: "Aufgabe " counter(AufgC) " ";  
    counter-increment: AufgC;  
    font-weight: bold;  
    display: inline;  
}
```

Selbsttestaufgabe 2.9

Entwerfen Sie eine CSS-Regel, die H2 Elemente fett setzt und in den darunter liegenden Text / Absatz laufen lässt.

```
H2 {  
    display: run-in;  
    font-size: 100%;  
    font-weight: bold;  
}
```

Selbsttestaufgabe 2.10

Wie wird JS in eine HTML-Seite eingebunden?

Es gibt zwei Möglichkeiten:

1. Der JavaScript-Code wird direkt zwischen dem öffnenden und dem schließenden </script> Tags notiert:

```
<script>  
    // JavaScript-Code  
  
    function beispielFunktion() {  
  
        document.getElementById("einBeispiel").innerHTML = "Hier wird  
        JavaScript ausgeführt!";  
  
    }  
</script>
```

2. Im Script steht lediglich ein Verweis mit dem src-Attribut auf eine externe Datei:

```
<script> src="beispielScript.js"></script>
```

Selbsttestaufgabe 2.11

Wie kann man in Javascript eine Bildschirmausgabe erzeugen, ohne die Funktion document.write() zu verwenden?

```
document.getElementById("textelement").innerHTML = "Hier wird das gefundene Element per  
JavaScript ersetzt. ";
```

Selbsttestaufgabe 2.12

Warum ist es sinnvoll, das Schlüsselwort var vor einen Variablennamen in JavaScript zu setzen?

Wird es weggelassen, erzeugt man eine globale Variable, die auch außerhalb einer Funktion gültig ist. Im ungünstigsten Fall werden bereits existierende globale Variablen überschrieben. Wie in jeder anderen Programmiersprache auch, sollten Variablen so lokal wie möglich definiert werden.

Selbsttestaufgabe 2.13

Inwiefern folgt JavaScript dem Konzept der objektorientierten Programmierung?

JavaScript kennt komplexe Variablentypen wie Objekte. Es gibt in JavaScript eigene Objekte, JavaScript-Objekte und HTML-Objekte. Diese verfügen über Eigenschaften (Attributes) und Methoden, welche in das Objekt verkapselt sind und Implementierungsdetails verbergen. Jede Eigenschaft hat einen Namen und einen Wert. Die Methoden sind die Funktionen, welche nur mit dem Objekt ausgeführt werden können. Jedes Objekt erbt vom Prototyp des globalen Objekt-Konstruktors.

In JavaScript werden eine Vielzahl von Objekten beispielsweise für das Browserfenster, das HTML-Dokument oder zur Ereignisbehandlung bereitgestellt.

Selbsttestaufgabe 2.14

Wie greift man per JavaScript auf ein DOM Element zu? Geben Sie ein Beispiel.

JavaScript kann auf jeden der Knoten im HTML-DOM mithilfe der verschiedenen HTML-DOM-Methoden und HTML-DOM-Eigenschaften zugreifen.

Das document-Objekt macht es z. B. möglich, dass mit JavaScript auf alle Elemente des HTML-Dokuments zugriffen werden kann:

document.getElementsByName() - alle Knoten mit einem bestimmten Namen werden gefunden

Selbsttestaufgabe 2.15

Gegeben sei die Datei test.html:

Erläutern Sie stichpunktartig anhand der Zeilennummern, wie per JavaScript das HTML-DOM beeinflusst wird:

```
test.html
1  <!doctype html>
2  <html>
3  <head>
4  <title>Test</title>
5
6  <script language="JavaScript">
7  function FensterAuf() {
8      setup = "toolbar=no,location=no,scrollbar=no,width=600,height=400,left=200,top=200";
9
10     fenster = window.open('', 'fenster', setup);
11     fenster.document.location.href = "thumb.png";
12 }
13 </script>
14
15 <style>
16     a:link {text-decoration: underline;}
17     a:hover {color: red;}
18 </style>
19 </head>
20
21 <body bgcolor="#eeeeff">
22
23 <A HREF="javascript:FensterAuf()"> Hier klicken</A>
24
25 </body>
26 </html>
```

Zeile 6-13: JavaScript-Bereich

Zeile 7-12: Definition der JavaScript-Funktion

Zeile 10: Öffnen eines neuen Fensters

Zeile 11: Angabe des Dokuments, das geöffnet werden soll

Zeile 23: Aufruf der JavaScript Funktion per Link im HTML-Bereich des Dokuments

000 000 000 (00/23)

00000-0-00-S1

Alle Rechte vorbehalten
© 2023 FernUniversität in Hagen
Fakultät für Mathematik und Informatik

Prof. Dr.-Ing. Matthias L. Hemmje

01877

Dokumenten- und Wissensmanagement im Internet

Kurseinheit 1:
Das Modell der strukturierten Dokumente

Fakultät für
Mathematik und
Informatik

Der Inhalt dieses Dokumentes darf ohne vorherige schriftliche Erlaubnis durch die FernUniversität in Hagen nicht (ganz oder teilweise) reproduziert, benutzt oder veröffentlicht werden. Das Copyright gilt für alle Formen der Speicherung und Reproduktion, in denen die vorliegenden Informationen eingeflossen sind, einschließlich und zwar ohne Begrenzung Magnetspeicher, Computerausdrucke und visuelle Anzeigen. Alle in diesem Dokument genannten Gebrauchsnamen, Handelsnamen und Warenbezeichnungen sind zumeist eingetragene Warenzeichen und urheberrechtlich geschützt. Warenzeichen, Patente oder Copyrights gelten gleich ohne ausdrückliche Nennung. In dieser Publikation enthaltene Informationen können ohne vorherige Ankündigung geändert werden.

Inhaltsverzeichnis

| | |
|---|-----|
| Inhaltsverzeichnis | III |
| 1. Einleitung..... | 4 |
| 2. Der Dokumentenbegriff..... | 4 |
| 3. Der Modellbegriff | 5 |
| 4. Realisierung des Dokumentenmodells | 5 |
| 5. Zeichencodierungen | 27 |
| 6. Zusammenfassung und Ausblick | 44 |
| 7. Anhang: Formate am Beispiel..... | 45 |
| 8. Literaturverzeichnis | 58 |
| 9. Abbildungsverzeichnis | 60 |
| 10. Tabellenverzeichnis..... | 60 |
| 11. Lösungen der Selbsttestaufgaben..... | 61 |

1. Einleitung

Dies ist die erste Kurseinheit des Kurses 1877 „Dokumenten- und Wissensmanagement im Internet“.

In diesem Kursabschnitt werden allgemeine Formate für Dokumente sowie für strukturierte Dokumente, die im Internet verfügbar sind, vorgestellt. Die Kursteilnehmer setzen sich zunächst mit einer Präzisierung des Dokumentenbegriffs und dem Modells auseinander.

Es folgt eine detaillierte Betrachtung des Modells der strukturierten Dokumente, in der die Studierenden verschiedene Arten von Textdokumenten, Aspekte der Strukturierung sowie Umsetzungsmöglichkeiten des Modells analysieren.

Weiter untersuchen sie verschiedene Modelle von Zeichencodierungen, die den Inhalt von Dokumenten in maschinenlesbarer Form repräsentieren, sowie deren einzelne Bestandteile, beispielsweise Codierungsschema und Übertragungssyntax.

2. Der Dokumentenbegriff

Für statisch-passive Dokumente wurde bereits in den achtziger Jahren das Modell der strukturierten Dokumente entwickelt. Durch dieses Modell wird eine Unterteilung in textuellen Inhalt, logische Struktur und grafisches Format möglich. Die

Quellrepräsentation des Dokuments bezieht sich auf den textuellen Inhalt und logische Struktur. Gesteuert von unabhängigen Formatvorgaben (engl. *stylesheets*) lässt sich daraus das grafische Format generieren.

Moderne Systeme zur Erstellung und Verwaltung von Dokumenten und großen Dokumentenbeständen profitieren vom Modell der strukturierten Dokumente, da es ein hohes Maß an Flexibilität bietet. Die logische Strukturierung unterstützt Anwendungen der Dokumentenverwaltung wie die automatische Generierung von Katalegeinträgen und Verzeichnissen oder die Wiedergewinnung von Information (engl. *information retrieval*) nach bestimmten logischen Kriterien und erlaubt es ganz allgemein, Dokumente als Daten zu betrachten, die mithilfe von Computersystemen verwaltet und manipuliert werden können.

Die Unabhängigkeit der Stylesheets von den Dokumenten ermöglicht auf der einen Seite, dass ein- und dasselbe Dokument durch Kombination mit verschiedenen Stylesheets passend zum Anwendungszweck formatiert werden kann; auf der anderen Seite kann man verschiedene Dokumente durch Kombination mit ein- und demselben Stylesheet einheitlich formatieren. Auch lässt sich die Struktur

Quellrepräsentation

Stylesheets

beim Erstellen und Lesen von Dokumenten nutzen, etwa zur Navigation und zur Definition von Sichten etwa von Outline-Sichten.

3. Der Modellbegriff

Modelle sind ein wichtiges, methodisches Mittel. Die Modellierung ist eine vereinfachte und auf das im aktuellen Kontext Wesentliche reduzierte und – unter geeigneten Gesichtspunkten – systematisierte Darstellung. Modelle sind aus vielen Bereichen bekannt wie etwa aus der Architektur, der Biologie oder der Physik. Modelle können physikalischer oder gedanklicher Art sein; letztere können umgangssprachlich oder formal definiert sein. Gegenstandsbereich einer Modellierung können Konzepte, Vorgänge oder Erscheinungen und ihre Wechselwirkungen sein. Modelle können unterschiedlichen Zwecken dienen: etwa der Entwicklung einer gemeinsamen Sicht und Terminologie, der Vermittlung erprobter Vorgehensweisen und Techniken, dem rationalen Beweisen oder Widerlegen von Hypothesen oder der Ableitung von Handlungsempfehlungen.

Zwei zentrale und umgangssprachlich erläuterte Modelle werden in dieser Kurseinheit eingeführt und näher erläutert: das Modell der strukturierten Dokumente und ein Modell für Zeichencodierungen.

4. Realisierung des Dokumentenmodells

Hier wird anhand verschiedener Beispiele und Systeme erläutert, wie das Modell der strukturierten Dokumente mit Zeichencodierungen, eingebetteten Auszeichnungen (Markierungen, engl. *markup*), Dokumentengrammatiken und Stylesheets in der Praxis umgesetzt wird. Auf der technischen Seite wird das Thema der Zeichencodierungen vertieft. In den nachfolgenden Kurseinheiten wird das Thema der strukturierten Dokumente anhand gängiger Internettechnologien weiter beleuchtet. Dazu gehören XML und die zugehörigen weiteren XML-basierten Sprachfamilien.

Dokumente näher betrachtet

Historisch gesehen findet sich beispielsweise in Brockhaus' Konversationslexikon in der vierzehnten Auflage von 1894 folgendes unter dem Begriff „Dokument“:

„Im weiteren Sinne jeder Gegenstand, welcher dazu dient, die Wahrheit einer zu erweisenden Tatsache, besonders einer für ein Rechtsverhältnis erheblichen Tatsache, zu bestätigen. Im engeren Sinne versteht man darunter Urkunden oder Schriftstücke, im Gegensatz zu anderen körperlichen Beweisstücken, wie Grenzsteinen, Wappen, beschädigten Sachen.“

ursprünglicher
Dokumentenbegriff

Ursprünglich waren Dokumente also tatsächlich physische Objekte, Gegenstände, die als einmalige Träger eines in menschliche Sprache gekleideten und schriftlich festgehaltenen Gedankens oder Sachverhalts fungierten. Ein Dokument neuerer Art, etwa ein sauber getippter Bogen Papier mit Name, Adresse, Personalausweisnummer und weiterer Information zur Person erfüllt einfach nicht denselben Zweck, auch wenn er dieselbe sprachliche Information enthält. Die Dauerhaftigkeit der Schriftform gegenüber der gesprochenen Sprache, die Kommunikation unabhängig von Zeit und Raum ermöglicht, und die Authentizität des Trägermediums, die durch Wasserzeichen, Stempel, Siegel, Barcode oder Ähnliches erreicht werden kann, machen das Wesen von Dokumenten in diesem ursprünglichen Sinne aus.

In der Brockhaus-Enzyklopädie in der siebzehnten Auflage von 1968 findet sich dagegen unter dem Stichwort „Dokument“:

neuerer
Dokumentenbegriff

„Als Dokumente können alle Unterlagen betrachtet werden, die Information beinhalten, also nicht nur publiziertes Wissen, sondern auch Briefe, Akten, Urkunden, Bildsammlungen, Filme u.v.a.“.

Die Dokumente neuerer Art sind also Informationsträger. Die Schriftform ist immer noch wesentlich, aber das Trägermedium und die Art, wie die Information auf das Trägermedium aufgebracht ist, sind nebensächlich. Auf den sprachlichen Ausdruck kommt es an.

Der Unterschied zwischen dem ursprünglichen und dem neueren Dokumentenbegriff kann an zwei Arten von Delikten klargemacht werden. Das Fälschen eines Personalausweises oder einer Banknote beinhaltet immer auch eine Fälschung des Trägermediums und betrifft Dokumente im ursprünglichen Sinne. Kopieren hingegen – sei es Fotokopieren oder Abschreiben – generiert immer ein zweites Exemplar des Dokuments, selbst wenn sich die Erscheinungsform dabei total ändert. Die Umstände, unter denen ein solches Kopieren zulässig ist, regelt das Urhebergesetz.

Das Gemeinsame an Dokumenten ursprünglicher und neuerer Art ist, dass sie durch ihre Schriftform eine ursprünglich sprachliche Mitteilung von der auditiven in die visuelle Übertragungsform und von der Flüchtigkeit der mündlichen Erzählung in die Dauerhaftigkeit des geschriebenen Wortes umsetzen. Somit werden die Grenzen von Zeit und Raum, die sonst der Senderin und dem Empfänger flüchtiger sprachlicher Mitteilungen gesetzt sind, überschritten.

Dokumente ursprünglicher und neuerer Art unterscheiden sich auch dadurch, wie viele Exemplare es von einem Dokument geben kann und wie viele Leserinnen und Leser gleichzeitig Gebrauch von so einem Dokument machen können.

Eine weitere und wichtige Unterscheidung, die zu einem dritten, dem modernen Begriff des digitalen Dokuments führt, hängt an der Idee vom Dokument als Informationsträger. Sprachlich formulierte Gedankengänge, Ideen, Fakten oder Aussagen werden erst dann zu Information, wenn sie von jemandem benötigt, aufgenommen und verarbeitet werden. Im folgenden Abschnitt, der auf Kuhlen [Kuh89] basiert, wird dies beleuchtet:

„Information ist Wissen in Aktion“ lautet eine Kurzformel der Informationswissenschaft, die sinngemäß von R. Kuh eingeführt wurde: „Information [...] as knowledge in action“.

Information ist Wissen
in Aktion

Er unterscheidet dabei die Konzepte von Information und Wissen dahingehend, dass Information als Teilmenge von Wissen aufgefasst wird. Weiterhin definiert er, dass Information zur Lösung eines konkreten Problems durch eine Person benötigt wird, aber nicht verfügbar ist. Dabei geht Kuhlen nicht auf den Grund für die Nicht-Verfügbarkeit der Information ein, im Gegensatz z. B. zu P. Wilson [Wil95], der erläutert, wie Informations-Überflutung zum „Nonuse“ von Information führt. Information als Wissen in Aktion bildet laut Kuhlen frei übersetzt das „pragmatische Prinzip der Informationsverarbeitung“ („pragmatic primacy of Information Work“).

Für die Informationsverarbeitung und das Design von Informationssystemen zieht er daraus die Schlussfolgerung, dass die Hauptaufgabe jeglicher Informationsverarbeitung darin besteht, Wissen in [nutzbare] Information zu transformieren und dabei Randbedingungen wie Zeitbedarf, Kosten, soziale Aspekte, kognitive Fähigkeiten und Ziele der jeweiligen Organisation zu berücksichtigen.

Weiterhin forderte Kuhlen, das pragmatic primacy of Information Work bei zukünftigen Entwicklungen von Informationssystemen ernst zu nehmen und zu berücksichtigen.

Mit Blick auf den Dokumentenbegriff bedeutet der Informations- und Wissensbegriff im Verständnis von Kuhlen, dass im Dokumentenbegriff Wissen beinhaltet ist. Dieses Wissen wird für einen Menschen erst in dem Moment zu Information, in dem er es z. B. für die Bearbeitung einer Aufgabe oder die Lösung eines Problems benötigt. Und auch das nur insoweit, als es nicht bereits Bestandteil seines menschlichen Wissens ist.

Dieses Bedürfnis nach Wissen wird in der Informationswissenschaft als Informationsbedürfnis bezeichnet. Das bedeutet weiterhin, dass es in der maschinellen Verarbeitung von Dokumenten nicht nur darauf ankommt, dort Wissen zu codieren, sondern dass es auch darauf ankommt, Dokumentinhalte nach Wissen durchsuchbar zu machen, welches für ein real existierendes menschliches Informationsbedürfnis relevant und damit Information im Sinne von Kuhlen ist.

Was bedeutet das konkret? In der Natur der Dinge liegt es dabei, zunächst an ein menschliches Gegenüber zu denken. Wie verhält es sich jedoch mit Computern, die ja dazu gedacht sind Daten, Dokumente und somit implizit in diesen enthaltenes Wissen, und darin wiederum ggf. enthaltene Information entgegenzunehmen, zu durchsuchen, weiterzuverarbeiten und anschließend verändert wieder auszugeben, wie dies ja bereits durch das EVA-Prinzip definiert wird.

Daten-, Dokumenten- und Informationsverarbeitung mithilfe von Computerprogrammen führt demnach unmittelbar zum modernen Dokumentenbegriff.

Wesentliche Tätigkeiten im Bereich der modernen Informationsverarbeitung werden heute durch Computersysteme unterstützt. Dies hat dazu beigetragen den modernen Dokumentenbegriff zu prägen. Unterstützt werden dabei Tätigkeiten und Anwendungen wie:

- Gedankengänge organisieren und Dokumente planen,
- Gedanken verbalisieren und Dokumente erfassen,
- Dokumente editieren, formatieren und publizieren, also der Öffentlichkeit zugänglich machen,
- Dokumente ablegen, verwalten, abfragen, zusammenfassen, kombinieren, transformieren, extrahieren,
- Dokumente lesen, verstehen, memorieren, aus ihnen lernen und über sie nachdenken.

der moderne
Dokumentenbegriff

Moderne Dokumente sind Informationsträger, die alle diese Anwendungsszenarien effizient unterstützen. Moderne Dokumente müssen deshalb in ihrer Urform elektronisch gespeichert sein und je nach Anwendungszweck unterschiedliche Präsentationsmedien unterstützen.

Selbsttestaufgabe 1.1:

Erläutern Sie, wie von einer Suchmaschine Kuhlens „Primacy of Information Work“ bei der Suche nach dem Stichwort „NASA Perseverance“ umgesetzt wird. Die Suche wird von einem Vater ausgeführt, um Fragen seiner Tochter Melissa zu beantworten.

Selbsttestaufgabe 1.2:

Überlegen Sie, ob auf die folgenden Dokumente eher der ursprüngliche, der neuere oder der moderne Dokumentenbegriff passt oder ob es sich vielleicht gar nicht um Dokumente handelt:

- Personalausweis
- Geldschein
- Gemälde
- Veranstaltungsplakat
- Privatbrief
- E-Mail-Nachricht
- Memo an die Geschäftsleitung
- Gedichtband
- Roman
- Bestellung
- Produktbeschreibung
- Fragebogen
- Nachricht in einem Protokoll zum digitalen Zahlungsverkehr (Electronic Banking)

Nicht alle Beispiele lassen sich eindeutig einordnen. Machen Sie sich klar, welche Argumente für oder gegen eine bestimmte Einordnung sprechen.

Vorangehend wurde eine Übersicht über die wesentlichen Kategorien von Dokumentenbegriffen gegeben. Im weiteren Verlauf werden in diesem Kurs jedoch nur noch Dokumente der modernen Art besprochen.

Dokumente sind also nachfolgend immer elektronisch gespeicherte Wissens- und ggf. auch Informationsträger, deren schriftsprachlich festgehaltene Inhalte sowohl von menschlichen Leserinnen und Lesern als auch von Computerprogrammen wahrgenommen und weiterverarbeitet werden können.

Im folgenden Abschnitt, der auf [BOB82] basiert, wird noch kurz der Aspekt des Informationsbedürfnisses und der Informationssuche angesprochen, bevor sich dann eine Analyse anschließt, welche Anforderungen sich daraus für Dokumentenformate ergeben und wie diese Anforderungen durch das Modell der strukturierten Dokumente erfüllt werden.

Verwandt zum Informations- und Wissenskonzept von Kuhlen ist das Konzept des Anomalous State of Knowledge (ASK) nach Belkin, welches folgendermaßen von ([BOB82], p. 62) definiert wird:

"The ASK hypothesis is that an information need arises from a recognized anomaly in the user's state of knowledge concerning some topic or situation and that, in general, the user is unable to specify precisely what is needed to resolve that anomaly."

Das bedeutet, unter einem ASK versteht „man einen „anormalen“ Wissenszustand, womit gemeint ist, dass einem Menschen zum Beispiel für die Bearbeitung einer Aufgabe oder zur Lösung eines Problems Wissen fehlt.

Unter einem „normalen“ Wissenszustand wird dabei verstanden, dass der Mensch das benötigte Wissen bereits besitzt. Insofern ist der ASK nichts anderes als die akademische Beschreibung eines Informationsbedürfnisses eines Menschen. Dies wird dann weiter so ausgeführt:

Ein ASK kann zu einem Informationsbedürfnis („Information Need“) führen, welches beispielsweise von [MRS08] wie folgt definiert wird:

„An information need is the topic about which the user desires to know more, and is differentiated from a query, which is what the user conveys to the computer in an attempt to communicate the information need“.

Das aus einem ASK resultierende Informationsbedürfnis stellt somit eine wichtige Anwendung der automatisierten Dokumentverarbeitung dar. Computersysteme, die durch automatische Dokumentverarbeitung Informationsbedürfnisse erfüllen können, indem sie relevantes Wissen in den Dokumenten suchen, heißen Informationssysteme. Umgangssprachlich werden sie oft als Suchmaschinen bezeichnet.

Diese implizite Beschreibung einer Interaktion eines Nutzers mit einer Suchmaschine während des Versuchs das ASK zu formulieren, wird als „Informational Behaviour“ bezeichnet. Laut Bates [Bat10, p. 2078] besteht Informational Behaviour aus der Interaktion von Nutzern mit Informationssystemen, der Frage, wie und wann sie Information suchen und was die Nutzer daraus machen.

Weiterhin stellt sie fest, dass der Vorgang des Suchens an sich ein interessantes Forschungsgebiet darstellt. In diesem Forschungsgebiet wurden daher verschiedene Suchstrategien („Information Seeking“, „Information Searching“) beschrieben. Im Rahmen einer Studie stellte Bates fest, dass die Strategien der Nutzer heuristisch und hochgradig interaktiv, gleichzeitig aber wenig geplant waren. Dabei unterscheidet sie zwei grundlegende Suchstrategien, nämlich „Lookup“ (Nutzung der Textsuche einer Suchmaschine) und „Examine“ (das Anzeigen von Titeln und Texten). Es wird dargelegt, dass Suchmaschinen neben Schlüsselwort-basierter Suche auch Browsing-orientierte Suchstrategien („Berrypicking“) unterstützen sollen. Die verschiedenen Modi der Informationssuche sind in Abbildung 4.1 zusammengefasst.

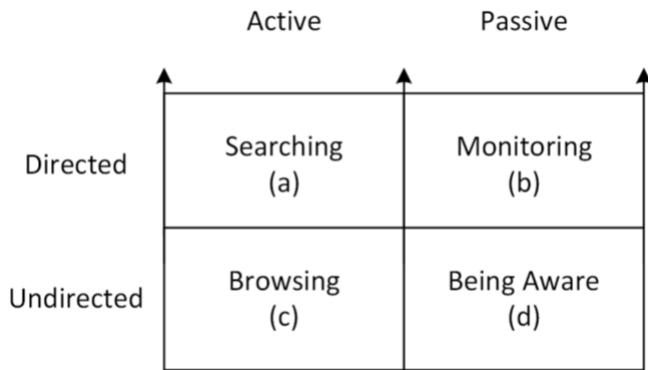


Abbildung 4.1 Modi der Informationssuche [Bat02]

Dabei wird zwischen aktiver und passiver, sowie gerichteter und ungerichteter Suche unterschieden. Entlang dieser Dimensionen unterscheidet Bates:

- direkte Suche als aktive Versuche, Fragen zu einem speziellen Thema zu beantworten und das Informationsbedürfnis zu befriedigen,
- Monitoring als aktiver Aufbau von Hintergrundwissen zu Interessengebieten und möglichen zukünftigen möglichen Fragestellungen,
- Browsing als das Gegenteil von Monitoring, um neues Wissen ohne konkretes Informationsbedürfnis durch aktive Auseinandersetzung mit neuen Inhalten zu erwerben,
- allgemeine Aufmerksamkeit (d) als ungerichtete und passive Form der Informationsbeschaffung. Laut Bates „gives us 80 percent of all we know“.

Da Dokumente Information und somit auch Wissen beinhalten, wird an dieser Stelle außerdem das sogenannte SECI-Modell betrachtet. Das SECI-Modell wurde von Nonaka eingeführt und später in den 90er Jahren von Takeuchi verfeinert [NT95]. Es beschreibt die Methoden, mit denen implizites Wissen in explizites Wissen umgewandelt wird und umgekehrt.

| SECI-Modell |

Das SECI-Modell kann wie folgt beschrieben werden:

1. Sozialisation (implizit zu implizit): Durch den Reifungsprozess sammelt jeder anderes Wissen. Auf diese Weise werden Fähigkeiten und Erfahrungen von Generation zu Generation weitergegeben. In der Vergangenheit, vor der Erfindung von Büchern, Fernsehen oder Internet, war der einzige Weg, Wissen zu übertragen, das Sozialisieren: Menschen kommunizieren direkt mit Menschen oder wenden den Wissensaustausch durch Erfahrungsaustausch an. Heutzutage werden viele fortschrittliche Technologien eingeführt, aber das Sozialisieren bleibt wichtig und kann nicht ersetzt werden.
2. Externalisierung (implizit bis explizit): Geselligkeit erfordert direkte Interak-

tion zwischen Menschen. Dies ist zeitaufwändig und nicht immer über eine lange Strecke möglich. Zusammen mit der Erfindung des Schreibens kann Wissen vom Gehirn auf andere Medien wie z. B. Bücher, Videos usw. Dieser Vorgang wird als Externalisierung bezeichnet.

3. Kombination (explizit zu explizit): Durch die Transformation von explizitem Wissen in neues explizites Wissen entsteht kein neues Wissen. Dieser Prozess besteht darin, das Wissen zu organisieren und Teile davon in neuen Formen zu kombinieren, damit es sinnvoller ist oder neue Erkenntnisse erklärt.
4. Internalisierung (explizit zu implizit): Personen können neues Wissen erwerben, indem sie explizites Wissen wahrnehmen, d. h.. verbrauchen, manchmal, indem sie wiederholt dieselbe Aktivität ausführen. Sie wandeln ihre Erfahrungen und Erinnerungen in mentale Modelle um. Einmal verinnerlicht, kann neues Wissen wiederverwendet, dokumentiert oder an andere Personen weitergegeben werden [Vu15]. Die folgende Abbildung zeigt das Konzept des Modells.

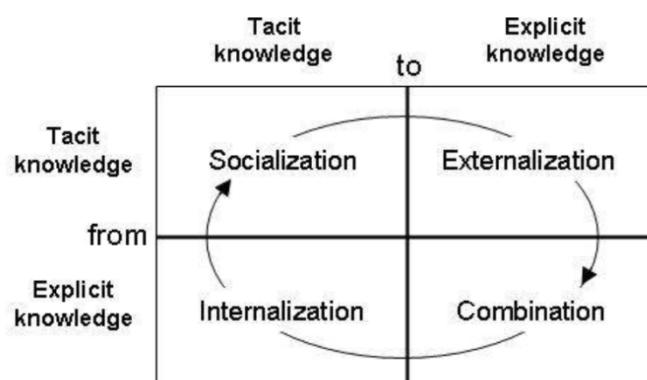


Abbildung 4.2 Das SECI Model [NT95]

Wissen existiert nicht von selbst, sondern wird aus Erfahrungen geschaffen und unter Verwendung verschiedener Arten von Medien oder durch direkte soziale Interaktion zwischen Menschen übertragen. Mit der Erfindung des Personal Computers erhalten Menschen Unterstützung von Computern beim Content Management und damit beim externalisierten Wissensmanagement. Infolgedessen wird dieser Prozess viel effizienter und effektiver als zuvor.

Moderne Textdokumente

Unter dem Begriff „Moderne Textdokumente“ werden Textdokumente verstanden, die in weltweit verteilten Informationssystemen wie dem Internet und dem World

Wide Web für die unterschiedlichen Nutzungsarten und Anwendungszwecke bereitgestellt werden. Einige Fragen die sich aus diesem Kontext ergeben sind:

- Welche Anforderungen ergeben sich für die Dokumentenformate?
- Welche Formate sind in der Praxis verfügbar?
- Welche Werkzeuge stehen zur Verfügung, um die Dokumente zu bearbeiten?
- Was können etablierende und etablierte Anwendungen wie Document Engineering, Content Management und Elektronisches Publizieren beitragen?

Zu Beginn der rasanten Entwicklung des WWW gab es nur ein relevantes Dokumentenformat im Web, nämlich die *Hypertext Markup Language*.

HTML

Die erste konsolidierte, und von der Internet Engineering Task Force verabschiedete, HTML-Version 2.0 umfasste ca. 150 Sprachelemente, wobei mit Sprachelementen sowohl die Elemente als auch die Attribute gemeint sind. Die besagte HTML-Version wurde im November 1995 als RFC 1866¹ eingeführt. Elektronisch arbeitende Produzenten und Verleger von Inhalten, sogenannte *Content Provider*, konnten mit einer Handvoll von HTML-Befehlen auskommen, um eine dem aktuellen Stand der Technik entsprechende Webseite zu erstellen.

IETF

RFC

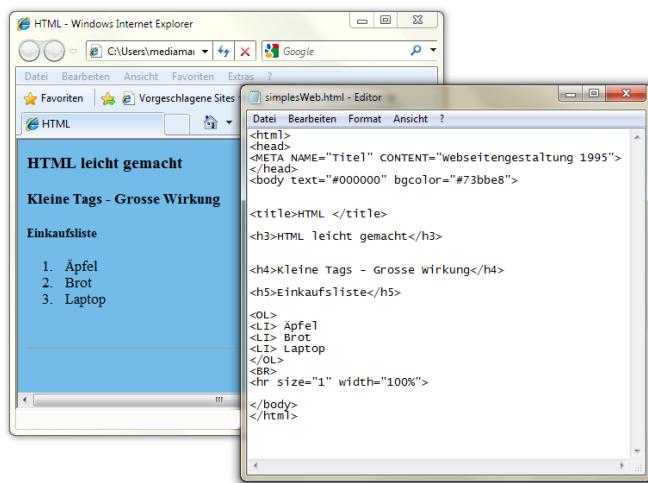


Abbildung 4.3 Webseitengestaltung von 1995

Die HTML-Version 4.0, welche am 24. April 1998 vom World Wide Web Consortium verabschiedet wurde, umfasste dann bereits ca. 1.600 Sprachelemente. An für Textdokumente relevanten Formaten sind zu HTML die Sprachen

W3C

¹ <https://www.rfc-editor.org/rfc/rfc1866>

- Extensible Markup Language,
- Cascading Style Sheets,
- Extensible Style Language,
- XML Schema,
- XML Linking Language,
- XML Path Language,
- XML Pointer Language

hinzugekommen.

Mittlerweile ist die fünfte Fassung von HTML aktuell (HTML5), welche das W3C am 28. Oktober 2014 als fertige Spezifikation vorgelegt hat. Diese ersetzt gleich mehrere Standards. Neben dem umfangreichen Vokabular wird u. a. ein klares Strukturmodell definiert, um modernen Textdokumenten gerecht zu werden. In Kurseinheit 2 wird ausführlicher auf HTML5 eingegangen. Nicht zuletzt ist HTML5 aktuell auch für die Unterstützung von Aspekten wie barrierefreie und mobile Nutzung des Internets zum Daten- und Dokumentenmanagement unverzichtbar.

Neben den statisch-passiven Textdokumenten, die klassischen, auf Papier gedruckten Dokumenten ähnlich sind, haben im Internet bereits seit längerem in zunehmendem Maße dynamisch-aktive Textdokumente an Bedeutung gewonnen. Sie enthalten aktive Komponenten, mit denen Leserinnen und Leser interagieren können. Es können sich auch der Inhalt und das Format dynamisch ändern. Beispiele für aktive Komponenten sind mathematische Formeln, die mittels eines Computer-Algebra-Pakets auf Knopfdruck evaluiert werden können. Weitere Beispiele für dynamische Dokumente sind sich regelmäßig aktualisierende Börsenkurse oder auf Knopfdruck expandierbare Übersichtsdarstellungen (engl. *outline*).

|W3C

Zahl, Umfang und Funktionsbereich der relevanten Empfehlungen des W3C und deren Implementierungen haben bis heute derart zugenommen, dass eine systematische Einführung in die Sprachen selbst und in darauf aufbauende erprobte Praktiken der professionellen und systematischen Dokumenterstellung und -verarbeitung einen organisierten Einstieg in die Internetpraxis der verteilten Informationsverarbeitung sehr erleichtern.

Die neuen Dokumentenformate für statisch-passive Dokumente im Internet folgen ebenfalls dem Modell der strukturierten Dokumente. Neben HTML hat XML für das Daten- und Dokumentenmanagement im Internet in den vergangenen Jahren immer mehr an Bedeutung gewonnen.

Dokumentenbestandteile

Dokumente moderner Art sind wie zuvor bereits beschrieben als Informationsträger zu sehen. Wie aber kommen die Leserinnen und Leser an die in einem Dokument enthaltene Information? Als Einstieg in Beantwortung dieser Frage kann zunächst einmal ein konkretes Dokument betrachtet werden: die Journalseite in Abschnitt 7.a im Anhang.

Es handelt sich hierbei um eine beispielhafte Annonce der Frauenbeauftragten der Technischen Universität München zu einem Computer-Propädeutikum. Bei der Lektüre dieses Dokumentes lässt sich feststellen, dass eine Reihe von Themen vorgeschlagen wurde, die in einem Kurs des Computer-Propädeutikums behandelt werden können. Wie genau aber wurde diese Information im Dokument kodiert, wie wird sie vom Leser wahrgenommen und gedanklich verarbeitet?

Die Ausgangsbasis, und damit die erste Ebene im Modell der strukturierten Dokumente, sind die in dem Dokument enthaltenen textuellen Inhalte, also die Buchstaben und Interpunktionszeichen, die beim Lesen vom Menschen wie auch von Maschinen zu Wörtern und Sätzen zusammengesetzt werden. Darüber hinaus werden diese von Menschen aber auch intellektuell verarbeitet und mit einer Bedeutung versehen.

Inhaltsebene des Dokumentes

Die Belegung von Symbolen mit Bedeutungen wird Semantik genannt. Der informationelle Inhalt des Dokuments wird also vom Leser erfasst. Dies ist eine Fähigkeit, die der Mensch den Maschinen zunächst voraushat. In der letzten Phase der Erfassung gibt es also wesentliche Unterschiede zwischen der Art und Weise wie ein Mensch und wie eine Maschine Inhalte wahrnimmt, deren informationellen Gehalt erfasst und verarbeitet.

Semantik

Es sind also zunächst nicht nur die Buchstaben und Interpunktionszeichen alleine, die den Erfassungsprozess ermöglichen. Zu dieser Einsicht verhilft folgender Gegen- test: Betrachtet ein menschlicher Leser die unformatierte Version des gleichen Dokumentes, die sich unter Abschnitt 7.b im Anhang befindet, und versucht, aus dieser Version die Gliederung des Konzepts herauszulesen, so stellt sich dies als mühsam bis unmöglich heraus.

Offenbar unterstützt also die Formatierung des Dokuments – etwa die Wahl der Schriften (groß oder klein, fett oder kursiv), die Ausrichtung von Textpassagen (linksbündig, rechtsbündig oder zentriert) und die Abstände zwischen einzelnen Textpassagen – die visuelle Aufnahme und Verarbeitung der Information durch den Menschen. Geht man zurück zur zuerst betrachteten, formatierten Version des Konzepts, so wird einem menschlichen Leser deshalb sofort die Gliederung in die Abschnitte „Hintergrund und Ziele“, „Das Kursangebot des Computer-Propädeuti-

kums“ usw. klar. Die visuelle Formatierung der Überschriften in Fettschrift und mit deutlichen Abständen zum umgebenden Text geben ausreichende Hinweise um die Gliederungselemente zu erkennen.

Die Buchstaben, Interpunktionszeichen und sonstigen Symbole, die in einem Dokument enthalten sind, zusammen mit dem Format, also der typografischen, sprich visuellen, Ausprägung der Symbole und der geometrischen Anordnung oder dem Layout, ermöglichen es also den menschlichen Leserinnen und Lesern, den Informationsgehalt eines Dokumentes leichter, d. h. kognitiv effizienter zu erfassen.

Was aber ist genau das Ergebnis dieses menschlichen Verarbeitungsprozesses? Zunächst einmal werden die Aussagen wörtlich erfasst und verstanden. Hinzu kommt jedoch auch das Erfassen der einzelnen Textpassagen und deren Rolle im Gesamttext. So identifiziert ein menschlicher Leser wahrscheinlich beim Betrachten der formatierten Version des Konzepts das Textstück „Ein Computer-Propädeutikum von Studentinnen für Studentinnen“ als Überschrift des gesamten Dokuments und das Textstück „Hintergrund und Ziele“ als Überschrift eines Abschnitts, der bis zur nächsten Überschrift reicht.

| logische Struktur Zusätzlich zur Aussage eines Dokuments erkennen menschliche Leserinnen und Leser also aufgrund des Inhalts und des Formats eine Strukturierung in logische Einheiten. In dem Konzept zum Computer-Propädeutikum ließen sich beispielsweise die folgenden Strukturelemente identifizieren: Briefkopf mit Namen der Autoren und Angaben zur Institution, Hauptüberschrift, Nebenüberschrift mit Datums- und Versionsangaben, Abschnitte mit Überschriften, Absätzen und Listen, Geldbeträge, Namen der Organisatorinnen und Datumsangaben.

Die von den Lesern erkannte Struktur ist nicht eindeutig. Hinweise auf die Struktur ergeben sich aus Inhalt und Format aufgrund von dem semantischen Verständnis der Textpassagen und der kulturellen, durch den Umgang mit zahlreichen Dokumenten ähnlicher Art gewonnenen Erfahrung.

Für menschliche Leser stellt sich also der Prozess der Informationsaufnahme und -verarbeitung folgendermaßen dar: Das Dokument wird mit Inhalt und Format präsentiert. Menschliche Leserinnen und Leser extrahieren daraus die Aussage des Dokumentes und konstruieren eine Struktur für die einzelnen Textelemente. Durch visuelle Wahrnehmung und gedankliche Arbeit unter Rückgriff auf das Sprachverständnis und das kulturelle Wissen wird die Bedeutung der Aussage interpretiert. Die Verwendung von Formatmerkmalen, Aussage und Struktur bilden die Grundlage für die weitere intellektuelle Informationsverarbeitung.

Ein Dokument kann in ganz unterschiedlichen grafischen Aufmachungen dargestellt werden. Die Darstellung hängt auch vom Präsentationsmedium und der Lesesituati-

on ab. Beispielsweise kann ein Dokument für das ermüdfreie Lesen am Bildschirm mit einer anderen Farbpalette und mit anderen Schriften formatiert sein als für das Lesen von einem Papierausdruck; oder Querverweise können für die Onlineversion als Hypertextlinks und für die Druckversion als Hinweise auf Seitenzahlen realisiert werden.

Entscheidend für die „Identität“ eines Dokumentes ist, dass aus den verschiedenen Formaten immer noch dieselbe Aussage und im Wesentlichen dieselbe Struktur erkannt wird.

Selbsttestaufgabe 1.3:

Welche Methoden, mit denen implizites Wissen in explizites Wissen umgewandelt werden kann, beschreibt das SECI-Modell?

Dokumente strukturieren

Bisher wurde das Erfassen der in einem Dokument enthaltenen Information von der Warte der menschlichen Leser aus betrachtet. Was ist aber, wenn Computerprogramme zur Informationsverarbeitung herangezogen werden sollen? Wollte man den menschlichen Prozess des Textverständnisses nachbilden, der bei einer formatierten und für das menschliche Wahrnehmungssystem „Auge“ realisierten Version des Dokuments startet, so müsste man Computerprogramme mit intelligenten Fähigkeiten ausstatten, die Zeichenerkennung, Sprachverständnis und Strukturerkennung einschließen. Von diesen Möglichkeiten ist man heute jedoch noch relativ weit entfernt, sie sind allerdings Gegenstand intensiver Forschung.

Der Ansatz des Modells der strukturierten Dokumente sieht neben der ersten Ebene des rein textuellen Inhaltes folgende zwei weitere Ausgangspunkte für die computergestützte Informationsverarbeitung vor: zunächst eine explizite Repräsentation der Struktur des Dokumentes (zweite Ebene) im Sinne einer strukturellen Repräsentation. D. h., Autoren eines strukturierten Dokumentes editieren den eigentlichen Inhalt des Dokumentes und markieren die notwendigen Strukturelemente.

Strukturebene des Dokumentes

Daraufhin folgt die dritte Ebene im Modell der strukturierten Ebene, die sich auf die Formatierung der bereits strukturierten Inhalte bezieht. Das zu den Textelementen passende Format kann aus einer separaten, vom Dokument unabhängigen und bei Bedarf auswechselbaren Formatvorlage, auch Stylesheet genannt, berechnet werden.

Formatebene des Dokumentes

Das Stylesheet enthält allgemeine Regeln, wie die einzelnen Strukturelemente zu formatieren sind. Durch eine Regel kann beispielsweise festgelegt werden, dass

Überschriften in einer fetten Kursivschrift mit einem gewissen Abstand zum umgebenden Text formatiert werden sollen.

Ein sogenannter *Formatierer* bringt das mit Strukturmarkierungen versehene Quell-dokument mit dem Stylesheet zusammen und formatiert das Dokument entsprechend der Vorgaben in dem Stylesheet.

Vorteile des Dokumentenmodells

Ein solches Vorgehen hat als entscheidenden Vorteil die Flexibilität, mit welcher die Formatierung der Texte immer wieder geändert werden kann. Die logische Strukturierung unterstützt aber auch Anwendungen der Dokumentenverwaltung, sodass etwa die automatische Generierung von Katalogeinträgen und Verzeichnissen möglich wird oder das Wiedergewinnen, sprich das Wiederfinden und Zugreifen von Information nach bestimmten logischen Kriterien. Die eigentliche Informationsverarbeitung kann sich also direkt auf die logische Struktur stützen. Die Unabhängigkeit der Stylesheets von den Dokumenten ermöglicht, dass ein Dokument durch Kombination mit verschiedenen Stylesheets passend zum Anwendungszweck formatiert werden kann. Außerdem können verschiedene Dokumente durch Kombination mit ein- und demselben Stylesheet einheitlich formatiert werden. Auch lässt sich die Struktur beim Erstellen und Lesen von Dokumenten nutzen, etwa zur Navigation und zur Bereitstellung von Übersichten. Von diesen Vorteilen profitieren besonders die Anwendungen, die große Dokumentenbestände erstellen und verwalten müssen.

Eine umfangreiche Sammlung von Dokumenten mit Projektvorschlägen, die alle nach dem gleichen Muster logisch strukturiert und markiert sind, kann so z. B. systematisch nach anwendungsnahen Kriterien abgefragt werden. Auch etwa die im letzten halben Jahr neu hinzugekommenen Vorschläge können mit Titel und Datum ausgegeben werden. Eine weitere Anwendung wäre, automatisch nach allen Dokumenten mit Projekten zu suchen, die ein bestimmtes gemeinsames Ziel verfolgen. Auch für die Verwaltung des Dokumentenbestandes bietet das Modell der strukturierten Dokumente Vorteile. Die mit der Verwaltung befassten Personen brauchen für den gesamten Bestand nur einmal die zu verwendenden Strukturelemente festzulegen. Für jedes Präsentationsmedium ist nur ein einziges Stylesheet nötig, welches dann auf alle Dokumente des Bestandes anwendbar ist. Für das Beispiel der Projektvorschläge würden zwei Stylesheets ausreichen, eines für das Lesen am Bildschirm und eines für das Steuern eines Ausdrucks.

Das Modell der strukturierten Dokumente wurde in den achtziger Jahren entwickelt [AFQ89][FHVV11][MvD82]; seine Wurzeln gehen sogar bis in die sechziger Jahre zurück [E63]. Alle modernen Textverarbeitungssysteme unterstützen das Modell zumindest in Ansätzen.

Verfeinerungen des Dokumentenmodells

Zum vollen Modell der strukturierten Dokumente gehört wesentlich dazu, dass die Namen der Strukturelemente frei wählbar sind. Wenn man beispielsweise ein Aufgabenblatt logisch strukturieren möchte, dann müssen die Strukturelemente wie Aufgabe, Abgabedatum, Punktezahl, Lösungshinweis oder Schwierigkeitsgrad angegeben werden können. Ist der Sprachvorrat, aus dem Strukturelemente gewählt werden können, fest vorgegeben und beschränkt, kann er nicht alle sich im Laufe der Zeit ergebenden Anwendungen abdecken. Dies verwässert allerdings die Vorteile der strukturierten Dokumente für die computergestützte Informationsverarbeitung, da semantische Information über die Rollen der Strukturelemente durch eventuelle Mehrfachbelegung verloren geht.

freie und vordefinierte
Strukturelemente

Eine weitere Überlegung zum Modell der strukturierten Dokumente ist, ob der zugrundeliegende Vorrat an Strukturelementen formal definiert werden soll. Eine Strukturvorgabe ist eine mit formalen Methoden festgelegte Definition des Vorrats an Strukturelementen, die auch Einschränkungen zur Verwendung der Elemente beinhalten kann. Eine Strukturvorgabe hat Vorteile. Beispielsweise können Dokumente automatisch auf ihre strukturelle Korrektheit überprüft werden oder Editierhilfen können solche Vorgaben auswerten und zur Unterstützung der Autorin oder des Autors einsetzen. Schließlich wird durch eine formale Strukturvorgabe auch das Vokabular einer Dokumentenanwendung dokumentiert.

Strukturdarstellungen

Zur Markierung der Strukturelemente gibt es viele verschiedene Konzepte. Bei den meisten Konzepten wird davon ausgegangen, dass die logische Strukturierung streng hierarchisch angelegt ist. Strukturelemente können also andere Strukturelemente vollständig enthalten, sich aber nicht mit ihnen überlappen. Die Strukturelemente, die zum Beispiel in der Annonce zu dem Computer-Propädeutikum in Abbildung unter Abschnitt 7.c eingeführt worden sind, gehorchen dem Gesetz der hierarchischen Schachtelung.

Häufig wird die hierarchische Struktur auch grafisch dargestellt. Hierzu können exemplarisch zunächst drei Varianten vorgestellt werden. Abbildungen zu allen drei Varianten befinden sich im Anhang.

Bei der ersten Variante werden ineinander geschachtelte Rechtecke oder Boxen, die mit dem Namen des jeweiligen Elementes gekennzeichnet sind, genutzt. Die zweite – dynamische – Variante, bei der mit Einrückungen und Ikonen gearbeitet wird, könnte aus Dateibrowsern bereits bekannt sein. Eine dritte – ebenfalls dynamische – Variante besteht darin, die logische Struktur als einen Baum darzustellen. Die grafischen Möglichkeiten stellen hohe Anforderungen an die Formatierungsmöglichkeiten.

ten des darstellenden und ja auch möglichst noch Interaktionen zulassenden Systems.

Tags

Eine nicht so anspruchsvolle Möglichkeit besteht darin, die hierarchische Struktur mittels einer sogenannten Markierung (engl. *tag*) im Text zu versehen und damit zu serialisieren. Zu diesem Zweck wird jeder Textbereich, der Inhalt eines logischen Elementes ist, an seinem Anfang und an seinem Ende mit einer öffnenden und einer schließenden Markierung versehen, die den Namen des entsprechenden Strukturelementes enthält. Eine solche Markierung ist einfach ein sogenannter Klartext, der sich durch syntaktische Konventionen vom eigentlichen textuellen Inhalt abhebt. Der textuelle Inhalt ist ansonsten völlig unformatiert. Bei der XML-Konvention wird der Anfang einer Markierung durch einen Namen, der in spitzen Klammern steht, signalisiert (z. B. <Titel>). Soll das Strukturelement mit einer Markierung am Ende quasi abgeschlossen werden, also die strukturelle Markierung danach nicht mehr gelten, wird zusätzlich ein Schrägstrich in die Markierung eingesetzt (z. B. </Titel>). Der Oberbegriff zu Markierungen und eventuellen weiteren Möglichkeiten, nicht zum Inhalt zählende Zeichen in einen Text einzustreuen, ist der Begriff Auszeichnung (engl. *markup*). Eine grafische Verfeinerung dieser Strukturrepräsentation durch Auszeichnungen wird erreicht, wenn mit Einrückungen und mit typografischer Differenzierung von Text und Auszeichnung gearbeitet wird.

Da die Auszeichnungen mit im Text enthalten sind, wenn auch über syntaktische Konventionen vom eigentlichen Inhalt getrennt, spricht man auch von einer eingebetteten Auszeichnung (engl. *embedded markup*). Von einem prinzipiellen Standpunkt aus gesehen sind alle Darstellungsformen für strukturierte Dokumente äquivalent. Die Repräsentation der hierarchischen Struktur durch eingebetteten Markup ist technisch am einfachsten zu handhaben und ist deshalb derzeit noch die gängigste. Sie hat den Vorteil, dass Dokumente mit eingebettetem Markup mit jedem einfachen ASCII-Texteditor angesehen und bearbeitet werden können. Die Darstellungsweise ist allerdings unübersichtlich und nicht sehr eingängig. Die anderen Darstellungsformen stoßen an ihre Grenzen, wenn die Schachtelungstiefe sehr groß wird oder die Granularität der Struktur sehr fein. Probleme tauchen unter anderem schnell bei der logischen Beschreibung mathematischer Formeln auf.

eingebettete Auszeichnungen

Diskussion des Dokumentenmodells

Das Modell der strukturierten Dokumente wird eingesetzt, um Dokumente für die Computerverarbeitung vorzubereiten. Die zentrale Idee ist, die Kerndaten des Dokuments redundanzfrei in einer Art Reinform vorzuhalten, wie man es auch beim Design relationaler Faktendatenbanken mit Normalformen verfolgt. Faktendaten, die sich aus dieser Quellform berechnen lassen oder die variabel gehalten werden sollen, werden nicht in der Quellform mit kodiert sondern bei Bedarf berechnet und

gegebenenfalls als Sichten generiert. Verarbeitungsfunktionen können auf dieser Reinform der Daten aufsetzen und werden nicht belastet durch Idiosynkrasien im Datenbestand, die durch einzelne Anwendungen begründet sind.

Im Datenbankbereich dient diese Methode gleichzeitig der logischen Datenunabhängigkeit. Änderungen an der Struktur der Quelldaten können vor Endsystemen, die über Sichten mit der Datenbasis zu tun haben, verborgen bleiben. Das Potential der Datenunabhängigkeit ist durch das Modell der strukturierten Dokumente und seine Umsetzung mit XML auch im Bereich der Webdokumente gegeben. In diesem Bereich sind noch interessante Forschungsarbeiten zu erledigen. Auch eine Theorie der Normalformen, wie sie im Datenbankbereich etabliert ist, gibt es für den Bereich der strukturierten Dokumente noch nicht. Dennoch, gerade die Forschung in diesem Bereich und aktuelle Entwicklungen im Bereich der dokumenten- und XML-basierten Datenbanken (z. B. BaseX, MongoDB etc.) sowie die Integration spezieller Datentypen und Indizes für Informationssysteme verschiedenster Art und Methoden zur Verarbeitung in relationalen Datenbankmanagementsystemen (z. B. Oracle XML DB) versprechen einiges für die Zukunft.

Das Modell der strukturierten Dokumente dient somit dazu, Inhalt und Struktur eines Dokumentes, auf denen die automatische Verarbeitung basieren soll, zu repräsentieren. Dabei ist es wichtig, die Strukturelemente je nach Anwendungsfall frei definieren und die entsprechenden Definitionen auch formal festzuhalten zu können. Auf dieser Basis können Verarbeitungsfunktionen aufgesetzt und Sichten generiert werden, wobei die Basis je nach Anwendungsfall durch zusätzliche und separate Steuerungsdaten ergänzt werden kann.

Historisch stammt das Modell aus dem Bereich der Dokumentenverarbeitung und Dokumentenverwaltung (Dokumentenmanagement, engl. *document management*). Ein wichtiger Anwendungsfall in diesem Bereich ist die Formatierung. Vorgaben für die Formatierung werden in separaten und auswechselbaren Formatierungsvorschriften (engl. *stylesheets*) vorgehalten und Formate aus beiden Beschreibungen zusammen berechnet. Über die Generierung formatierter Sichten hinaus sind Systeme zur Unterstützung des Dokumentenmanagements typischerweise in der Lage, Nummerierungen, Verzeichnisse und Querverweise aus der expliziten Repräsentation von Inhalt und Struktur zu erzeugen. Dies geschieht unter Hinzunahme von oft ebenfalls in Stylesheets enthaltenen Steuerungsdaten.

Dokumenten-
management

Das Modell unterstützt jedoch nicht nur Anwendungsfälle des Dokumentenmanagements im engeren Sinne, sondern trägt auch Anwendungen im weiteren Bereich des Informations- und Wissensmanagements Rechnung. Das Modell der strukturierten Dokumente dient somit als Werkzeug, um Texte zu gut zu verwaltenden Texten weiterzuentwickeln und die maschinelle Lesbarkeit und automatisierte Verarbeitung zu ermöglichen.

Informations- und
Wissensmanagement

Zudem sollen Bereiche im Inhalt eines Dokuments mit zusätzlicher semantischer Information markiert und die Bedeutung oder Rolle des entsprechenden Textstücks somit durch eine Annotation repräsentiert werden. Das Modell platziert sich damit in einem Spektrum, das von einer rein grafischen Repräsentation eines Dokuments, etwa als Pixelmuster oder als PostScript-Dokument, bis zu einer expliziten Repräsentation der Semantik, etwa als semantisches Netz, reicht.

Was in diesem Kurs Dokumentenmodell genannt wird, heißt in der älteren Literatur auch manchmal Dokumentenarchitektur. Der Begriff der Architektur macht klarer deutlich, worum es in einer ersten Annäherung an Dokumente und ihre Formate geht: relevante Aspekte von Dokumenten sollen identifizierbar und das Zusammenwirken transparent gemacht werden. Es werden im weiteren Verlauf des Kurses auch Überlegungen dazu angestellt werden, welche abstrakten Modelle den sich im Internet manifestierenden Formaten zugrunde liegen. Es steht dabei nicht immer für jede Ebene der Modellierung eine separate Begrifflichkeit – etwa wie hier „Architektur“ – zur Verfügung. Bei dem noch folgenden Modell für die ZeichenCodierung wird jedoch sehr schnell klar, warum in diesem Kurs der Begriff Modell eingeführt wird und wir die Thematik nicht unter dem Begriff Architektur der strukturierten Dokumente erläutern.

Umsetzungen des Dokumentenmodells

Anhand der meisten modernen Textverarbeitungssysteme wird das Modell der strukturierten Dokumente in irgendeiner Form umgesetzt. Die beiden Systeme LaTeX und MS-Word decken vom Typ her zusammen das Spektrum von Textsystemen ab, die in akademischen und technischen Dokumentierungen sowie im Büro zu finden sind. Die folgende Einführung der beiden Beispielsysteme soll dabei aufzeigen, in welcher Weise Verarbeitungssoftware für Dokumente heute das Modell der strukturierten Dokumente erfüllt.

Die Diskussion soll darüber hinaus auch dafür sensibilisieren, bei der Erstellung von neuen eigenen Dokumenten die angebotenen Mechanismen eines Dokumentensystems zur Unterstützung des Modells der strukturierten Dokumente möglichst weitgehend auszunutzen. Dies ist im Interesse eines effizienteren Dokumentenmanagements und einer flexibleren Weiterverwendung der daraus resultierenden Dokument- und Datensammlungen.

Umsetzung des Dokumentenmodells mit LaTeX

Das Beispiel des Konzeptes zum Computer-Propädeutikum befindet sich im LaTeX-Format im Anhang unter 7.e. Der textliche Inhalt ist gleich dem in der Anzeige unter 7.a und dem im XML-Format unter 7.d. Es ist auch eine gewisse Übereinstimmung bei den markierten Strukturen festzustellen. Die in der LaTeX-Version explizit markierten logischen Elemente sind die Angaben über die Autorinnen \author{...},

die Hauptüberschrift `\title{...}`, die Abschnittüberschriften `\section{...}`, die Liste `\begin{itemize} ... \end{itemize}`, die Aufzählungspunkte `\item`, das Dokument als Ganzes `\begin{document} ... \end{document}`, das Datum `\Datum` und die Versionsnummer `\Version{...}`.

Die syntaktischen Formen des Markup in LaTeX sind vielfältiger als in XML, auch wenn es gewisse Ähnlichkeiten gibt. Eine direkte Entsprechung zu XMLs Konstrukt `<xxx>...</xxx>`, um ein logisches Element `xxx` zu markieren, ist `\begin{xxx} ... \end{xxx}`, aber es existieren auch die Formen `\xxx{...}` und sogar `\xxx ...` ohne explizite Markierung des Elementes.

Die Funktionsbedeutung ergibt sich aus der Anfangsmarkierung eines Elements zusammen mit der Metainformation, ob dieses nächste Element im Element `XXX` enthalten sein kann. Einen Sonderfall stellt die implizite Abgrenzung von Absätzen durch Leerzeilen dar.

Das zur Verfügung stehende Repertoire von logischen Elementen ist aus der grundlegenden Dokumentenklasse `\documentclass{...}` und den hinzugeladenen Paketen `\usepackage[...]{...}` ersichtlich. Hier werden externe Dateien referenziert, die systemweit zur Verfügung stehen können. Es werden die Sprachelemente definiert, die im Dokument vorkommen. Am Anfang der LaTeX-Version befinden sich außerdem auch intern definierte Elemente, nämlich `\Datum` und `\Version`.

Bei LaTeX ist also vorgesehen, dass die Autoren ihr eigenes Vokabular von Elementen definieren und dass eine Gruppe von Autoren ein gemeinsames Vokabular benutzt. Dieses wird dann nur einmal definiert und kann von verschiedenen Dokumenten aus referenziert werden. Das Vokabular selbst ist formal definiert und weitere syntaktische Einschränkungen sind implizit.

Mit LaTeX wird somit die erste Anforderung des Modells der strukturierten Dokumente erfüllt: Der Inhalt und die logische Struktur eines Dokumentes werden explizit repräsentiert. Die Strukturelemente können je nach Anwendungsfall frei definiert werden. Ansatzweise ist es auch möglich, die Strukturelemente formal festzulegen.

Wie sieht es mit der zweiten Anforderung aus, dass Vorgaben für die Formatierung in separaten und auswechselbaren Stylesheets vorzuhalten sind? Hier kommen wieder die Konstrukte `\documentclass{...}` und `\usepackage[...]{...}` ins Spiel, die schon bei der Definition der Strukturelemente herangezogen worden sind. Die hier referenzierten externen Dateien erfüllen demnach eine Doppelfunktion: Die Festlegung darauf, welche Strukturelemente im Dokument vorkommen können und die Vorgabe, wie die Strukturelemente formatiert werden sollen.

Man kann diese Doppelfunktion nachvollziehen, wenn man sich die interne Definition von \Datum und \Version anschauen. Die Definition von \Version besagt beispielsweise, dass das nach der Markierung in geschweifte Klammern eingeschlossene Argument, also die Versionsnummer, fett gesetzt werden soll: `fett=boldface` und wird abgekürzt zu `bf`. Die Definition von \Datum impliziert, dass die Systemfunktion \today aufgerufen wird, die dann das aktuelle Datum einsetzt.

Die Forderung nach Stylesheets ist erfüllt. Im Falle von LaTeX sind diese extern und somit auswechselbar oder intern und an das Dokument gebunden.

Die Schwachstellen, die es in der Umsetzung des Modells der strukturierten Dokumente bei LaTeX gibt, werden bei einer genaueren Analyse offensichtlich. Die Vorgehensweise, mit der bei LaTeX die logische Markierung mit der Formatvorgabe verbunden wird, und wie das Format selber definiert ist, weist einige Schwächen auf. In LaTeX wird die Markierung als Kommando einer sogenannten Makroprogrammiersprache behandelt und die Formatdefinition als Programm in dieser Programmiersprache. Mit dieser Makroprogrammierung kann mehr erreicht werden als nur die Formatierung festzulegen. So speichern die Programme für \author und \title ihre Argumente nur intern. Erst das Kommando \maketitle im Eingabedokument, das gar keine offensichtliche logische Funktion hat, platziert die entsprechenden Angaben auf der Seite und legt ihre grafische Erscheinungsform fest. Im Extremfall können Makroprogramme sogar andere Kommandos umdefinieren. Beispielsweise könnten die Leerzeilen, die Absatzgrenzen markieren, so umdefiniert werden, dass das jeweils erste Wort eines Absatzes als Schlüsselwort fett gedruckt wird. Ebenso bewirkt das Klammerpaar \begin{verbatim} und \end{verbatim}, dass da zwischen eingeschlossenes Markup im Klartext ausgegeben und nicht interpretiert wird.

In einer reinen Markupsprache werden nur die Markierungen bereitgestellt, die im Dokument verwendet werden. Bei LaTeX wird die Funktion von Markupsprache, Formatdefinition und Programmierung untrennbar aneinander gekoppelt. Dies muss nicht unbedingt problematisch sein – allerdings widerspricht es dem allgemeinen und wohlbegündeten Informatik-Prinzip, der Trennung von konzeptuellen Dimensionen. In der Praxis bedeutet das, dass eine Person, die ein Stylefile für LaTeX erstellen möchte, über ein ganzes Spektrum von Fähigkeiten und Fertigkeiten verfügen muss. Die gewünschte logische Struktur muss geplant und organisiert werden, das grafische Design muss entworfen und die so entwickelten Vorstellungen sollten dann auch durch Computerprogramme realisiert werden. Die Einstiegshürde für einen kreativen Gebrauch von LaTeX ist also hoch. Im Gegenzug ist LaTeX durch seine Programmierbarkeit ein mächtiges und in seiner Funktionalität bei Bedarf beträchtlich erweiterbares System.

LaTeX ist ursprünglich als Textsatzsystem entwickelt worden. Die Funktionen beschränken sich also im Wesentlichen auf den Bereich des Dokumentenmanagements. Da jedoch bei LaTeX die Technik des eingebetteten Markups zur Repräsentation von Strukturelementen zur Verfügung steht, können mithilfe von weiteren Programmen sowie entsprechenden Schnittstellen die LaTeX-Quelldokumente als Text eingelesen und automatisch weiterverarbeitet werden.

Die LaTeX-Software ist frei verfügbar. Die deutsche TeX Users Group DANTE e.V. stellt LaTeX für verschiedene Plattformen unter <http://www.dante.de> zur Verfügung.

Zusammenfassend lässt sich also feststellen, dass LaTeX das Modell der strukturierten Dokumente weitgehend erfüllt. Die Markupsprache besteht jedoch nicht aus einfach deklarativen Markierungen, sondern ist eine kommandoorientierte Textsprache und in wesentlichen Teilen sogar quasi eine Programmiersprache für diesen Zweck.

Umsetzung des Dokumentenmodells mit Microsoft Word

Das System Microsoft Word (ursprüngliche Version vor OpenOffice-XML-Standard), das nun näher betrachtet wird, steht LaTeX in seinen grundlegenden Eigenschaften fast diametral entgegen. Die Einfachheit der Benutzung als Bürosystem ist oberstes Ziel, für das Abstriche bei der Mächtigkeit, Flexibilität und Erweiterbarkeit in Kauf genommen werden.

Ein Word-Dokument wird in einer bereits formatierten Sicht editiert. Markup wird in der Grundeinstellung nicht angezeigt. Die zugeordnete Formatierung wird gleich umgesetzt und am Bildschirm angezeigt. Das sogenannte Whatyou-SeeisWhatYouGet-Prinzip (WYSIWYG-Prinzip) ist somit so gut wie perfekt umgesetzt wie die Abbildung unter 7.a zeigt.

What you see is what
you get

Der Anhang enthält bei Abbildung 7.2 ein Bildschirmfoto des Beispiel-Dokumentes mit struktureller Unterteilung. Wer ein Microsoft Word-System installiert hat, kann vielleicht nachvollziehen, welche Strukturelemente im vorliegenden Beispiel verwendet wurden: Autorin, Titel, Hauptüberschrift, Nebenüberschrift, Datum, Version, Abschnittüberschrift, Absatz und Aufzählungspunkt.

Autoren können mit Word Zeichenketten oder ganzen Absätzen eine Formatierung zuweisen. Die unterschiedlichen Formatierungen werden als Satz von Formatier-Parametern mit den jeweiligen Werten unter einem individuellen Bezeichner abgespeichert.

Vom Modell der strukturierten Dokumente aus gesehen entsprechen die Namen der Formatvorlagen dem Vokabular an Strukturelementen, die Formatvorlage selbst entspricht einer Regel im Stylesheet und die Zuweisungen der Formatvorlagen an die

Absätze und Zeichenketten im Dokument entsprechen der Markierung mit logischen Strukturelementen. Mit Word wird damit die explizite Repräsentation von Inhalt und logischer Struktur eines Dokumentes ermöglicht. Die Strukturelemente können je nach Anwendungsfall frei definiert werden und ansatzweise ist es möglich, die Strukturelemente formal festzulegen.

Word und Stylesheets

Darüber hinaus ist es in Word möglich, in sogenannten Musterdokumenten (engl. *document templates*), die quasi als Stylesheets fungieren, Sammlungen von Formatvorlagen anzulegen, sie zu verwalten und sie Dokumenten zuzuordnen. Stylesheets in Word sind auf diese Weise auswechselbar.

Das Word-Dateiformat auf Basis von XML ist jetzt bereits seit einiger Zeit das annähernd standardkonforme Format und zugleich entsteht ein ZIP-Archiv. Wenn ein solches Archiv geöffnet oder extrahiert wird, findet man eine mehrstufige Hierarchie von Komponenten, die über entsprechende Auflistungen zusammengehalten werden. In Form sogenannter *<Relationship>*-Elemente werden die Beziehungen der Komponenten aufgezeigt und befinden sich im Ordner „rels“. Einzelne Komponenten können ausgetauscht werden ohne auf die Office-Anwendung zurückzugreifen. Der Kern des Dokuments ist die Datei „document.xml“, die Textdaten sowie Format- und Zeichensatzeinstellungen enthält. Diese Datei ist der einzige erforderliche Teil. Falls Kopf- und Fußzeilen vorhanden sind, findet man diese in separaten Dateien: „header.xml“ bzw. „footer.xml“. Die Datei „fontTable.xml“ enthält die Liste der Zeichensätze (engl. *fonts*), in der Datei „settings.xml“ befinden sich die anwendungsspezifischen Einstellungen und in der Datei „styles.xml“ sind die verwendeten Formate. Bilder oder andere Mediendateien werden im Ordner „media“ binär abgelegt. Mit jeder neuen Word-Version wird das Modell der strukturierten Dokumente somit besser umgesetzt. Die aktuelle Version ist Word 2021.

Selbsttestaufgabe 1.4:

Wie unterscheiden sich die im Kurstext beschriebenen Konzepte zur grafischen Markierung der Strukturelemente eines Dokuments gegenüber der Markierung durch Auszeichnung?

Selbsttestaufgabe 1.5:

Beschreiben Sie die drei Ebenen des Modells der strukturierten Dokumente, die im Kurstext genannt werden, in eigenen Worten.

5. Zeichencodierungen

Der Inhalt ist ein grundlegender Bestandteil eines strukturierten Dokuments. Die Repräsentation im Computer, also die kodierte Datei, spielt somit auch eine wichtige Rolle. Es wird dabei davon ausgegangen, dass der Inhalt aus sogenanntem glatten Text besteht. Dies ist eine Folge von elementaren Texteinheiten wie Buchstaben, Ziffern, Interpunktionszeichen und anderen Zeichen. Diese Zeichen müssen letztendlich in Bits und Bytes kodiert werden. Eine ZeichenCodierung gibt dazu eine Vorschrift an.

Die bekannteste ZeichenCodierung ist US-ASCII, standardisiert als ANSI X3.4 im Jahre 1968. Sie ist eine Variante von ISO 646-US und somit auch von ISO 646 aus dem Jahre 1972. Der Zeichensatz von US-ASCII in Abbildung 5.1 reicht lediglich aus, um lateinische und US-amerikanische Texte sowie Texte in einigen wenigen weiteren Sprachen zu codieren.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|
| 20 | 21 | ! | 22 | , | 23 | # | 24 | \$ | 25 | % | 26 | & | 27 | , | 28 | (| 29 |) | 2A | * | 2B | + | 2C | , | 2D | - | 2E | . | 2F | / | |
| 30 | 0 | 31 | 1 | 32 | 2 | 33 | 34 | 4 | 35 | 5 | 36 | 6 | 37 | 7 | 38 | 8 | 39 | 9 | 3A | : | 3B | ; | 3C | < | 3D | = | 3E | > | 3F | ? | |
| 40 | @ | 41 | A | 42 | B | 43 | C | 44 | D | 45 | E | 46 | F | 47 | G | 48 | H | 49 | I | 4A | J | 4B | K | 4C | L | 4D | M | 4E | N | 4F | O |
| 50 | P | 51 | Q | 52 | R | 53 | S | 54 | T | 55 | U | 56 | V | 57 | W | 58 | X | 59 | Y | 5A | Z | 5B | [| 5C | \ | 5D |] | 5E | ^ | 5F | _ |
| 60 | , | 61 | a | 62 | b | 63 | c | 64 | d | 65 | e | 66 | f | 67 | g | 68 | h | 69 | i | 6A | j | 6B | k | 6C | l | 6D | m | 6E | n | 6F | o |
| 70 | p | 71 | q | 72 | r | 73 | s | 74 | t | 75 | u | 76 | v | 77 | w | 78 | x | 79 | y | 7A | z | 7B | { | 7C | | 7D | } | 7E | ~ | | |

Abbildung 5.1 Der Zeichensatz ISO 646-US (US-ASCII)

Für das Deutsche, das Dänische und vierzehn weitere Sprachen sieht ISO 646 deshalb nationale Varianten vor, welche die US-ASCII-Zeichen # (Hash), \$ (Dollar), @ (At), [(eckige Klammer auf), \ (Backslash),] (eckige Klammer zu), ^ (Caret), ` (Grave), { (geschweifte Klammer auf), | (Strich), } (geschweifte Klammer zu) und ~ (Tilde) durch nationale Zeichen ersetzen. Die deutsche Variante ISO 646-DE in Abbildung 5.2 nimmt beispielsweise die in Tabelle 5.1 dargestellten Ersetzungen vor.

| ISO 646-US | ISO 646-DE |
|------------|------------|
| [| Ä |
| \ | Ö |
|] | Ü |
| { | ä |
| | ö |

| | |
|---|---|
| { | ü |
| ~ | ß |
| @ | § |

Tabelle 5.1 Die Ersetzungen in der deutschen Variante von ISO 646

Aus diesem Grund erscheint ein Absender „Technische Universität München, Arcisstraße 21“ gelegentlich auch heute noch in amerikanischen E-Mail-Programmen in der Form „Technische Universit{t M}nchen, Arcisstraße 21“. Die ursprünglich mit ISO 646-DE angelegte Codierung der Adresse wird dann mithilfe von ISO 646-US ausgetauscht.

| | | | | | | | | | | | | | | | |
|------|------|------|------|-------|------|------|------|------|------|------|------|------|------|------|------|
| 20 | 21 ! | 22 „ | 23 # | 24 \$ | 25 % | 26 & | 27 , | 28 (| 29) | 2A * | 2B + | 2C , | 2D - | 2E . | 2F / |
| 30 0 | 31 1 | 32 2 | 33 3 | 34 4 | 35 5 | 36 6 | 37 7 | 38 8 | 39 9 | 3A : | 3B ; | 3C < | 3D = | 3E > | 3F ? |
| 40 § | 41 A | 42 B | 43 C | 44 D | 45 E | 46 F | 47 G | 48 H | 49 I | 4A J | 4B K | 4C L | 4D M | 4E N | 4F O |
| 50 P | 51 Q | 52 R | 53 S | 54 T | 55 U | 56 V | 57 W | 58 X | 59 Y | 5A Z | 5B Ä | 5C Ö | 5D Ü | 5E ^ | 5F _ |
| 60 , | 61 a | 62 b | 63 c | 64 d | 65 e | 66 f | 67 g | 68 h | 69 i | 6A j | 6B k | 6C l | 6D m | 6E n | 6F o |
| 70 p | 71 q | 72 r | 73 s | 74 t | 75 u | 76 v | 77 w | 78 x | 79 y | 7A z | 7B ä | 7C ö | 7D ü | 7E ß | |

Abbildung 5.2 Die deutsche Variante ISO 646-DE von ISO 646

Zeichensätze

Mitte der achtziger Jahre erweiterte die European Computer Manufacturer's Association (ECMA) den Zeichensatz US-ASCII zu einer Familie von Zeichensätzen, mit denen die alphabetischen Schriftsysteme kodiert werden können. Die inzwischen von der ISO unter dem Namen ISO 8859 kodierte Zeichensatzfamilie besteht aus den Zeichensätzen ISO 8859-1 bis ISO 8859-15. Jeder Zeichensatz ISO 8859-X umfasst die 128 US-ASCII-Zeichen und ergänzt sie um weitere 128 Zeichen. Für die meisten westeuropäischen Sprachen wie das Deutsche, Dänische oder Französische, ist ISO 8859-1, auch ISO Latin-1 erstellt worden (siehe Tabelle 5.3).

| | | | | | | | | | | | | | | | |
|------|------|------|------|------|------|------|------|-------|------|------|------|------|------|------|------|
| A0 | A1 i | A2 ¢ | A3 £ | A4 ¤ | A5 ¥ | A6 ¦ | A7 § | A8 .. | A9 © | AA ª | AB « | AC ¬ | AD — | AE ® | AF — |
| B0 ° | B1 ± | B2 ² | B3 ³ | B4 , | B5 µ | B6 ¶ | B7 . | B8 , | B9 ¹ | BA º | BB » | BC ¼ | BD ½ | BE ¾ | BF ¸ |
| C0 À | C1 Á | C2 Â | C3 Ã | C4 Ä | C5 Å | C6 Æ | C7 Ç | C8 È | C9 É | CA È | CB Ë | CC Ì | CD Í | CE Í | CF Í |
| D0 Đ | D1 Ñ | D2 Ò | D3 Ó | D4 Ô | D5 Õ | D6 Ö | D7 × | D8 Ø | D9 Ù | DA Ú | DB Û | DC Ü | DD Ý | DE Þ | DF ß |
| E0 à | E1 á | E2 â | E3 ã | E4 ä | E5 å | E6 æ | E7 ç | E8 è | E9 é | EA ê | EB ë | EC ì | ED í | EE î | EF ï |
| F0 ð | F1 ñ | F2 ò | F3 ó | F4 ô | F5 õ | F6 ö | F7 ÷ | F8 ø | F9 ù | FA ú | FB û | FC ü | FD ý | FE þ | FF ÿ |

Abbildung 5.3 Der Zeichensatz ISO 8859-1 (ISO Latin-1)

ISO 8859-7 deckt das griechische Alphabet ab und ISO-8859-15 ersetzt im Wesentlichen das internationale Währungssymbol mit dem Eurozeichen.

Die sogenannte Code Page 1252 von Microsoft Windows (siehe Abbildung 7.1) ist in weiten Teilen identisch mit ISO 8859-1, ersetzt aber einige Kontrollzeichen von ISO 8859-1 durch druckbare Zeichen. So ist u. a. das Eurozeichen € vorhanden.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|---|----|---|----|---|----|----|----|-----|----|---|----|---|----|---|----|----|----|---|----|----|----|----|----|----|----|----|----|---|----|---|----|---|----|---|----|----|----|---|----|---|----|---|----|---|----|---|----|---|
| 80 | € | 82 | , | 83 | f | 84 | ,, | 85 | ... | 86 | † | 87 | ‡ | 88 | ^ | 89 | % | 90 | Š | 91 | ‘ | 92 | , | 93 | ‘‘ | 94 | ’’ | 95 | • | 96 | - | 97 | - | 98 | ~ | 99 | TM | 9A | š | 9B | < | 9C | Œ | 9D | œ | 9E | ž | 9F | Ÿ |
| A0 | i | A1 | ı | A2 | ç | A3 | £ | A4 | ¤ | A5 | ¥ | A6 | ı | A7 | § | A8 | .. | A9 | © | AA | ä | AB | « | AC | ¬ | AD | - | AE | ® | AF | - | — | | | | | | | | | | | | | | | | | |
| B0 | o | B1 | ± | B2 | 2 | B3 | 3 | B4 | , | B5 | μ | B6 | ¶ | B7 | . | B8 | „ | B9 | 1 | BA | º | BB | » | BC | ¼ | BD | ½ | BE | ¾ | BF | ¿ | | | | | | | | | | | | | | | | | | |
| C0 | À | C1 | Á | C2 | Â | C3 | Ã | C4 | Ä | C5 | Å | C6 | Æ | C7 | Ç | C8 | È | C9 | É | CA | GB | CC | CD | CE | Í | CF | Ï | | | | | | | | | | | | | | | | | | | | | | |
| D0 | Đ | D1 | Ñ | D2 | Ò | D3 | Ó | D4 | Ô | D5 | Ö | D6 | Ö | D7 | × | D8 | Ø | D9 | Ù | DA | Ú | DB | Û | DC | Ü | DD | Ý | DE | Þ | DF | Þ | | | | | | | | | | | | | | | | | | |
| E0 | à | E1 | á | E2 | â | E3 | ã | E4 | ä | E5 | å | E6 | æ | E7 | ç | E8 | è | E9 | é | EA | eb | EC | ED | EE | í | EF | ï | ñ | ò | ó | ô | ö | ÷ | ø | ù | ú | fb | û | ü | ý | þ | ÿ | | | | | | | |
| F0 | õ | F1 | ñ | F2 | ò | F3 | ó | F4 | ô | F5 | ö | F6 | ö | F7 | ÷ | F8 | ø | F9 | ù | FA | ú | FB | û | FC | ü | FD | ý | FE | þ | FF | ÿ | | | | | | | | | | | | | | | | | | |

Abbildung 5.4 Der Windows-Zeichensatz 1252

Anfang der neunziger Jahre kamen Unicode und ISO/IEC 10646 heraus. Das Ziel dieser beiden in einem Sinne, den wir noch genauer kennen lernen werden, äquivalenten Zeichen-Codierungen ist Universalität, also Texte aus sämtlichen Sprachen der Welt eindeutig zu codieren. Die aktuellen Versionen sind Unicode 8.0 und ISO/IEC 10646-2014, die alle modernen und viele klassische Sprachen abdecken. Unicode und ISO/IEC 10646 werden laufend um weitere historisch bedeutsame Zeichen und Sprachen ergänzt.

Am Rande sei erwähnt, dass in unseren heutigen mobilen Nachrichtenanwendungen sogenannte Emoji-Piktogramme millionenfach zum Einsatz kommen. Das Unicode-Konsortium hat diese Nicht-Buchstabenzeichen in den Unicode-Zeichensatzstandard seit Version 6.0 aufgenommen. In der kommenden Version 9.0 (neue Hauptversionen sollen regelmäßig jedes Jahr im Juni veröffentlicht werden) kommen viele weitere Emojis hinzu.

Die Codierungen ISO 8859 und Unicode sind von zentraler Bedeutung für die Dokumentrepräsentation in den im Internet häufig verwendeten Auszeichnungssprachen HTML und XML. Ohne internationale Standards für die Codierung universeller Zeichensätze wäre der grenzüberschreitende Dokumentenaustausch im Internet und im Web zum Scheitern verurteilt.

Selbsttestaufgabe 1.6:

Was versteht man unter dem Begriff „glatter Text“? Wozu benötigt man in diesem Zusammenhang eine ZeichenCodierung?

Im Folgenden werden diese Codierungen genauer vorgestellt, und zwar im Rahmen eines abstrakten Codierungsmodells für Zeichen [DYIWFT02][FSS82][WD00]. Das Codierungsmodell stellt einen konzeptuellen Rahmen dar, innerhalb dessen man die Codierung von Zeichen in Bitmuster strukturieren und so besser verstehen kann. Die einzelnen Bestandteile des Codierungsmodells sind:

- ein abstrakter Zeichensatz,
- eine Codetabelle,
- ein Codierungsformat,
- ein Codierungsschema und
- eine Übertragungssyntax, die über sogenannten „glatten Text“ hinausgeht.

generisches
Zeichensatzmodell

Die einzelnen Bestandteile des Codierungsmodells entsprechen zunehmend konkreteren Repräsentationen von Zeichen, von einem abstrakten Begriff hin zu Bitmustern. Zwischen einer abstrakten Repräsentationsebene und der nächstliegenden konkreteren Repräsentationsebene besteht eine Abbildungsvorschrift im mathematischen Sinne. Um über mehrere Ebenen hinweg von einer Zeichenposition in einer Codetabelle direkt zu seiner Codierung in einem Codierungsschema zu gelangen, können die verschiedenen Abbildungsvorschriften konkateniert werden. Daraus entsteht eine sogenannte Zeichenkarte.

Abstrakter Zeichensatz

abstrakter Zeichensatz

Die obere abstrakte Ebene des Codierungsmodells ist der abstrakte Zeichensatz. Dieser besteht aus einer Menge sogenannter abstrakter Zeichen. Ein abstraktes Zeichen ist eine Informationseinheit, die zur Repräsentation, Organisation oder Kontrolle von Text dient. Es handelt sich hierbei also um Buchstaben, Ziffern, Interpunktionszeichen, Akzente, grafische Symbole, ideografische Zeichen, Leerzeichen, Tabulatoren, Zeilenweiterschaltung und -vorschub und Kontrollcodes für Auswahl-Markierungen. Der Zeichensatz ISO 64-US (US-ASCII) beispielsweise besteht aus 33 Kontrollzeichen und 95 druckbaren Zeichen.

Ein abstrakter Zeichensatz beinhaltet jedoch noch keine Ordnung oder Nummerierung der im Zeichensatz enthaltenen Zeichen. Dieser Aspekt kommt in der nächsten Stufe des Codierungsmodells in Form einer sogenannten Codetabelle hinzu.

Abstrakte Zeichensätze wurden in der Regel festgelegt, um alle Texte einer bestimmten Sprache oder Sprachfamilie angemessen zu codieren. Der Anspruch, alle

lebenden und alle historisch bedeutsamen Sprachen mit einem einzigen Zeichensatz codieren zu können, führte zu den (identischen) Zeichensätzen von Unicode und ISO/IEC 10646, dem sogenannten Universal Character Set, kurz UCS, genannt.

Zur visuellen Präsentation von Text dienen abstrakte grafische Einheiten, die Glyphen genannt werden. Glyphen entsprechen auf der Präsentationsebene den Zeichen auf der Codierungsebene. Die Abbildung 5.5 enthält grafische Ausprägungen der Glyphen für den Buchstaben A, sogenannte Glyphenbilder. Nach Design und anderen Charakteristika zusammengehörige Glyphenbilder bilden einen Font.



Abbildung 5.5 Typografie und Auswirkungen

Die Beziehung zwischen abstrakten Zeichen und Glyphen ist komplex. In Tabelle 5.2 gibt es eine beispielhafte Gegenüberstellung. Im einfachsten Fall präsentiert sich ein einzelnes Zeichen als genau eine Glyphe. Einzelne Glyphen können jedoch auch ganze Folgen von abstrakten Zeichen darstellen, wie etwa bei den Ligaturen fi, fl, ff oder ffl. Ein einzelnes abstraktes Zeichen mit Akzent, wie das Zeichen ö, kann durch zwei Glyphen dargestellt sein, nämlich die Glyphe für das Basiszeichen o und die darüber positionierte Glyphe für den Akzent. Im Tamilischen gibt es Buchstaben, die durch ein Paar von Glyphen repräsentiert werden. Dieses Glyphenpaar umrahmt dann in der formatierten Sicht einen weiteren Glyphen, der den Nachbarbuchstaben im Text darstellt. Im Arabischen schließlich gibt es kontextuelle Formen: ein Zeichen wird je nach dem Kontext seiner Nachbarzeichen durch Glyphen ganz verschiedener grafischer Form dargestellt. Ähnliche Phänomene sind aus der mittelalterlichen Satztechnik bekannt.

Gutenberg verwendete bereits für ein- und denselben Buchstaben verschiedene breite Glyphen. Er verfügte auch bereits über abkürzende Symbole für Wörter oder Silben, um einen gleichmäßigen rechten Rand zu erzeugen. Hier gibt also die Formierung den Kontext für die Wahl eines bzw. mehrerer Glyphen.

In vielen Fällen sind die elementaren Texteinheiten, die als abstrakte Zeichen Eingang in einen Zeichensatz finden sollen, nicht offensichtlich. Sowohl aus den verschiedenen Sprachen, deren Texte kodierbar sein sollen, als auch aus den verschiedenen Verarbeitungsfunktionen, die auf die Texte Anwendung finden sollen, können Anforderungen und somit Konflikte erwachsen.

| Abstrakte Zeichen | Glyphen |
|-------------------|---------|
| A | A |
| f+i | Fi |
| f+f+l | Ffl |
| ö | o+~ |
| → | oder → |

Tabelle 5.2 Das Verhältnis von abstrakten Zeichen und Glyphen

Im Schwedischen beispielsweise sind die Buchstaben ä und ö Buchstaben für sich, die hinter dem Buchstaben z im Alphabet angeordnet sind. Im Französischen dagegen gibt die Diaräse über einem Vokal lediglich einen Hinweis, dass der Vokal separat zu sprechen ist, etwa in duët. Anstelle eines einzigen abstrakten Zeichens è wären also die beiden abstrakten Zeichen e und „ mit zwei kombinierbaren Glyphen natürlicher.

Für die Darstellung eines deutschen Textes in gedruckter Form ist die Unterscheidung von Groß- und Kleinbuchstaben im Zeichensatz äußerst bequem. Bei der Nutzung einer Sortierungsfunktion stellt sie jedoch ein kleines Hindernis dar. Im Spanischen gilt // hinsichtlich des alphabetischen Sortierens als ein einziger Buchstabe, für die Formatierung wird // jedoch als Folge von zwei Buchstaben behandelt.

Die Festlegung eines abstrakten Zeichensatzes erfordert also eine sorgfältige Abwägung von verschiedenen Alternativen, wobei Sprachen, Schriften und Bearbeitungsfunktionen zu berücksichtigen sind. Das zugegebenermaßen sehr allgemein formulierte Ziel bei der Entwicklung des Unicode-Zeichensatzes und der gesamten Unicode-Codierung war es, die Implementierung nützlicher Verarbeitungsprozesse für Textdaten zu ermöglichen.

Unicode 3.0 formuliert zehn Designprinzipien für den Unicode-Standard. Zwei davon lassen sich bereits mit den Begriffen auf der Ebene des Zeichensatzes formulieren:

- Unicode kodiert Zeichen, nicht Glyphen.
- Unicode kodiert glatten Text, keine Formatinformation.

Glyphen spielen bei der Formatierung von Texten eine Rolle und sind auch bei der automatischen Texterkennung interessant. Für die computerisierte Weiterverarbeitung von Inhalten sind sie naturgemäß nicht besonders wichtig.

Codetabelle

Auf der nächsten Stufe des Codierungsmodells steht die Codetabelle, die den abstrakten Zeichen im Zeichensatz eine Codeposition zuweist. Eine Codeposition ist immer eine natürliche Zahl, also größer oder gleich 0 (Für diesen Kurs ist die Menge der natürlichen Zahlen so definiert, dass sie die 0 enthält: Es gilt demzufolge \mathbb{N} entspricht \mathbb{N}^{+0}).

Codetabelle

Der Code-Raum einer ZeichenCodierung ist immer ein Abschnitt der natürlichen Zahlen, der alle Codepositionen enthält. Der Code-Raum kann jedoch auch Zahlen enthalten, die keine zulässigen Codepositionen sind.

Codepositionen werden üblicherweise in Hexadezimalnotation angegeben, wodurch eine natürliche Beziehung zu Bitmustern hergestellt wird. Diese Beziehung kommt auf der nächsten Stufe des Codierungsmodells – dem Codierungsformat – zum Tragen.

Ein Bit dient zur Darstellung oder Speicherung von zwei Zuständen, die durch die Ziffern 1 und 0 repräsentiert werden. Die sechzehn möglichen Sequenzen von vier Bits werden üblicherweise mit den Ziffern 0,...,9 und den Buch-staben A,...,F, auch Hexadezimal-Ziffern genannt, bezeichnet.

Die Hexadezimal-Ziffern 0,...,9 und A,...,F stehen für die Dezimalzahlen von 0 bis 15. Jede Hexadezimalzahl stellt also eine natürliche Zahl dar. Eine Sequenz von Hexadezimal-Ziffern $h_n \dots h_0$ kann in eine Dezimalzahl umgewandelt werden mit folgender Summationsregel:

$$h_0 + h_1 16^1 + h_2 16^2 + \dots + h_n 16^n$$

Die Zahl 16 ist hierbei in Dezimalschreibweise – erkennbar durch das Zeichen $_d$. Die Tabelle 5.3 zeigt eine Beziehung zwischen dezimaler, hex- und binärer Schreibweise.

Die Dezimal-Zahlen von 0 bis 255 können somit durch Sequenzen von zwei Hexadezimal-Ziffern dargestellt werden und Zahlen von 0 bis 65.535 durch Sequenzen von vier Hexadezimal-Ziffern und damit durch 16 Bit. Sequenzen mit einer Länge von 16 Bit werden auch *Word* genannt.

Wie in Abbildung 5.1 gezeigt, umfasst der Code-Raum für US-ASCII 128 Positionen. In hexadezimaler Schreibweise geht der Code-Raum also von 0 bis 7F.

| Hex-Ziffer | Bitmuster | Dezimalschreibweise |
|------------|-----------|---------------------|
| 0 | 0000 | 0 |
| 1 | 0001 | 1 |
| 2 | 0010 | 2 |
| 3 | 0011 | 3 |
| 4 | 0100 | 4 |
| 5 | 0101 | 5 |
| 6 | 0110 | 6 |
| 7 | 0111 | 7 |
| 8 | 1000 | 8 |
| 9 | 1001 | 9 |
| A | 1010 | 10 |
| B | 1011 | 11 |
| C | 1100 | 12 |
| D | 1101 | 13 |
| E | 1110 | 14 |
| F | 1111 | 15 |

Tabelle 5.3 Die sechzehn Hex-Ziffern und ihre Repräsentationen als Bitmuster

Der Code-Raum für ISO 8859-X umfasst die 256_d Positionen von 0 bis FF_h , der für Unicode die 65.536_d Positionen von 0 bis $FFFF_h$ plus die $1.048.576_d$ Positionen von 10000_h bis $10FFFF_h$ und der für ISO/IEC 10646 die $2.147.483.648_d$ Positionen von 0 bis $7FFFFFFF_h$.

| Kodierung | Code-Raum | Zahl der Codeposition |
|-----------------|------------|-----------------------|
| US-ASCII | 0-7F | 128 |
| ISO 8859-X | 0-FF | 256 |
| Unicode | 0-10FFFF | $65.536+1.048.576$ |
| ISO / IEC 10646 | 0-7FFFFFFF | $2.147.483.648$ |

Tabelle 5.4 Code-Räume für verschiedene Codierungen

Die Code-Räume können weiter strukturiert sein:

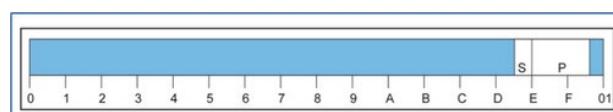


Abbildung 5.6 Unicode-Code-Raum mit Surrogatpositionen und privatem Bereich

Die Positionen von 0 bis $1F_h$ und die Position $7F_h$ von US-ASCII sind Kontrollzwecken vorbehalten, ebenso die Positionen 80_h bis $9F_h$ von ISO 8859-X. Beide Code-Räume können durch ein Byte repräsentiert werden.

Der Unicode-Code-Raum in Abbildung 5.6 enthält Lücken an den 2.048_d *Surrogatpositionen* von $D800_h$ bis $DFFF_h$ und an den Positionen $FFFE_h$ sowie $FFFF_h$. Diese Positionen sind für spezielle Funktionen vorgesehen und es werden ihnen auch künftig keine Unicode-Zeichen zugeordnet werden.

Die 6.400 Unicode-Codepositionen von $E000_h$ bis $F8FF_h$ – also knapp 10 Prozent des Code-Raums von 0 bis $FFFF_h$ – sind für private Zwecke reserviert. Die Positionen sind etwa für Logos, für Icons oder für in speziellen Notationen benötigte Zeichen, etwa wie bei Musik und Tanz, außerhalb des Unicode-Zeichensatzes freigegeben. Die Codepositionen in diesem Bereich werden auch in späteren Unicode-Ergänzungen für private Verwendung freigegeben.

Von den übrigen 57.086_d Unicode-Positionen im Bereich von 0 bis $FFFF_h$ sind 65_d mit Kontrollzeichen belegt, 49.194_d Positionen ein druckbares Zeichen zugeordnet und 7.827_d Positionen noch frei für Erweiterungen von Unicode. Im nächsten Abschnitt bei der Diskussion von Codierungsformaten wird erläutert, wie der Unicode-Code-Raum im Bereich von 0 bis $FFFF_h$ durch zwei Bytes und im Bereich von $10000h$ bis $10FFFF_h$ durch vier Bytes repräsentiert wird. Derzeit, d. h. bis zur Version 3.0, enthält die Unicode-Codetabelle nur Werte im Bereich von 0 h bis $FFFF_h$.

Der Code-Raum von ISO/IEC 10646 ist konzeptuell unterteilt in 128 Gruppen von 256 Ebenen, wobei jede Ebene 256 Spalten mit je 256 Seiten, also insgesamt 65.536 Zeichen, enthält. Derzeit weist ISO/IEC 10646-1:2000 nur den Codepositionen von 0 bis $FFFF_h$, die in Ebene 0 von Gruppe 0 liegen Werte zu. Diese Ebene heißt auch Basic Multilingual Plane. Die Codetabellen von Unicode Version 3.0 und ISO/IEC 10646-1:2000 sind Position für Position identisch. Das Unicode-Konsortium und die ISO haben sich verpflichtet, diese Übereinstimmung auch bei späteren Ergänzungen der Codetabelle aufrechtzuerhalten; sie haben eine Organisationsform wechselseitiger Liaisons geschaffen, die dies garantieren können. Unicode beinhaltet für jedes in seiner Codetabelle kodierte Zeichen die folgenden Daten:

- die Codeposition des Zeichens,
- ein typisches Glyphenbild für das Zeichen,
- einen Namen,
- semantische Information.

Die semantische Information ist eine Spezialität bei Unicode. ISO/IEC 10646 stellt diese Information nicht zur Verfügung. Für die Codeposition 00AF_h steht in Unicode die folgende Information zur Verfügung:

Das Zeichen hat (1) den Unicode-Namen „Macron“, (2) alternative Namen „Overline“ und „APL Overline“ und ist (3) grafisch ein hochgestellter waagerechter Strich. Das Macron bewirkt (4) einen horizontalen Vorschub (engl. *spacing character/blank*) und steht (5) in Beziehung zu weiteren Unicode-Zeichen, u. a. dem Zeichen 0304 Combining Macron, das genauso aussieht wie das Macron selbst, aber beim Formattieren keinen Vorschub bewirkt (engl. *nonspacing character*), sondern sich über das vorangehende Basiszeichen positioniert. Das Makron kann (6) zerlegt werden in das Leerzeichen 0020, gefolgt von dem Combining Macron 0304. Das Macron 00AF ist (7) im Unterschied zum Combining Macron kein Kombinationszeichen sondern ein Basiszeichen.

Das Macron wurde, wie viele andere Unicode-Zeichen, aus Kompatibilitätsgründen in die Codetabelle aufgenommen. Hier wirkt sich das Designziel der Konvertierbarkeit von existierenden Standards nach Unicode und zurück aus. Ein Zeichen, das nur aus Kompatibilitätsgründen in der Codetabelle steht, lässt sich gemäß einer Unicode-Vorgabe immer zerlegen in eine Sequenz aus sogenannten Primärzeichen.

Für Basiszeichen, die von Kombinationszeichen gefolgt sind, wird bei Unicode ein Äquivalenzbegriff definiert. Kombinationszeichen, die grafisch an unterschiedlichen Stellen am Basiszeichen positioniert werden, können miteinander ausgetauscht werden. Beispielsweise können für die beiden Kombinationszeichen 0307_h Combining Dot Above und 0323_h Combining Dot Below die Plätze getauscht werden, sodass die beiden Codierungen 006F:0307:0323_h und 006F:0323:0307_h für ein o mit einem Punkt über sich und einem Punkt unter sich miteinander äquivalent sind.

Auf dem Äquivalenzbegriff für Zeichenfolgen und der Zerlegung von Kompatibilitätszeichen in Folgen von Primärzeichen und der Komposition von Kompatibilitätszeichen aus Folgen von Primärzeichen beruhen auch zwei Normalisierungen von Unicode Zeichenketten, nämlich die Precomposed Form und die Decomposed Form.

Bei Web-Texten sollten stets Zeichen benutzt werden, die schon so weit wie möglich zusammengesetzt sind. Verbleibende Kombinationszeichen sollten in normierter Reihenfolge angegeben werden. Web-Anwendungen sollten bei Texten, die eingelesen werden, so liberal wie möglich programmiert sein. Im Gegenzug dazu sollten sie so konservativ filtern bei Texten, die sie ausgeben. Dies soll Kompatibilitätsprobleme weitgehend vermeiden.

Von den zehn Designprinzipien, die Unicode anführt, lassen sich fünf auf der Ebene der Codetabelle erklären:

- 1 **Unifikation:** Unicode repräsentiert Zeichen, die in verschiedenen Alphabeten vorkommen, aber vom Aussehen her ähnlich sind, nur einmal.
- 2 **Konvertierbarkeit zwischen etablierten Standards:** Eine eineindeutige Abbildung aus einem etablierten Standard nach Unicode soll ermöglicht werden. Zeichenpositionen aus einem einzelnen international verbreiteten Zeichensatz, die nach dem Designprinzip der Unifikation eigentlich miteinander identifiziert werden müssten, erhalten trotzdem getrennte Unicode-Codepositionen.
- 3 **Semantik:** Unicode definiert semantische Eigenschaften für Zeichen.
- 4 **Dynamische Komposition:** Jedes Basiszeichen kann mit beliebig vielen Kombinationszeichen gleichzeitig gepaart werden.
- 5 **Charakterisierung äquivalenter Codierungen:** Für die verschiedenen Codierungen eines Basiszeichens mit Kombinationszeichen oder eines primären und eines aus Kompatibilitätsgründen in den Zeichensatz aufgenommenen Zeichens kann eine normalisierte Codierung gefunden werden.

Codierungsformat

Mithilfe des Codierungsformats für eine Codetabelle werden die Bitrepräsentationen für die Codeposition festgelegt. Dazu gibt es eine Code-Einheit, die in der Regel aus acht oder sechzehn Bits besteht. Durch das Codierungsformat werden dann die Positionen eines Code-Raums in Sequenzen von Code-Einheiten und somit in Bitmuster abgebildet. Wird jede Codeposition einer Codetabelle auf die gleiche Anzahl von Code-Einheiten abgebildet, sprechen wir von einem Codierungsformat fester Länge; andernfalls hat das Codierungsformat variable Länge.

Ein kanonisches Codierungsformat fester Länge ergibt sich aus der Dualzahldarstellung von Codepositionen. Die Anzahl der jeweils benötigten Code-Einheiten ergibt sich aus der Größe des Code-Raum und der Zahl der Bits in einer Code- Einheit. In Tabelle 5.5 gibt es eine Übersicht über die verschiedenen Codierungsformate.

| Codierung/Codierungsformat | Code-Raum/Code-Einheit/Länge |
|---------------------------------|------------------------------------|
| US-ASCII / kanonisch | 0-7F / 1 Byte / fest (1) |
| ISO 8859-X / kanonisch | 0-FF / 1 Byte / fest (1) |
| SO / IEC 10646, Ebene 0 / UCS-2 | 0-FFFF / 1 Word / fest (1) |
| ISO / IEC 10646 / UCS-4 | 0-7FFFFFFF / 2 Word / fest (1) |
| Unicode 3.0 / UTF8 | 0-10FFFF / 1 Byte / variabel (1-4) |
| Unicode 3.0 / UTF16 | 0-10FFFF / 1 Word / variabel (1-2) |

Tabelle 5.5 Codierungsformate für verschiedene Codierungen

Für US-ASCII bzw. seine ISO 646-Varianten sowie für die ISO 8859-XCodetabellen ist das kanonische Codierungsformat das einzige gebräuchliche. Für die höchstens

256_d vielen Positionen genügen eine Code-Einheit von acht Bits sowie eine Code-Einheit pro Zeichenposition.

Die beiden kanonischen Codierungsformate für die Code-Räume von 0 bis $FFFF_h$ bzw. von 0 bis $7FFFFFFF_h$ heißen UCS2 und UCS4. Beide Codierungsformate repräsentieren eine Codeposition mit genau einer Code-Einheit. Im Falle von UCS2 ist die Code-Einheit 16 Bit lang und im Falle von UCS4 ist sie 32 Bits lang.

Prinzipiell kann eine Codierung mehrere Codierungsformate definieren und das kanonische Codierungsformat muss nicht darunter sein. Unicode 3.0 beispielsweise definiert nur zwei Codierungsformate, nämlich UTF8 und UTF16, wobei UTF für Unicode Transfer Format steht. UTF8 hat eine Code-Einheit von 8 Bits Länge und repräsentiert Codepositionen mit einer bis vier Code-Einheiten. UTF16 hat eine Code-Einheit von 16 Bits Länge und repräsentiert Codepositionen mit einer bis zwei Code-Einheiten.

Ursprünglich war Unicode für einen Code-Raum von 0 bis $FFFF_h$ ausgelegt, sodass sechzehn Bits genügt hätten, um Codepositionen darzustellen. Tatsächlich hätte dieser Code-Raum ja auch mindestens bis Version 3.0 ausgereicht. Als jedoch klar wurde, dass zumindest für alle historisch interessanten Sprachen und Alphabete der Code-Raum nicht ausreichen würde und weil man mit ISO/IEC 10646 und seinem größeren Code-Raum kompatibel bleiben wollte, wurde das Konzept der Surrogat-Codepositionen eingeführt und der Code-Raum von Unicode bis zur Position $10FFFF_h$ erweitert. Das Codierungsformat UTF16 benutzt jeweils ein Paar von Surrogatpositionen, um Positionen jenseits von $FFFF_h$ darzustellen.

UTF16 stellt wie die kanonische Codierung jede gültige Codeposition im Bereich von 0 bis $FFFF_h$ dar. Die ungültigen Codepositionen in dem sogenannten hohen Surrogatbereich von $D800_h$ bis $DBFF_h$ und in dem niedrigen Surrogatbereich von $DC00$ bis $DFFF$ entsprechen den hohen und niedrigen Surrogat-Bitfolgen der Form $110110xxxxxxxx_b$ und $110111xxxxxxxx_b$, in denen jeweils zehn Bitpositionen für 2^{10} verschiedene Werte frei wählbar sind. Die Gegenrechnungen

$$\begin{aligned} DBFF_h + 1 - D800_h &= DC00_h - D800_h = 400_h = 4 \times 16^2_d = 2^{10}_d \\ DFFF_h + 1 - DC00_h &= E000_h - DC00_h = 400_h = 2^{10}_d \end{aligned}$$

bestätigen, dass in jedem der beiden Surrogatbereiche 2^{10} , also 1024_d ungültige Codepositionen liegen. Ein Paar von Surrogatpositionen, von denen die erste im hohen und die zweite im niedrigen Bereich liegt, ist somit eindeutig charakterisiert durch zwanzig Bits, die wiederum die 2^{20} Werte von 10000_h bis $10FFFF_h$ repräsentieren können:

$$10FFFF_h + 1 - 10000_h = 110000_h - 10000_h = 100000_h = 16^5_d = 2^{20}_d$$

Es ist deshalb stimmig, wenn bei UTF16 eine Codeposition P im Bereich von 10000_h bis $10FFFF_h$ durch die kanonische Bitdarstellung der beiden hohen und niedrigen Surrogatpositionen H und L repräsentiert, wobei sich H und L wie folgt berechnen:

Umgekehrt bestimmen eine hohe und eine niedrige Surrogatposition H und L eine Unicode-Position P im Bereich von 10000 bis 10 FFFF_h wie folgt:

$$P = (H - D800_h) \cdot 400_h + (L - DC00_h) + 10000_h$$

Das zweite Unicode-Codierungsformat, nämlich UTF8, hat die besondere Eigenschaft, die ersten 128 Codepositionen kanonisch mit einem Byte darzustellen. UTF8 ist damit US-ASCII-transparent.

Die Codierungseinheit von UTF8 ist 8 Bits lang. UTF8 stellt jede Position im Bereich von 0 bis $FFFF_h$ mit ein bis drei Codierungseinheiten, im Bereich von 0 bis $10FFFF_h$ mit ein bis vier Codierungseinheiten und im Bereich von 0 bis $7FFFFFFF_h$ mit ein bis sechs Codierungseinheiten dar.

Die UTF8-Darstellung einer Position mit einer einzigen Codierungseinheit hat die Form $0xxxxxx_b$ mit einer führenden 0 und beliebigen Bits, die anstelle des Platzhalters x stehen. Die UTF8-Darstellung einer Position mit n Codierungseinheiten hat die Form $1^n0x^{7-n}_b$ gefolgt von $n - 1$ Codierungseinheiten der Form $10xxxxxx_b$. Bei drei Codierungseinheiten sind also genau sechzehn Bits frei wählbar, sodass mit drei Codierungseinheiten unter UTF8 Code-Positionen bis $FFFF_h$ darstellbar sind.

Die mit n Codierungseinheiten darstellbaren Positionsbereiche lassen sich aus der Tabelle 5.6 Grunddaten zu Codierungseinheiten entnehmen.

| Einheiten | Darstellungsform | freie Bits | Maximum |
|------------------|-----------------------------|-------------------|-----------------|
| 1 | 0xxxxxx | 7 | 7F |
| 2 | 110xxxxx 10xxxxxx | 11 | 7FF |
| 3 | 1110xxxx (10xxxxxx)2 | 16 | FFFF |
| 4 | 11110xxx (10xxxxxx)3 | 21 | 1FFFFFF |
| 5 | 111110xx (10xxxxxx)4 | 26 | 3FFFFFF |
| 6 | 1111110x (10xxxxxx)5 | 31 | 7FFFFFFF |

Tabelle 5.6 Grunddaten zu Codierungseinheiten

Für die Darstellung einer Codeposition unter UTF8 muss immer die kleinste Zahl von Codierungseinheiten gewählt werden. Eine Codeposition im Bereich von $800h$ bis $FFFF_h$ muss also mit drei Codierungseinheiten und darf nicht – obwohl das technisch möglich wäre – mit vier Codierungseinheiten dargestellt werden.

Die Umkehrfunktion von UTF8, die aus einer Folge von Code-Einheiten eine Codeposition berechnet, darf jedoch auch zu lange Darstellungen korrekt umrechnen. Die zwei Bytes C1BF_h dürfen also unter UTF8 in die Position 7F_h zurückgerechnet werden, obwohl die UTF8-Darstellung der Position 7F_h das Byte 7F_h ist.

Das Unicode-Designprinzip der Effizienz lässt sich auf Ebene von Codierungsformaten erläutern. Für jedes der beiden Unicode-Codierungsformate UTF8 und UTF16 lässt sich in einem Strom von Positionsdarstellungen von jeder Codierungseinheit aus der Anfang der zugehörigen Positionsdarstellung mit beschränktem Backup finden. Im Falle von UTF16 muss höchstens um eine Codierungseinheit zurückgegangen werden, im Falle von UTF8 höchstens um drei Positionen.

Es sei noch angemerkt, dass UTF32 eine Zeichencodierung ist, welche im Gegensatz zu allen anderen UTF-Codierungen jedes Zeichen mit 4-Byte codiert und der UCS4-Codierung entspricht. Für Zeichen wie historische Schriftzeichen, alt- ägyptische Hieroglyphen oder seltene chinesische Schriftzeichen reichen 16 Bit oft nicht mehr aus. Hierfür wird jedes Zeichen mit 32 Bit kodiert. UTF32 wurde zukunftsorientiert entwickelt. Da viele Bytes noch nicht belegt sind, ist es möglich, auch zukünftige Symbole, Zeichen, oder Sprachen mit zu integrieren. Dadurch, dass UTF32 keine variablen Bytelängen besitzt, ergibt sich der Vorteil, dass jedes Byte eindeutig zugeordnet ist und nicht an einer variablen Stelle sitzen kann.

Somit kann der Benutzer sehr einfach auf ein bestimmtes Zeichen im Speicherbereich zugreifen. Des Weiteren lässt sich durch die feste Speichervorgabe sehr einfach eine Textlänge ermitteln, indem der Gesamtspeicher lediglich durch vier Byte dividiert wird. Es sollte allerdings bei der Verwendung von UTF32 der wesentlich (!) höhere Speicherplatz-Bedarf bedacht werden.

In der Version ISO/IEC 10646-3:2003 werden die gleichen Formate UTF8, UTF16 und UTF32 beschrieben wie in Unicode 4.0. Seit Revision 2011 sind die Standards hinsichtlich der Codierungen deckungsgleich.

Codierungsschema

Damit Daten über Netzwerke zuverlässig ausgetauscht werden können, müssen sie in eine lineare Folge von Bytes gebracht werden. Dieser Vorgang wird auch Serialisierung genannt. Mithilfe eines Codierungsformats wird der Text als Folge von Codierungseinheiten dargestellt. Ein Codierungsschema wird nun dazu genutzt zu einem Codierungsformat zusätzlich festzulegen, wie die Codierungseinheiten in Bytefolgen zu serialisieren sind.

tisch. So kann also UTF8 sowohl als Codierungsformat als auch als Codierungsschema bezeichnet werden.

Ist die Codierungseinheit ein Word, wie bei UTF16, so gibt es zwei Möglichkeiten der Serialisierung: Big Endian (das höherwertige Bit kommt zuerst) und Little Endian. Dementsprechend gibt es zu UTF16 zwei Codierungsschemata: UTF16-BE und UTF16-LE. Analoges gilt für UCS2, UCS4 und UTF32.

Codierungsschema

Mit dem sogenannten Byte-Reihenfolge-Codierungszeichen (engl. *byte order mark, BOM*) an Position FEFF_h kann bei einem UTF16-repräsentierten Datenstrom signalisiert werden, welche der beiden Serialisierungen, höherwertiges Byte zuerst (engl. *big endian*) oder niederwertiges Byte zuerst (engl. *little endian*), vorliegt. Da FFE h keine zulässige Codeposition bei Unicode ist, lässt sich schließen, dass das Codierungsschema UTF16-LE vorliegt. Dies funktioniert jedoch nur dann, wenn als die ersten zwei Bytes FF_h und FE_h gelesen werden. Ein BOM am Anfang eines UTF16- kodierten Datenstroms ist nicht Bestandteil der Daten und wird – abgesehen von seiner Signalwirkung für die Serialisierung – ignoriert. Tritt die Codeposition FEFF_h dagegen an anderer Stelle im Datenstrom auf, so wird sie als das entsprechende Unicode-Zeichen Zero Width No-Break Space interpretiert, das dieser Position zugewiesen ist. Bei UTF-32 stehen davor oder dahinter noch zwei Nullbytes, die zur Erkennung der Byte-Reihenfolge dienen (UTF32-BE 00 00 FE FF, UTF32-LE FF FE 00 00).

BOM

Generell darf ein Unicode-Zeichenstrom am Anfang mit dem zusätzlichen BOM versehen werden. Anwendungen, die über keine Metainformation über die Natur des Datenstroms verfügen, erhalten so das Signal, dass es sich im Folgenden um Unicode-Daten, die in einem bestimmten Codierungsschema vorliegen, handelt. Das erste BOM eines Zeichenstroms bildet also keinen Teil der eigentlichen Daten.

An dieser Stelle seien noch folgende ergänzende Links des W3C zur Wahl der Zeichencodierung in HTML benannt:

<https://www.w3.org/International/questions/qa-choosing-encodings.de>,

<https://www.w3.org/International/questions/qa-html-encoding-declarations>,

<https://www.w3.org/International/questions/qa-byte-order-mark>

Speziell zum Thema unicode:

<http://unicode.org/>

Unicode Consortium

<http://unicode.org/charts/> Zeichen der Unicode-Codierung

Jenseits von glattem Text – eine Übertragungssyntax

Mit glattem Text, der als Folge von Unicode-Zeichen kodiert ist, kann der reine Inhalt von strukturierten Dokumenten gehandhabt werden. Da auch eingebettetes Markup – wie etwa bei den im Abschnitt Strukturdarstellungen eingeführten Tags der Form <xxx> und </xxx> – eine Folge von Zeichen ist, scheint es auf den ersten Blick als wäre das Problem, strukturierte Dokumente zu codieren, bereits vollständig gelöst.

Es ergibt sich jedoch eine zusätzliche Komplikation dadurch, dass der Markuptext vom Inhaltstext syntaktisch voneinander unterscheidbar sein muss. Zur Abgrenzung von Markup und Inhalt werden bestimmte Funktionszeichen eingesetzt, die dann außerhalb ihrer syntaktischen Rolle weder im Markup selbst noch im Inhalt des Klartextes vorkommen dürfen. In XML ist < ein solches Funktionszeichen, das den Anfang eines Tags charakterisiert. Das Zeichen < darf deswegen im Inhaltstext eines XML-Dokuments nicht vorkommen.

Die klassische Methode Funktionszeichen in glattem Text unterzubringen, ist die Verwendung von sogenannten Auszeichnungszeichen (engl. *escape characters*). Diese werden nicht selbst als Bestandteil des Textes interpretiert, sondern signalisieren die Präsenz eines Zeichens, das eigentlich nicht im Text vorkommen darf. Im Falle von XML bezeichnet die Formel &#<<Hexziffern>>; oder als &<<Dezimalziffern>>; im Inhaltstext das Unicode- Zeichen an Position <<Hexziffern>>; bzw. an Position <<Dezimalziffern>>. Wir können ein < im Inhaltstext als z. B. als < notieren und das zusätzliche Funktionszeichen & durch &;.

Im Allgemeinen wird glatter Text nicht direkt als Folge von Zeichen auftreten. Vielmehr wird die Zeichenfolge, die einen Text ausmacht, konstruiert aus einer weiteren Folge von sogenannten Eingabezeichen, von denen ganze Teilesequenzen einzelne Zeichen des Textes repräsentieren. In der Praxis existiert also eine zweistufige Codierung von Texten. Zeichen entweder durch sich selbst oder durch spezielle Zeichenfolgen wie &#<<Hexziffern>>; zu codieren, löst insgesamt drei bekannte Probleme:

- In glattem Text lassen sich Funktionszeichen an Stellen unterbringen, an denen sie nicht im Klartext vorkommen dürfen.
- Es können Zeichen in einen Text eingefügt werden, die das verwendete Eingabewerkzeug nicht unterstützt. So kann also in ein XML-Dokument auch mithilfe einer amerikanischen Tastatur etwa der Umlaut ä als ä eingegeben werden.

- Die Paare aus den beiden ASCII-Steuerzeichen linefeed bzw. newline und carriage return können als ein- und dasselbe Zeilenendezeichen interpretiert werden. Auf diese Weise werden die beiden gängigen verschiedenen Konventionen im Text ein Zeilenende zu signalisieren vereinheitlicht. Werkzeuge mit denen die Texte verarbeitet werden, verfügen damit über eine plattformunabhängige Methode zur Zeilen-Nummerierung.

Die Technik, Zeichen durch Zeichenfolgen spezieller Syntax zu codieren, ist Standard in allen Bereichen, in denen Texte erstellt werden. Insbesondere bei der Erstellung von Programmtexten spielt sie eine wichtige Rolle.

Java-Programme etwa dürfen beliebige Unicode-Zeichen der Basic Multilingual Plane enthalten. Dies gilt unter anderem für Namen, Literale für Zeichenketten oder bei Kommentaren. Grundsätzlich kann im Programmtext jedes Zeichen durch sich selbst oder durch eine Sequenz `\u<<xxxx>>` mit einer ungeraden Anzahl des Zeichens `\`, einer positiven Anzahl des Zeichens `u` und einer Sequenz `<<xxxx>>` von genau vier Hexziffern dargestellt werden. Die einzige Ausnahme betrifft das Zeichen `\`, das nur dann für sich selbst stehen darf, wenn es nicht in einer Sequenz der Form `\u<<xxxx>>` vorkommt.

Der Programm-Code in Java zu dem Klassiker *HelloWorld*:

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World");  
    }  
}
```

dürfte also auch in folgender Form erscheinen

```
\u0070ublic class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("\uuuu0048ello World");  
    \\ \u007D  
}
```

Die Java-Notation macht es also möglich, einen beliebigen Unicode-Text mit Zeichen im Bereich zwischen 0 und FFFF_h als einen 7-Bit-ASCII-Text zu codieren.

Selbsttestaufgabe 1.7:

Welches sind die Bestandteile des im Kurstext beschriebenen abstrakten Codierungsmodells? Erläutern Sie diese kurz in eigenen Worten.

Unicode-Werkzeuge

Bei der Diskussion von Werkzeugen zur Bearbeitung von Unicode-Texten gibt es den entscheidenden Unterschied zwischen der Eingabe und der Darstellung von Unicode-Zeichen. Wird ein genuiner Unicode-Editor gewünscht, so ist die Auswahl derzeit noch nicht allzu groß.

Ansonsten unterstützt jeder beliebige Editor für 7-Bit-ASCII den Unicode. Die so erstellten Texte können als Unicode-Texte in UTF8-Codierung ausgegeben werden. Damit können auch nur 7-Bit-ASCII-Zeichen in eigenen Texten verwendet werden. Indirekt gibt es eine weitere Codierungsstufe, über die der Zugriff zu dem vollen Unicode-Zeichenrepertoire gewährleistet ist. Im Falle der Java-Codierung steht ein Werkzeug zur Verfügung, mit dem die Java-Notation wieder dekodiert werden kann. Sie wird so und in verschiedene Codierungsformate übersetzt. Das Werkzeug heißt *native2ascii* und ist Bestandteil des Java Software Development Kit.

<http://www.cs.tut.fi/~jkorpela/iso8859/>

<http://msdn.microsoft.com/en-us/goglobal/cc305145.aspx>

<http://www.unicode.org/versions/Unicode8.0.0/>

<http://www.humancomp.org/uniintro.htm>

6. Zusammenfassung und Ausblick

Zwei wichtige Faktoren machen digitale Dokumente zu modernen Informationsträgern, die vielfach verwendbar und der automatischen Verarbeitung zugänglich sind.

Die beiden Faktoren sind das Modell der strukturierten Dokumente, dem solche digitalen Dokumente entsprechen sollen, und die Standardisierung der Textcodierung, die Textinhalte austauschbar macht.

Beide Faktoren wurden in dieser Kurseinheit besprochen. Als dritter Faktor steht die Standardisierung der Strukturinformation noch aus; damit werden die kommenden Kurseinheiten im Zusammenhang mit XML und den verwandten Sprachfamilien zu tun haben.

7. Anhang: Formate am Beispiel

a. Fertig formatiertes Dokument für ein Journal



Prof. Dr. Anne Brüggemann-Klein
Dr. Angelika Reiser
Prof. Dr. Doris Schmitt-Landsiedel

Ein Computer-Propädeutikum von Studentinnen für Studentinnen

Konzept vom 12.10.1999, Version 1.3

Hintergrund und Ziele

Gerade in technischen Studiengängen spielen Computer als alltägliches Arbeitsmittel eine wichtige Rolle. Es ist wesentlich, dass die Studierenden gleich zu Beginn des Studiums einen vertrauten und sicheren Umgang mit diesem Handwerkszeug haben. Eventuell vorhandene Hemmschwellen müssen sobald wie möglich abgebaut werden; davon ist eine studienzeitverkürzende Wirkung zu erwarten.

Es ist bekannt, dass Schülerinnen weniger Zugang zu einem Computer haben als Schüler und weniger häufig einen eigenen Computer besitzen. Das Computer-Propädeutikum richtet sich deshalb primär an Studentinnen, die dadurch eventuell fehlende Vorerfahrungen ausgleichen können.

Das Kursangebot des Computer-Propädeutikums

Das Computer-Propädeutikum bietet Kurzeinführungen mit praktischen Übungen in für das Studium an der TU München relevante Themen rund um den Computer. Jeder Kurs dauert als Block angeboten-drei Stunden und beinhaltet eine tutorielle Einführung in das Thema, praktische Übungen am Rechner und eine Abschlussbesprechung zu den Übungen. Kursthemen können sein:

- Textverarbeitung mit Word
- Textverarbeitung mit LaTeX
- Textverarbeitung mit Star Office
- E-Mail
- Browse und Informationsnutzung im World-Wide Web
- Erstellen von Informationsangeboten im World-Wide Web (HTML)
- Betriebssystem, Dateiorganisation
- Gestaltung des Desktops (Windows-Manager)
- ASCII-Editoren (vi, emacs)
- Kauf eines PCs

- Datenverbindungen von zu Hause (analog, ISDN)
- Fakultätspezifische Informationsinfrastrukturen

Jeder Kurs sollte sich auf ein enges Thema konzentrieren. Die Themenliste ist offen für Ergänzungen. Kriterium ist, dass der Kursinhalt propädeutischer Natur für das Studium an der TU ist und eine Form der Computernutzung zum Inhalt hat. Ein Kurs kann fakultätsübergreifend oder fakultätspezifisch sein. Die Kurse sind in der Regel auf eine spezielle Rechnerplattform (PC/Windows, Unix) zugeschnitten.

Die Zielgruppe des Computer-Propädeutikums

Das Kursangebot richtet sich an Studentinnen im ersten Semester. Wird ein Angebot von dieser Zielgruppe nicht ausgeschöpft, steht es auch Studentinnen höherer Semester und, in dritter Priorität, männlichen Studierenden offen.

Die Leiterinnen und Leiter des Computer-Propädeutikums

Die Kurse werden vorzugsweise von Studentinnen höherer Semester konzipiert und durchgeführt. Zur Ergänzung des Kursangebots können auch Mitarbeiterinnen und Mitarbeiter oder Studenten höherer Semester Kurse anbieten. Nur von Studierenden angebotene Kurse können im Rahmen des Programms finanziert werden.

Die Kursleiterinnen und -leiter sind selbst für die Organisation von Seminar- und Rechnerräumen und von Zugangsmöglichkeiten zu den Räumen am Wochenende für ihren Kurs verantwortlich. Wir gehen davon aus, dass die Studiendekaninnen und -dekanen sie bei der Organisation unterstützen.

Durchführung

Das Computer-Propädeutikum ist eine Initiative der Frauenbeauftragten der TU München und wird vom Frauenbüro (Kerstin Hansen, Anja Quindeau) organisiert. Die TU München finanziert das Programm aus dem Tutorienprogramm zur Verkürzung der Studiendauer. Für von Studierenden angebotene Kurse wird ein Werksvertrag abgeschlossen, der die Konzeption, Vorbereitung und Durchführung eines dreistündigen Kurses beinhaltet. Konzeption und Vorbereitung eines Kurses werden mit 200 DM und jede Durchführung mit 100 DM honoriert. Das Frauenbüro erhält Hilfskraftmittel aus dem Tutorienprogramm für die Organisation des Computer-Propädeutikums. Wir rechnen mit folgenden Kosten: 20 Kurse, jeder zweimal durchgeführt, zu 20 mal 200 DM plus 20 mal 2 mal 100 DM, also 8000 DM, plus 1000 DM für die Organisation im Frauenbüro.

Eine erste Ausschreibung des Programms ist bereits erfolgt. Interessierte Kursleiterinnen und -leiter können ihr Kursangebot bis zum 5.11.1999 abgeben. Bitte benutzen Sie dazu unser Formular. Die ersten Kurse sollen noch im November 1999 stattfinden. Bei Bedarf erfolgt eine zweite Ausschreibung, zu der neue Kursangebote abgegeben werden können, Anfang Dezember 1999. Die Ausschreibung des Programms, die Gestaltung des Kursangebots sowie die Ankündigung der Kurse erfolgt durch das Frauenbüro in Zusammenarbeit mit den Studierendenvertretungen, den Studiendekanen und den Frauenbeauftragten. Die Verträge werden nach Absprache mit dem Frauenbüro von der Hochschulverwaltung abgeschlossen.

Die Kurse sollen zu Randzeiten (abends, freitagnachmittags oder am Wochenende) stattfinden.

b. Unformatiertes Dokument mit reinem Text

Prof. Dr. Anne Brueggemann-Klein Dr. Angelika Reiser Prof. Dr. Doris Schmitt-Landsiedel Ein Computer-Propaedeutikum von Studentinnen fuer Studentinnen Konzept vom 12.10.1999, Version 1.3 Hintergrund und Ziele Gerae in technischen Studiengaengen spielen Computer als alltaegliches Arbeitsmittel eine wichtige Rolle. Es ist wesentlich, dass die Studierenden gleich zu Beginn des Studiums einen vertrauten und sicheren Umgang mit diesem Handwerkszeug haben. Eventuell vorhandene Hemmschwellen muessen sobald wie moeglich abgebaut werden; davon ist eine studienzeitverkuerzende Wirkung zu erwarten. Es ist bekannt, dass Schuelerinnen weniger Zugang zu einem Computer haben als Schueler und weniger haeufig einen eigenen Computer besitzen. Das Computer-Propaedeutikum

richtet sich deshalb primaer an Studentinnen, die dadurch eventuell fehlende Vorerfahrungen ausgleichen koennen. Das Kursangebot des Computer-Propaedeutikums Das Computer-Propaedeutikum bietet Kurzeinfuehrungen mit praktischen Uebungen in fuer das Studium an der TU Muenchen relevante Themen rund um den Computer. Jeder Kurs dauert als Block angeboten-drei Stunden und beinhaltet eine tutorielle Einfuehrung in das Thema, praktische Uebungen am Rechner und eine Abschlussbesprechung zu den Uebungen. Kursthemen koennen sein: Textverarbeitung mit Word Textverarbeitung mit LaTeX Textverarbeitung mit Star Office E-Mail Browsen und Informationsnutzung im World-Wide Web Erstellen von Informationsangeboten im World-Wide Web (HTML) Betriebssystem, Dateiorganisation Gestaltung des Desktops (Windows-Manager) ASCII-Editoren (vi, emacs) Kauf eines PCs Datenverbindungen von zu Hause (analog, ISDN) Fakultaetsspezifische Informationsinfrastrukturen Jeder Kurs sollte sich auf ein enges Thema konzentrieren. Die Themenliste ist offen fuer Ergaenzungen. Kriterium ist, dass der Kursinhalt propaedeutischer Natur fuer das Studium an der TU ist und eine Form der Computernutzung zum Inhalt hat. Ein Kurs kann fakultaetsuebergreifend oder fakultaetsspezifisch sein. Die Kurse sind in der Regel auf eine spezielle Rechnerplattform (PC/Windows, Unix) zugeschnitten. Die Zielgruppe des Computer-Propaedeutikums Das Kursangebot richtet sich an Studentinnen im ersten Semester. Wird ein Angebot von dieser Zielgruppe nicht ausgeschoepft, steht es auch Studentinnen hoherer Semester und, in dritter Prioritaet, maennlichen Studierenden offen. Die Leiterinnen und Leiter des Computer-Propaedeutikums Die Kurse werden vorzugsweise von Studentinnen hoherer Semester konzipiert und durchgefuehrt. Zur Ergaenzung des Kursangebots koennen auch Mitarbeiterinnen und Mitarbeiter oder Studenten hoherer Semester Kurse anbieten. Nur von Studierenden angebotene Kurse koennen im Rahmen des Programms finanziert werden. Die Kursleiterinnen und -leiter sind selbst fuer die Organisation von Seminar- und Rechnerraeumen und von Zugangsmoeglichkeiten zu den Raeumen am Wochenende fuer ihren Kurs verantwortlich. Wir gehen davon aus, dass die Studiendekaninnen und -dekanen sie bei der Organisation unterstuetzen. Durchfuehrung Das Computer-Propaedeutikum ist eine Initiative der Frauenbeauftragten der TU Muenchen und wird vom Frauenbuero (Kerstin Han-

sen, Anja Quindeau) organisiert. Die TU Muenchen finanziert das Programm aus dem Tutorienprogramm zur Verkuerzung der Studiendauer. Fuer von Studierenden angebotene Kurse wird ein Werksvertrag abgeschlossen, der die Konzeption, Vorbereitung und Durchfuehrung eines dreistuendigen Kurses beinhaltet. Konzeption und Vorbereitung eines Kurses werden mit 200 DM und jede Durchfuehrung mit 100 DM honoriert. Das Frauenbuero erhaelt Hilfskraftmittel aus dem Tutorienprogramm fuer die Organisation des Computer-Propaedeutikums. Wir rechnen mit folgenden Kosten: 20 Kurse, jeder zweimal durchgefuehrt, zu 20 mal 200 DM plus 20mal 2mal 100 DM, also 8000 DM, plus 1000 DM fuer die Organisation im Frauenbuero. Eine erste Ausschreibung des Programms ist bereits erfolgt. Interessierte Kursleiterinnen und -leiter koennen ihr Kursangebot bis zum 5.11.1999 abgeben. Bitte benutzen Sie dazu unser Formular. Die ersten Kurse sollen noch im November 1999 stattfinden. Bei Bedarf erfolgt eine zweite Ausschreibung, zu der neue Kursangebote abgegeben werden koennen, Anfang Dezember 1999. Die Ausschreibung des Programms, die Gestaltung des Kursangebots sowie die Ankuendigung der Kurse erfolgt durch das Frauenbuero in Zusammenarbeit mit den Studierendenvertretungen, den Studiendekanen und den Frauenbeauftragten. Die Vertraege werden nach Absprache mit dem Frauenbuero von der Hochschulverwaltung abgeschlossen. Die Kurse sollen zu Randzeiten (abends, freitagsnachmittags oder am Wochenende) stattfinden

c. Dokumentdarstellung mit optischer Gliederung



Abbildung 7.1 Dokumentdarstellung mit Ikonen

| | |
|--------------------------|---|
| Konzept | |
| Briefkopf | |
| Institution | = Logo fbLogo |
| Autorinnen | |
| | Autorin (3) |
| | 1 Prof. Dr. Anne Brüggemann-Klein |
| | 2 Dr. Angelika Reiser |
| | 3 Prof. Dr. Doris Schmitt-Landsberg |
| (-) Hauptueberschrift | Ein Computer-Propädeutikum von Studentinnen für Studentinnen |
| Nebenueberschrift | |
| | Abc Text Konzept vom |
| (-) Datum | 12.10.1999 |
| Abc Text | , |
| (-) Version | Version 1.3 |
| Abschnitt | |
| | Abschnittueberschrift Hintergrund und Ziele |
| (-) P | Gerade in technischen Studiengängen spielen Computer als alltägliches Arbeitsmittel eine wichtige Rolle. Es ist wesentlich, daß die Studierenden gleich zu Beginn des Studiums einen vertrauten und sicheren Umgang mit diesem Handwerkzeug haben. Eventuell vorhandene Hemmschwellen müssen sobald wie möglich abgebaut werden; davon ist eine studienzeitverkürzende Wirkung zu erwarten. |
| (-) P | Es ist bekannt, daß Schülerinnen weniger Zugang zu einem Computer haben als Schüler und weniger häufig einen eigenen Computer besitzen. Das Computer-Propädeutikum richtet sich deshalb primär an Studentinnen, die dadurch eventuell fehlende Vorerfahrungen ausgleichen können. |
| Abschnitt | |
| | Abschnittueberschrift Das Kursangebot des |

Abbildung 7.2 Dokumentdarstellung mit verschachtelten Boxen

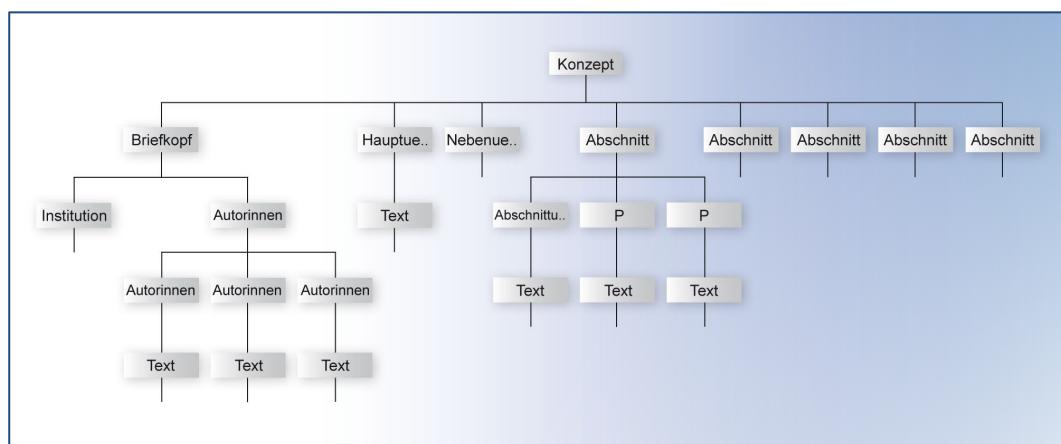


Abbildung 7.3 Dokumentdarstellung als Baum

d. Strukturcodierung mithilfe von eingebetteten Markierungen

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

<Konzept>
<Briefkopf>
<Institution Logo="fbLogo"/>
<Autorinnen>
<Autorin>Prof. Dr. Anne Brügmann-Klein</Autorin>
<Autorin>Dr. Angelika Reiser</Autorin>
<Autorin>Prof. Dr. Doris Schmitt-Landsiedel</Autorin>
</Autorinnen>
</Briefkopf>
<Hauptueberschrift>
Ein Computer-Programmdeutikum von Studentinnen für
Studentinnen
</Hauptueberschrift>
<Nebenueberschrift>
Konzept vom
<Datum>12.10.1999</Datum>
, Version
<Version>1.3</Version>
</Nebenueberschrift>
<Abschnitt>
<Abschnittueberschrift>
Hintergrund und Ziele
</Abschnittueberschrift>
<P>
Gerade in technischen Studiengängen spielen
Computer als alltägliches Arbeitsmittel eine
wichtige Rolle. Es ist wesentlich, dass die
Studierendengleich zu Beginn des Studiums einen
vertrauten und sicheren Umgang mit diesem
Handwerkszeug haben. Eventuell vorhandene
Hemmschwellen müssen sobald wie möglich
abgebaut werden; davon ist eine
studienzeitverkürzende Wirkung zu erwarten.
</P>
<P>
Es ist bekannt, dass Schülerinnen weniger
Zugang zu einem Computer haben als Schüler und
weniger häufig eigenen Computer besitzen.
Das Computer-Programmdeutikum richtet sich deshalb
primär an Studentinnen, die dadurch eventuell
fehlende Vorerfahrungsausgleichen können.
</P>
</Abschnitt>
<Abschnitt>
<Abschnittueberschrift>
```

Das Kursangebot des Computer
Programmdeutikums

</Abschnittueberschrift>

<P>Das Computer-Propädeutikum bietet Kurzeinführungen mit praktischen Übungen in für das Studium an der TU München relevante Themen rund um den Computer. Jeder Kurs dauert als Block angeboten-drei Stunden und beinhaltet eine tutorielle Einführung in das Thema, praktische Übungen am Rechner und eine Abschlu&s;szlig;besprechung zu den Übungen. Kursthemen können sein:</P>

- Textverarbeitung mit Word
 - Textverarbeitung mit LaTeX
 - Textverarbeitung mit Star Office
 - E-Mail
 - Browsen und Informationsnutzung im World-Wide Web
 - Erstellen von Informationsangeboten im World-Wide Web (HTML)
 - Betriebssystem, Dateiorganisation
 - Gestaltung des Desktops (Window-Manager)
 - ASCII-Editoren (vi, emacs)
 - Kauf eines PCs
 - Datenverbindungen von zu Hause (analog, ISDN)
 - Fakultätsspezifische Informationsinfrastrukturen
-

<P>Jeder Kurs sollte sich auf ein enges Thema konzentrieren. Die Themenliste ist offen für Ergünzungen. Kriterium ist, da&s;szlig; der Kursinhalt propädeutischer Natur für das Studium an der TU ist und eine Form der Computernutzung zum Inhalt hat. Ein Kurs kann fakultätsübergeifend oder fakultätsspezifisch sein. Die Kurse sind in der Regel auf eine spezielle Rechnerplattform (PC/Windows, Unix) zugeschnitten.</P>

</Abschnitt>

<Abschnitt>
<Abschnittueberschrift>Die Zielgruppe des Computer-Propädeutikums</Abschnittueberschrift>

<P>Das Kursangebot richtet sich an Studentinnen im ersten Semester. Wird ein Angebot von dieser Zielgruppe nicht ausgeschöpft, steht es auch Studentinnen höherer Semester und, in dritter Priorität, männlichen Studierenden offen.</P>

</Abschnitt>

<Abschnitt>
<Abschnittueberschrift>Die Leiterinnen und Leiter des Computer-Propädeutikums</Abschnittueberschrift>

<P>Die Kurse werden vorzugsweise von Studentinnen höherer Semester konzipiert und durchgeführt. Zur Ergünzung des Kursangebots können auch Mitarbeiterinnen und Mitarbeiter

oder Studenten höherer Semester Kurse anbieten.
Nur von Studierenden angebotene Kurse können im Rahmen
des Programms finanziert werden.</P>

<P>Die Kursleiterinnen und -leiter sind selbst für die
Organisation von Seminar- und Rechnerräumen und von
Zugangsmöglichkeiten zu den Räumen am Wochenende
für ihren Kurs verantwortlich. Wir gehen davon aus,
da&szig; die Studiendekaninnen und -dekanen sie bei der
Organisation unterstützen.</P>

</Abschnitt>

<Abschnitt>

<Abschnittueberschrift>Durchführung</Abschnittueberschrift>

<P>Das Computer-Propädeutikum ist eine Initiative der
Frauenbeauftragten der TU München und wird vom
Frauenbüro (Kerstin Hansen, Anja Quindeau) organisiert.
Die TU München finanziert das Programm aus dem Tutorienprogramm
zur Verküzung der Studiendauer.
Für von Studierenden angebotene Kurse
wird ein Werksvertrag abgeschlossen, der
die Konzeption, Vorbereitung und Durchführung
eines dreistündigen Kurses beinhaltet.

Konzeption und Vorbereitung eines Kurses werden mit 200 DM und jede
Durchführung mit 100 DM honoriert.

Das Frauenbüro erhält Hilfskraftmittel
aus dem Tutorienprogramm für die
Organisation des Computer-Propädeutikums.

Wir rechnen mit folgenden Kosten: 20 Kurse,
jeder zweimal durchgeführt,
zu 20 mal 200 DM plus 20 mal 2 mal 100 DM, also 8000 DM,
plus 1000 DM für die Organisation im Frauenbüro.</P>

<P>Eine erste Ausschreibung
des Programms ist bereits erfolgt.

Interessierte Kursleiterinnen und -leiter
können ihr Kursangebot bis zum 5.11.1999 abgeben.

Bitte benutzen Sie dazu unser
Formular.

Die ersten Kurse sollen noch im November 1999
stattfinden. Bei Bedarf erfolgt eine zweite Ausschreibung,
zu der neue Kursangebote abgegeben werden können,
Anfang Dezember 1999.

Die Ausschreibung des Programms, die Gestaltung des Kursangebots
sowie die Ankündigung der Kurse erfolgt durch das Frauenbüro
in Zusammenarbeit mit den Studierendenvertretungen, den Studiendekanen
und den Frauenbeauftragten. Die Verträge werden nach Absprache
mit dem Frauenbüro von der Hochschulverwaltung abgeschlossen.</P>

<P>Die Kurse sollen zu Randzeiten (Abends, Freitags Nachmittags
oder am Wochenende) stattfinden.</P>

</Abschnitt>

[**</Konzept>**](#)

e. Das Dokument im LaTeX-Format

```
\documentclass{article}
\usepackage[german]{babel}
\usepackage{myPage}

\def\Datum{\today}
\def\Version#1{\bf #1}

\begin{document}

\author{Prof. Dr. Anne Br"uggemann-Klein \and
Dr. Angelika Reiser \and
Prof. Dr. Doris Schmitt-Landsiedel}

\title{Ein Computer-Prop"adeutikum
von Studentinnen f"ur Studentinnen
(Konzept vom \Datum, Version \Version{1.3})}

\maketitle

\section{Hintergrund und Ziele}

Gerade in technischen Studieng"angen spielen
Computer als allt"agliches Arbeitsmittel eine wichtige Rolle.
Es ist wesentlich, da"s die Studierenden
gleich zu Beginn des Studiums einen vertrauten und sicheren Umgang
mit diesem Handwerkszeug haben. Eventuell vorhandene
Hemmschwellen m"ussen sobald wie m"oglich abgebaut werden;
davon ist eine studienzeitverk"urzende Wirkung zu erwarten.

Es ist bekannt, da"s Sch"ulerinnen weniger Zugang zu einem Computer
haben als Sch"uler und weniger h"aufig einen eigenen Computer besitzen.
Das Computer-Prop"adeutikum richtet sich deshalb prim"ar an
Studentinnen, die dadurch eventuell fehlende Vorerfahrungen
ausgleichen k"onnen.

\section{Das Kursangebot des Computer-Prop"adeutikums}

Das Computer-Prop"adeutikum bietet Kurzeinf"ührungen
mit praktischen "Übungen in f"ür das Studium an
der TU M"unchen relevante Themen rund um den Computer.
Jeder Kurs dauert als Block angeboten-drei Stunden
und beinhaltet eine tutorielle Einf"ührung in das Thema,
praktische "Übungen am Rechner und eine
Abschlussbesprechung zu den "Übungen. Kursthemen k"onnen sein:

\begin{itemize}
```

```
\item Textverarbeitung mit Word
\item Textverarbeitung mit LaTeX
\item Textverarbeitung mit Star Office
\item E-Mail
\item Browsen und Informationsnutzung im World-Wide Web
\item Erstellen von Informationsangeboten im World-Wide Web (HTML)
\item Betriebssystem, Dateiorganisation
\item Gestaltung des Desktops (Window-Manager)
\item ASCII-Editoren (vi, emacs)
\item Kauf eines PCs
\item Datenverbindungen von zu Hause (analog, ISDN)
\item Fakult"atsspezifische Informationsinfrastrukturen
\end{itemize}
```

Jeder Kurs sollte sich auf ein enges Thema konzentrieren.

Die Themenliste ist offen f"ur Erg"anzungen.

Kriterium ist, da"s der

Kursinhalt prop"adeutischer Natur f"ur das Studium an der TU ist
und eine Form der Computernutzung zum Inhalt hat.

Ein Kurs kann fakult"ats"ubergreifend

oder fakult"atsspezifisch sein. Die Kurse sind in der Regel auf
eine spezielle Rechnerplattform (PC/Windows, Unix) zugeschnitten.

\section{Die Zielgruppe des Computer-Prop"adeutikums}

Das Kursangebot richtet sich an Studentinnen im ersten Semester.

Wird ein Angebot von dieser Zielgruppe nicht ausgesch"opft, steht es
auch Studentinnen h"ohrer Semester und, in dritter Priorit"at,
m"annlichen Studierenden offen.

\section{Die Leiterinnen und Leiter des Computer-Prop"adeutikums}

Die Kurse werden vorzugsweise von Studentinnen h"ohrer Semester
konzipiert und durchgef"uhrt. Zur Erg"anzung des Kursangebots
k"onnen auch Mitarbeiterinnen und Mitarbeiter
oder Studenten h"ohrer Semester Kurse anbieten.

Nur von Studierenden angebotene Kurse k"onnen im Rahmen
des Programms finanziert werden.

Die Kursleiterinnen und -leiter sind selbst f"ur die
Organisation von Seminar- und Rechner"äumen und von
Zugangsmöglichkeiten zu den Räumen am Wochenende
f"ur ihren Kurs verantwortlich. Wir gehen davon aus,
da"s die Studiendekaninnen und -dekanen sie bei der
Organisation unterstützen.

\section{Durchführung}

Das Computer-Prop"adeutikum ist eine Initiative der
Frauenbeauftragten der TU München und wird vom
Frauenbüro (Kerstin Hansen, Anja Quindeau) organisiert.

Die TU München finanziert das Programm aus dem Tutorienprogramm zur Verkürzung der Studiendauer. Für von Studierenden angebotene Kurse wird ein Werksvertrag abgeschlossen, der die Konzeption, Vorbereitung und Durchführung eines dreistündigen Kurses beinhaltet.

Konzeption und Vorbereitung eines Kurses werden mit 200 DM und jede Durchführung mit 100 DM honoriert.

Das Frauenbüro erhält Hilfskraftmittel aus dem Tutorienprogramm für die Organisation des Computer-Programmabikutums.

Wir rechnen mit folgenden Kosten: 20 Kurse, jeder zweimal durchgeführt, zu 20 mal 200 DM plus 20 mal 2 mal 100 DM, also 8000 DM, plus 1000 DM für die Organisation im Frauenbüro.

Eine erste Ausschreibung des Programms ist bereits erfolgt.

Interessierte Kursleiterinnen und -leiter

können ihr Kursangebot bis zum 5.11.1999 abgeben.

Bitte benutzen Sie dazu unser

Formular. Die ersten Kurse sollen noch im November 1999 stattfinden. Bei Bedarf erfolgt eine zweite Ausschreibung, zu der neue Kursangebote abgegeben werden können, Anfang Dezember 1999.

Die Ausschreibung des Programms, die Gestaltung des Kursangebots sowie die Ankündigung der Kurse erfolgt durch das Frauenbüro in Zusammenarbeit mit den Studierendenvertretungen, den Studiendekanen und den Frauenbeauftragten.

Die Verträge werden nach Absprache mit dem Frauenbüro von der Hochschulverwaltung abgeschlossen.

Die Kurse sollen zu Randzeiten

(Abends, Freitags Nachmittags oder am Wochenende) stattfinden.

\end{document}

f. Prinzipien aus der Informatik

Das Studium dieses Kurses setzt nicht voraus, dass andere Kurse aus dem Bereich der Informatik bereits belegt und erfolgreich absolviert wurden. Bei einem so spezialisierten Wissensgebiet, wie es das Daten- und Dokumentmanagement darstellt, ist es jedoch unumgänglich, einige grundlegende Prinzipien zu verstehen. Die nötigen Basisinformationen folgen diesem Abschnitt.

Spätes Binden

Spätes Binden (engl. *late binding*) bedeutet, Entscheidungen so lange hinauszuzögern, bis sie unumgänglich sind. Das Prinzip der späten Bindung bringt oft einen Gewinn an Flexibilität und wird in ganz unterschiedlichen Teilgebieten der Informa-

tik angewendet. Polymorphie in objektorientierten Programmiersprachen ist ein Beispiel. Polymorphie bedeutet, dass in einer Klassenhierarchie spezialisierte Klassen die Methoden allgemeinerer Klassen, von denen sie erben, umdefinieren dürfen. Erst zur Laufzeit steht für ein jedes Objekt fest, wie es in der Klassenhierarchie einzuordnen ist und welche der polymorph definierten Methoden für das Objekt Gültigkeit haben. Im Zusammenhang mit Formatvorschriften (engl. *stylesheets*) bedeutet der Begriff der späten Bindung (engl. *late binding*) folgendes: Erst wenn ein Dokument präsentiert werden soll, wird ein zum Anwendungszweck passendes Stylesheet zugeschaltet und das grafische Aussehen des Dokuments festgelegt.

Abstraktionsebenen

Abstraktionsebenen (engl. *levels of abstraction*) einzuführen bedeutet, einen komplexen Sachverhalt ausgehend von einer hohen Abstraktionsstufe auf immer niedrigeren Abstraktionsstufen zu beschreiben und auf diese Weise einen konzeptuellen Rahmen herzustellen, in dem der Sachverhalt verstanden werden kann. Eine typische Anwendung des Prinzips ist die Rechnerarchitektur, in der ein Rechensystem auf den Ebenen Hardware, Maschinensprache, Betriebssystem und Anwendungssystem beschrieben wird. Bei DDM kommt das Prinzip der Levels of Abstraction im Zusammenhang mit Codierungen zum Tragen, die im Rahmen des Codierungsmodells auf immer konkreterer Ebene beschrieben werden.

Separation of Concerns

Die Trennung konzeptueller Dimensionen/Ebenen von Zusammenhängen (engl. *separation of concern*) bedeutet, ein System so zu gestalten, dass unterschiedliche Aufgaben unabhängig voneinander durchgeführt werden können. Separation of Concerns erfordert entsprechende Modelle und Architekturen. Ein klassischer Fall von Separation of Concerns sind Datenbankanwendungen. Durch die Trennung in eine Anwendungsschicht, eine konzeptuelle Schicht und eine physikalische Schicht kann unabhängig voneinander an der Anwendungssoftware und an den Speicher- und Zugriffsstrukturen gearbeitet werden. Dies garantiert Datenunabhängigkeit. Separation of Concerns wird durch das Dokumentenmodell erzielt. Unabhängig voneinander können entsprechende Fachleute an der inhaltlichen Gestaltung eines Dokuments arbeiten.

8. Literaturverzeichnis

- [AFQ89] J. André, R. Furuta, and V. Quint. Structured Documents. The Cambridge Series on Electronic Publishing. Cambridge University Press, Cambridge, 1989.
- [DYIWFT02] M. J. Dürst, F. Yergeau, R. Ishida, M. Wolf, A. Freytag, and T. Texin. Character model for the World Wide Web 1.0. www.w3.org/TR/charmod, February 2002. W3C Working Draft.
- [E63] D. Engelbart. A conceptual framework for the augmentation of man's intellect. In P. W. Howerton and D. C. Weeks, editors, *Vistas in Information Handling*, volume 1, pages 1–29, Washington, DC, 1963. Spartan Books.
- [FHVV11] T. Feuerstack, A. Hartmann, B. Vogeler, and J. Vieler. Einmal um die Erde und zurück: Unterwegs im Internet. Broschüre A/003/0811, Universitätsrechenzentrum FernUniversität Hagen, August 2011.
- [FSS82] R. Furuta, J. Scofield, and A. Shaw. Document formatting: Survey, concepts, and issues. *Computing Surveys*, 14(3):417–172, September 1982
- [MvD82] N. Meyrowitz and A. van Dam. Interactive editing systems (Parts I and II). *Computing Surveys*, 14(3):321–415, September 1982.
- [WD00] K. Whistler and M. Davis. Character encoding model. Unicode Technical Report 17-3.1, Unicode, Inc., August 2000.
- [P01] Peter Constable. Character set encoding basics. NRSI: Computer & Writing Systems, 2001.
- [Kuh89] Rainer Kuhlen. Pragmatischer Mehrwert von Information. Sprachspiele mit informationswissenschaftlichen Grundbegriffen. Universität Konstanz / Informationswissenschaft: Bericht;89,1. Konstanz: Universität Konstanz, 1989. URL: <http://nbn-resolving.de/urn:nbn:de:bsz:352-247844>.
- [Wil95] Patrick Wilson. “Unused relevant information in research and development”. In: *Journal of the American society for Information Science* 46.1 (1995), pp. 45–51. ISSN: 0002-8231
- [NT95] I. Nonaka and H. Takeuchi, *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*, Oxford University Press, 1995.
- [Vu15] B. Vu, "Realizing an Applied Gaming Ecosystem - Extending an Education Portal Suite towards an Ecosystem Portal," TU Darmstadt, 2015.

- [Bat02] Marcia J. Bates. "Toward an integrated model of information seeking and searching". In: *The New Review of Information Behaviour Research* 3.1 (2002), pp. 1–15.
- [Bat10] Marcia J. Bates. "Information behavior". In: *Encyclopedia of library and information sciences* 3 (2010). Publisher: CRC press New York, pp. 2381–2391.
- [Bat79] Marcia J. Bates. "Information search tactics". In: *Journal of the American Society for information Science* 30.4 (1979). Publisher: Wiley Online Library, pp. 205–214.
- [Bat89] Marcia J. Bates. "The design of browsing and berrypicking techniques for the online search interface". In: *Online review* 13.5 (1989), pp. 407–424.
- [Mar89] Gary Marchionini. "Information-seeking strategies of novices using a full-text electronic encyclopedia". In: *Journal of the American Society for Information Science* 40.1 (1989). Publisher: Wiley Online Library, pp. 54–66.
- [MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. New York: Cambridge University Press, 2008. ISBN: 978-0-521-86571-5.
- [Naw21] Christian Nawroth, Supporting Information Retrieval of Emerging Knowledge and Argumentation, Dissertation, FernUniversität in Hagen, eingereicht zur Begutachtung und Veröffentlichung, 2021
- [BOB82] BELKIN, N.J., ODDY, R.N. and BROOKS, H.M. (1982), "ASK FOR INFORMATION RETRIEVAL: PART I. BACKGROUND AND THEORY", *Journal of Documentation*, Vol. 38 No. 2, pp. 61-71. <https://doi.org/10.1108/eb026722>

9. Abbildungsverzeichnis

| | |
|---|----|
| Abbildung 4.1 Modi der Informationssuche [Bat02]..... | 11 |
| Abbildung 4.2 Das SECI Model [NT95] | 12 |
| Abbildung 4.3 Webseitengestaltung von 1995..... | 13 |
| Abbildung 5.1 Der Zeichensatz ISO 646-US (US-ASCII)..... | 27 |
| Abbildung 5.2 Die deutsche Variante ISO 646-DE von ISO 646 | 28 |
| Abbildung 5.3 Der Zeichensatz ISO 8859-1 (ISO Latin-1)..... | 28 |
| Abbildung 5.4 Der Windows-Zeichensatz 1252 | 29 |
| Abbildung 5.5 Typografie und Auswirkungen | 31 |
| Abbildung 5.6 Unicode-Code-Raum mit Surrogatpositionen und privatem Bereich | 34 |
| Abbildung 7.1 Dokumentdarstellung mit Ikonen | 49 |
| Abbildung 7.2 Dokumentdarstellung mit verschachtelten Boxen..... | 50 |
| Abbildung 7.3 Dokumentdarstellung als Baum..... | 50 |

10. Tabellenverzeichnis

| | |
|---|----|
| Tabelle 5.1 Die Ersetzungen in der deutschen Variante von ISO 646..... | 28 |
| Tabelle 5.2 Das Verhältnis von abstrakten Zeichen und Glyphen | 32 |
| Tabelle 5.3 Die sechzehn Hex-Ziffern und ihre Repräsentationen als Bitmuster..... | 34 |
| Tabelle 5.4 Code-Räume für verschiedene Codierungen | 34 |
| Tabelle 5.5 Codierungsformate für verschiedene Codierungen..... | 37 |
| Tabelle 5.6 Grunddaten zu Codierungseinheiten | 39 |

11. Lösungen der Selbsttestaufgaben

Selbsttestaufgabe 1.1

Erläutern Sie, wie von einer Suchmaschine Kuhlens „Primacy of Information Work“ bei einer Suche nach dem Stichwort „NASA Perseverance“ umgesetzt wird. Die Suche wird von einem Vater ausgeführt, um Fragen seiner Tochter zu der Mission zu beantworten.

- Die Suchanfrage wird durch ein Informationsbedürfnis auf Nutzerseite ausgelöst, da dieser die benötigte Information nicht hat, um das Problem (Nachfrage der Tochter) zu lösen, wie von Kuhlen skizziert („needed by but not needed by but not available to a specific person in a concrete situation to solve a problem“).
- Das dafür benötigte Wissen liegt in einer Vielzahl von Quellen (z.B. NASA Homepage, Wissenschaftliche Portale, Nachrichtenseiten) prinzipiell vor, steht dem Nutzer aber nicht zur Verfügung.
- Durch die Nutzung einer Suchmaschine und Zugriff auf die Ergebnisse erfolgt die Transformation des Wissens in Information für den Vater, die dieser wiederum Nutzen kann, um seiner Tochter zu berichten.
- Damit findet auf diese Weise die Transformation von Wissen in Aktion (=Information) statt, was das Primacy of Information Work nach Kuhlen darstellt.

Selbsttestaufgabe 1.2

Überlegen Sie, ob auf die folgenden Dokumente eher der ursprüngliche, der neuere oder der moderne Dokumentenbegriff passt oder ob es sich vielleicht gar nicht um Dokumente handelt.

- Personalausweis
- Geldschein
- Gemälde
- Veranstaltungsplakat
- Privatbrief
- E-Mail-Nachricht
- Memo an die Geschäftsleitung
- Gedichtband
- Roman
- Bestellung
- Produktbeschreibung
- Fragebogen
- Nachricht in einem Protokoll zum digitalen Zahlungsverkehr (Electronic Banking)

Bei Gemälden handelt es sich wegen des fehlenden sprachlichen Ausdrucks nicht um Dokumente.

Personalausweise und Geldscheine sind eindeutig Dokumente ursprünglicher Art, für die das Trägermedium ebenso essenziell ist wie der sprachliche Informationsgehalt.

Veranstaltungsplakate, Privatbriefe, Gedichtbände und Romane sind wohl in der Regel Dokumente neuerer Art, in denen neben dem Informationsgehalt auch die Formatierung eine Rolle spielt.

Limitierte Auflagen und handschriftliche Briefe mit Unterschrift könnten wir jedoch genauso gut als Dokumente ursprünglicher Art einordnen.

Memos, Bestellungen, Produktbeschreibungen und Fragebögen sehe ich je nach Kontext als Dokumente neuerer oder moderner Art an. Ein wichtiges Entscheidungskriterium ist, ob diese Dokumente digital oder auf Papier vorliegen.

E-Mail-Nachrichten oder Nachrichten in einem Protokoll zum digitalen Zahlungsverkehr sind Nachrichten moderner Art, solange man den Protokollnachrichten den sprachlichen Ausdruck zugesteht.

Selbsttestaufgabe 1.3

Welche Methoden, mit denen implizites Wissen in explizites Wissen umgewandelt werden kann, beschreibt das SECI- Modell?

1. **Sozialisation** (implizit zu implizit): Menschen kommunizieren direkt mit Menschen oder wenden den Wissensaustausch durch Erfahrungsaustausch an. Heutzutage werden viele fortschrittliche Technologien eingeführt, aber das Sozialisieren bleibt wichtig und kann nicht ersetzt werden.
2. **Externalisierung** (implizit bis explizit): Zusammen mit der Erfindung des Schreibens kann Wissen vom Gehirn auf andere Medien wie z.B. Bücher, Videos usw. Dieser Vorgang wird als Externalisierung bezeichnet.
3. **Kombination** (explizit zu explizit): Dieser Prozess besteht darin, das Wissen zu organisieren und Teile davon in neuen Formen zu kombinieren, damit es sinnvoller ist oder neue Erkenntnisse erklärt.
4. **Internalisierung** (explizit zu implizit): Personen können neues Wissen erwerben, indem sie explizites Wissen wahrnehmen. Einmal verinnerlicht, kann neues Wissen wiederverwendet, dokumentiert oder an andere Personen weitergegeben werden.

Selbsttestaufgabe 1.4

Wie unterscheiden sich die im Kurstext beschriebenen Konzepte zur grafischen Markierung der Strukturelemente eines Dokuments gegenüber der Markierung durch Auszeichnung?

Übereinstimmend wird in den Konzepten davon ausgegangen, dass die logische Strukturierung hierarchisch angelegt ist. Bei grafischen Konzepten wird die hierarchische Struktur grafisch dargestellt. Es gibt drei Varianten:

- Bei der ersten Variante werden ineinander geschachtelte Rechtecke oder Boxen, die mit dem Namen des jeweiligen Elementes gekennzeichnet sind, genutzt.
- Die zweite Variante verwendet zur Darstellung der Struktur Einrückungen und Icons.
- Die dritte Variante besteht darin, die logische Struktur als einen Baum darzustellen.

Konzepte, die auf Markup basieren, bestehen darin, die hierarchische Struktur mittels einer sogenannten Markierung (engl. *tag*) im Text zu versehen: jeder Textbereich, der Inhalt eines logischen Elements ist, wird an seinem Anfang und an seinem Ende mit einer öffnenden und einer schließenden Markierung versehen, die den Namen des entsprechenden Strukturelementes enthält.

Der Oberbegriff zu Markierungen und eventuellen weiteren Möglichkeiten, nicht zum Inhalt zählende Zeichen in einen Text einzustreuen, ist der Begriff Auszeichnung (engl. *markup*).

Selbsttestaufgabe 1.5

Beschreiben Sie die drei Ebenen des Modells der strukturierten Dokumente, die im Kurstext genannt werden, in eigenen Worten.

Die erste Ebene des Modells der strukturierten Dokumente bezieht sich auf den Inhalt eines Dokuments. Gemeint sind damit die rein textuellen Inhalte in Form von Buchstaben und Zeichen, die beim Verarbeiten / Lesen zu Wörtern zusammengesetzt und mit Bedeutung versehen werden. (Inhaltsebene)

Die zweite Ebene (Strukturebene) bezeichnet die explizite Repräsentation der Struktur des Dokuments, die über das Einfügen von z. B. Markup realisiert wird.

Darauf aufbauend folgt die dritte Ebene im Modell der strukturierten Ebene (Formatebene), die sich auf die Formatierung der bereits strukturierten Inhalte bezieht. Das zu den Textelementen passende Format kann aus einer separaten Formatvorlage, auch Stylesheet genannt, erzeugt werden.

Selbsttestaufgabe 1.6

Was versteht man unter dem Begriff "Glatter Text"? Wozu benötigt man in diesem Zusammenhang eine Zeichencodierung?

Sogenannter „Glatter Text“ ist eine Folge von elementaren Texteinheiten wie Buchstaben, Ziffern, Interpunktionszeichen und anderen Zeichen. Er enthält keinerlei Formatierungsangaben oder sonstige spezielle Steuerzeichen der Textcodierung oder Markierungen bezüglich der Strukturauszeichnung.

Eine Zeichencodierung gibt die Vorschrift dazu an, wie diese Zeichen in Bits und Bytes umgesetzt werden.

Selbsttestaufgabe 1.7

Welches sind die Bestandteile des im Kurstext beschriebenen generischen Zeichensatzmodells? Erläutern Sie diese kurz in eigenen Worten.

Die einzelnen Bestandteile des Codierungsmodells sind:

- ein abstrakter Zeichensatz,
- eine Codetabelle,
- ein Codierungsformat,
- ein Codierungsschema und
- eine Übertragungssyntax

Die obere Ebene des Codierungsmodells ist der abstrakte Zeichensatz. Dieser besteht aus einer Menge sogenannter abstrakter Zeichen. Es handelt sich hierbei um Buchstaben, Ziffern, Interpunktionszeichen, Akzente, grafische Symbole etc.

An nächster Stelle des Codierungsmodells steht die Codetabelle, die den abstrakten Zeichen im Zeichensatz eine Codeposition zuweist. Eine Codeposition ist immer eine natürliche Zahl, also größer oder gleich 0. Der Code-Raum kann jedoch auch Zahlen enthalten, die keine zulässigen Codepositionen sind. Codepositionen werden üblicherweise in Hexadezimalnotation angegeben.

Mithilfe des Codierungsformats für eine Codetabelle werden die Bitrepräsentationen für die Codeposition festgelegt. Dazu gibt es eine Code-Einheit, die in der Regel aus acht oder sechzehn Bits besteht. Durch das Codierungsformat werden dann die Positionen eines Code-Raums in Sequenzen von Code-Einheiten und somit in Bitmuster abgebildet.

Ein Codierungsschema wird nun dazu genutzt zu einem Codierungsformat zusätzlich festzulegen, wie die Codierungseinheiten in Bytefolgen zu serialisieren sind.

Ist die Codierungseinheit eines Codierungsformats selbst acht Bits lang, so ist nichts mehr festzulegen und das Codierungsschema ist mit dem Codierungsformat identisch. So kann also UTF8 sowohl als Codierungsformat als auch als Codierungsschema bezeichnet werden.

Zur Abgrenzung von Markup und Inhalt werden bestimmte Funktionszeichen eingesetzt, die dann außerhalb ihrer syntaktischen Rolle weder im Markup selbst noch im Inhalt des Klartextes vorkommen dürfen. In XML ist < ein solches Funktionszeichen, das den Anfang eines Tags charakterisiert. Das Zeichen < darf deswegen im Inhaltstext eines XML- Dokuments nicht vorkommen.

000 000 000 (00/23)

00000-0-00-S 1

Prof. Dr. Matthias Hemmje

01877

Dokumenten- und Wissensmanagement im Internet

Kurseinheit 6:
Semantische Integration

Fakultät für
Mathematik und
Informatik

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere das Recht der Vervielfältigung und Verbreitung sowie der Übersetzung und des Nachdrucks, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder ein anderes Verfahren) ohne schriftliche Genehmigung der FernUniversität reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Wir weisen darauf hin, dass die vorgenannten Verwertungsalternativen je nach Ausgestaltung der Nutzungsbedingungen bereits durch Einstellen in Cloud-Systeme verwirklicht sein können.

Der Inhalt dieses Studienbriefs wird gedruckt auf Recyclingpapier (80 g/m², weiß), hergestellt aus 100 % Altpapier.

Inhaltsverzeichnis

| | |
|---|-----|
| Inhaltsverzeichnis..... | III |
| 1 Einleitung | 4 |
| 2 Semantische Integration: Konflikte und Lösungsansätze | 4 |
| 3 Heterogenitätskonflikte zwischen Daten-Instanzen | 9 |
| 4 Lösungsansätze für die Semantische Integration | 12 |
| 5 Semantische Modelle darstellen und vergleichen | 21 |
| 6 Zusammenfassung | 25 |
| 7 Literaturverzeichnis..... | 28 |
| 8 Abbildungsverzeichnis | 30 |
| 9 Lösungen der Selbsttestaufgaben | 31 |

1 Einleitung

Dies ist die sechste Kurseinheit des Kurses 1877 „Dokumenten- und Wissensmanagement im Internet“.

Die Kursteilnehmenden beurteilen des Weiteren die Heterogenität von verteilten Informationsquellen im Web hinsichtlich Herausforderungen bei deren Integration und stellen Heterogenitätskonflikte auf syntaktischer, struktureller und semantischer Ebene gegenüber. Der Integration auf semantischer Ebene kommt dabei eine besondere Bedeutung zu.

Um diese Konflikte zu beheben, existieren verschiedene Lösungsansätze, die von den Studierenden untersucht und in ihrer Umsetzung hinsichtlich des Ziels eines semantischen Webs beurteilt werden. In diesem Zusammenhang werden auch unterschiedliche semantische Modelle dargestellt und verglichen.

2 Semantische Integration: Konflikte und Lösungsansätze

Die Integration heterogener Datenquellen erfordert mehr als nur das Darstellen der Daten in einer gemeinsamen Syntax. Die oben beschriebenen syntaktischen Standards ermöglichen zwar die einheitliche Abbildung und Strukturierung von Informationen im Web, wodurch die automatische Verarbeitung sowohl von lokal vorliegender Information als auch aus entfernten Quellen stammender Information erheblich erleichtert wird.

Die unterschiedliche Struktur und Semantik von Informationen aus unterschiedlichen Quellen, wie sie im Web vorzufinden sind, führen jedoch zu differierenden Problemen bei der Informationsintegration, die das Eingreifen auf mehreren Ebenen erforderlich machen [BEF01].

Heterogenitätskonflikte auf unterschiedlichen Ebenen

Syntaktische Homogenität ist eine notwendige, aber nicht hinreichende Bedingung für die gemeinsame Informationsnutzung.

Abstrakt betrachtet können die oben beschriebenen Heterogenitätskonflikte folgenden Ebenen zugeordnet werden:

- Datenmodellebene
- Datenschemaebene
- Dateninstanzebene

In den folgenden Abschnitten werden diese Konflikte näher beleuchtet.

Heterogenitätskonflikte zwischen Datenmodellen

Datenquellen können sich deutlich in der Darstellung von Daten unterscheiden (z. B. als Tabellen, Objekte, Dateien usw.). Hiermit wird die syntaktische Ebene angesprochen. Das Abgleichen heterogener Datenquellen macht ein gemeinsames Datenmodell erforderlich, worauf die Informationen aus den unterschiedlichen Quellen abgebildet werden können. Es müssen geeignete Transformationen gefunden werden, um die Daten in das entsprechende Modell zu überführen (XSLT, XQuery). Abbildung 2.1 soll dies veranschaulichen:

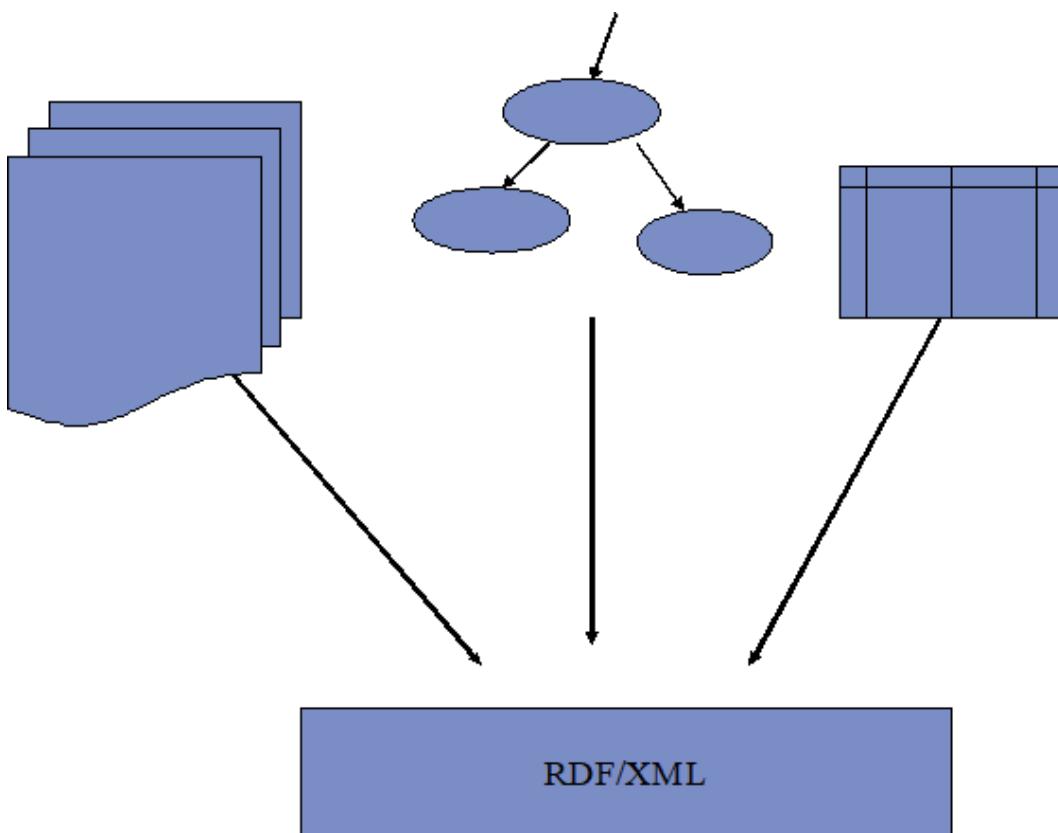


Abbildung 2.1 Heterogenität auf syntaktischer Ebene

Mit RDF in Verbindung mit RDF-S und XML/XML-S als Beschreibungssprache steht ein mächtiges Werkzeug zur Datenmodellierung zur Verfügung.

Selbsttestaufgabe 6.1:

Auf welchen drei Ebenen können Heterogenitätskonflikte auftreten?

Heterogenitätskonflikte zwischen Datenschemata

Auf der strukturellen Ebene stellt sich das Abgleichen unterschiedlicher schematischer Repräsentationen ein und desselben Objekts oder Merkmals als problematisch dar. Dabei unterscheidet man die nachstehenden Konflikte [SvH05]:

- bilaterale Konflikte

- multilaterale Konflikte
- metalevel Konflikte

Zur Verdeutlichung dieser Konflikte soll im Folgenden ein einfaches Beispiel zur Veranschaulichung dienen. Ein Hotel sei in einer bestimmten Datenquelle durch folgende Repräsentation als RDF-Tripel beschrieben:

```
(http://www.hotels.com#42 name "Amstel Hotel")
(http://www.hotels.com#42 category luxury)
(http://www.hotels.com#42 location "Amsterdam,
Netherlands")
(http://www.hotels.com#42 priceSingle 250)
(http://www.hotels.com#42 priceDouble 350)
```

Abbildung 2.2 Beispiel 4 [MCC05]

Dabei gilt es zu beachten, dass in anderen Quellen dasselbe Hotel in differierender Form repräsentiert werden kann.

Bilaterale Konflikte

| Bilaterale Konflikte

| Bilaterale Konflikte betreffen in der Regel genau ein Objekt. Dieses eine Objekt wird in unterschiedlichen Informationsquellen durch unterschiedliche, beschreibende Strukturen abgelegt. Es wird dabei zwischen diesen Konflikten unterschieden:

- Namenskonflikte (Bezeichnerkonflikte)
- Datentypkonflikte
- Integritätskonflikte

| Namenskonflikte

| Namenskonflikte treten in allen Fällen auf, in denen Quellen unterschiedliche Namen für dasselbe Objekt der realen Welt (real world object) verwenden. Ein typischer Fall ist die Verwendung unterschiedlicher Namen für das gleiche Attribut in relationalen Datenbanken (Synonyme).

Ein solcher Konflikt ist in unserem oben dargestellten RDF-Beispiel möglich: so könnten etwa andere Quellen andere Bezeichnungen für die Kategorie (category, siehe Abbildung 2.2, Zeile 2) eines Hotels wie z. B. „Klasse“, „Sterne“ o. ä. verwenden. Ebenso können Homonyme, also syntaktisch gleiche Bezeichner, aber semantisch unterschiedliche Relationen, zu Namenskonflikten führen.

| Datentypkonflikte

| RDF unterstützt die Benutzung von XML-Schema-Datentypen, um die einem Objekt zugewiesenen Werte zu repräsentieren. Als *Datentypkonflikt* bezeichnet man den Fall, dass verschiedene Datentypen für denselben Wert verwendet

werden, z. B. der Preis für eine Unterkunft einmal im integer-, in einer anderen Datenquelle jedoch im real- oder string-Format angegeben wird. Dies erschwert etwa Vergleichsoperationen.

Ressourcen werden in RDF durch eine eindeutige Bezeichnung, z. B. eine URI gekennzeichnet (in unserem Beispiel <http://www.hotels.com#42>). Dasselbe Objekt kann in einer anderen Repräsentation jedoch durchaus eine andere Bezeichnung tragen, z. B. <http://www.vacation.org/hotels#666>. Die Verwendung unterschiedlicher Identifikatoren für dasselbe Objekt erschwert es, Information über das Objekt aus verschiedenen Quellen zusammenzufassen und führt zum Integritätskonflikt.

Integritätskonflikte

Multilaterale Konflikte

Multilaterale Konflikte sind Konflikte, die mehr als ein Objekt einer Repräsentation berühren. Sie treten auf, wenn eine Information, welche in einer Quelle durch ein einzelnes Objekt repräsentiert wird, in einer anderen Quelle auf mehrere Objekte verteilt ist. Dies führt zu Konflikten, wobei hier drei Konflikttypen unterschieden werden:

multilaterale Konflikte

- multilateral attribute correspondences
- multilateral entity correspondances
- missing values

Als *multilateral attribute correspondances* werden Konflikte bezeichnet, die durch Verteilung von Informationen auf mehrere Eigenschaften (Prädikate) auftreten. In dem Beispiel ist die Information, wo sich die Unterkunft befindet, über die Eigenschaft location beschrieben.

multilateral attribute correspondances

In anderen Quellen könnte dieselbe Information durch die zwei Eigenschaften city und country beschrieben werden. Wird eine Unterkunft an einem bestimmten Ort gesucht, muss die entsprechende Information der verschiedenen Quellen entweder aufgespalten oder kombiniert werden, um sie vergleichbar zu machen (matchen).

Als *multilateral entity correspondances* bezeichnet man Konflikte, die entstehen, wenn einzelne oder mehrere Ressourcen verwendet werden, um eine bestimmte Information darzustellen. In dem vorliegenden Beispiel ist die Information über die Unterkunft und deren Lage in der Beschreibung einer Ressource zusammengefasst. Der Attributwert bzgl. der Lage des Hotels ist hier als Literalwert angegeben. In anderen Quellen könnten spezielle Ressourcen als einmalige Repräsentationen der Lage, wie zum Beispiel cities und countries, dienen:

multilateral entity correspondances

(

(

Abbildung 2.3 multilateraler Konflikt

Dieser Zusammenhang wird in Abbildung 2.4 noch einmal grafisch veranschaulicht.

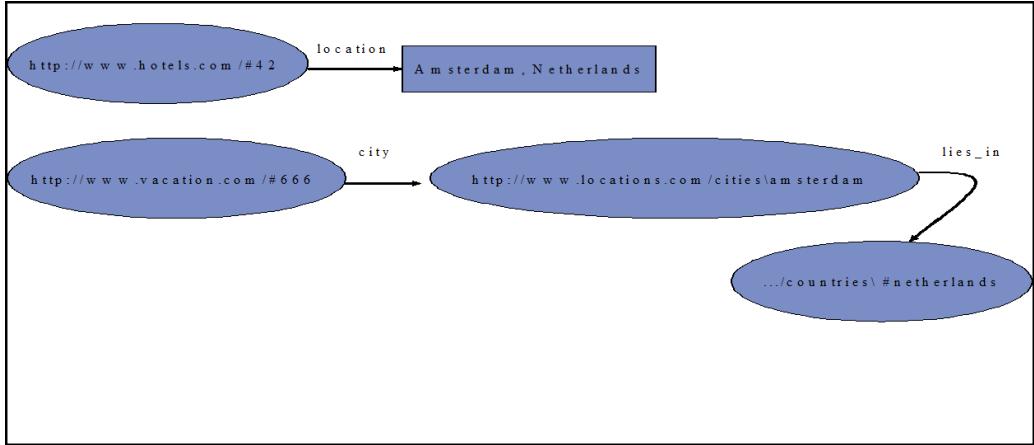


Abbildung 2.4 Grafische Darstellung des multilateralen Konfliktes

missing values

Als *missing values* werden Konflikte bezeichnet, die entstehen, wenn bestimmte Teile einer Information, die in einer Informationsquelle enthalten sind, in einer anderen Informationsquelle fehlen. In dem oben angeführten Beispiel gibt es Informationen über Einzel- und Doppelzimmer, in anderen Quellen u. U. nur über Doppelzimmer. Dies kann entweder bedeuten, dass es keine Einzelzimmer gibt, dass der Preis identisch ist oder dass diese spezielle Information der Repräsentation nicht eingelesen worden ist.

Meta-Level-Konflikte

Meta-Level-Konflikte

Diese Konflikttypen sind bedingt durch die Verwendung unterschiedlicher Modellierungselemente zur Repräsentation von Information derselben Art. In konzeptionellen Datenmodellen sind diese Grundelemente Dateneinheiten (entities), Attribute und Daten; in RDF-Ressourcen sind es Merkmale und Literale. Die Vermischung dieser Modellierungselemente bei der Repräsentation kann zu folgenden Konflikten führen:

- Im vorigen Absatz ist deutlich geworden, dass man location sowohl als Resource als auch als Literal repräsentieren kann.
- Außerdem kann eine Situation vorkommen, in welcher dieselbe Information, die in einer Quelle explizit als Relation codiert ist (netherlands part of europe), in einer anderen implizit im Typ der Ressource angegeben ist (netherlands type european country).

- Darüber hinaus können Datenquellen dieselbe Information beinhalten, aber unterschiedliche Datenstrukturen verwenden. XML-Dokumente können z. B. Elemente in verschiedenen Konstellationen zusammenstellen [RAV03] oder es werden unterschiedliche Wege beschritten, um ein und dieselbe Information darzustellen etwa die Altersangabe in „Alter“ vs. „Geburtsdatum“ [BEF01].

Selbsttestaufgabe 6.2:

Fassen Sie mit eigenen Worten zusammen was ein „Meta-Level Konflikt“ ist.

3 Heterogenitätskonflikte zwischen Daten-Instanzen

Auf semantischer Ebene stellt sich das Entscheidungsproblem, ob verschiedene Objekte aus unterschiedlichen Quellen dasselbe so genannte *real world object* repräsentieren. Ebenso ist danach zu fragen, ob eine gemeinsame Quelle ausgewählt wird, falls widersprüchliche Informationen, etwa durch einfache (Tipp-)Fehler erzeugt, gefunden werden [RAV03]. Es muss also deren Semantik betrachtet werden, da die beabsichtigten Interpretationen der Repräsentationen von Quelle zu Quelle differieren können. Es kann hier zwischen zwei Konflikttypen unterschieden werden. *Datenkonflikte* gehen aus unterschiedlicher Codierung hervor und *Domänenkonflikte* erwachsen aus der unterschiedlichen Konzeptionierung des jeweiligen Wissensbereiches [SvH05], wie hier nun noch näher erläutert wird.

Eine Domäne sollte möglichst kompakt und vergleichbar repräsentiert werden.

Datenkonflikte

Hierzu werden die wichtigen Eigenschaften der relevanten Objekte beschrieben, indem den einzelnen Merkmalstypen entsprechende Werte zugeordnet werden, die häufig vom konkreten Wert abstrahieren. Je nach Abstraktionsgrad ergeben sich jedoch beim Vergleich verschiedener Repräsentationen u. U. unterschiedliche Auswahlmöglichkeiten entsprechend dieses Werte-Systems, da jede Informationsquelle einen eigenen Maßstab des jeweiligen Werts oder einen eigenen Wertebereich verwendet.

Im Einzelnen spricht man von unterschiedlichen Skalen (*different scales*), wenn insbesondere numerische Werte auf verschiedenen Maßstäben basieren. Ein Beispiel dafür könnte die Angabe des Preises einer Unterkunft je nach Repräsentation in unterschiedlichen Währungen sein. Hinzu kommt, dass die Beziehung zwischen den unterschiedlichen Maßstäben eines Werts (den Währungen) nicht unbedingt konstant sein muss (fester Umrechnungskurs €«DM vs. variabler Umrechnungskurs €«\$), sodass sie ggf. zum Vergleichszeitpunkt ermittelt werden muss, wie in der folgenden Abbildung durch die Herausstellung von Datenkonflikten wegen unterschiedlichen Maßstabs (Preise) oder Skalierung (Kategorie) sowie Sterne vs. „luxury“ (*surjective mapping*) verdeutlicht werden soll.

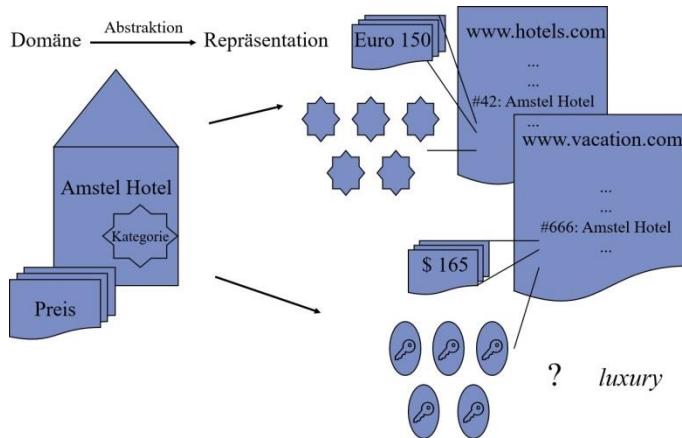


Abbildung 3.1 Datenkonflikte

Eine weitere Problematik liegt in der abhängig von der Quelle differierenden Skalierung der Werte. Betrachtet man dies in Bezug auf das Beispiel aus Abbildung 2.2, so ergibt sich daraus folgender Konflikt: Die Qualität eines Hotels in Mitteleuropa wird auf einer Skala von einem bis zu fünf Sternen gemessen, in Spanien hingegen verwendet man die Anzahl von Schlüsseln als Qualitätsangabe. Dieser Fall wird

different value ranges als different *value ranges* bezeichnet. Besitzt man keine Kenntnis über die zugrunde liegende Skala, ist auch kein Vergleich der Merkmale bzw. deren Abstraktion als Werte möglich.

Wenn ein Wert einer Quelle auf mehrere Werte der anderen Quelle abgebildet wird, spricht man von dem Konflikttypen des surjective mappings. In unserem Beispiel kann die Kategorie *luxury* etwa vier bis fünf Sternen entsprechen. Das Problem liegt demnach darin, dass die der Abstraktion zugrunde liegenden Wertesysteme nicht dieselbe Anzahl an Werten besitzen. Dabei ist es normalerweise nicht möglich, zu entscheiden, ob das Merkmal *luxury* als vier oder fünf Sterne angesehen werden soll. Werden unterschiedliche Repräsentationen für dieselbe Information verwendet, kann die Konvertierung mithilfe von header-Informationen (in XML-Dokumenten) durchgeführt werden. Ein Beispiel dazu könnte die Verwendung sowohl unterschiedlicher Zeichenketten in XML-Dokumenten wie „1“ versus „sehr gut“ als auch unterschiedlicher Zeichensätze sein.

Die Identifizierung von Objekten bzw. Datenobjekten über Dokumentengrenzen hinweg kann auf Grundlage von ID-Attributen via Referenzierung stattfinden (vgl. XLink, XPointer, [RAV03]). Die Anwendungen, welche die entsprechenden Dokumente erstellen, müssen jedoch sicherstellen, dass diese Mechanismen richtig verwendet werden. Beim Vergleich voneinander unabhängiger Dokumente müssen Elemente, die entweder in Beziehung zueinander stehen oder dasselbe beschreiben, schon während der Integration bestimmt werden. Dieser Konfliktfall wird als schema mapping oder schema matching bezeichnet.

different value ranges

surjective mapping

schema mapping

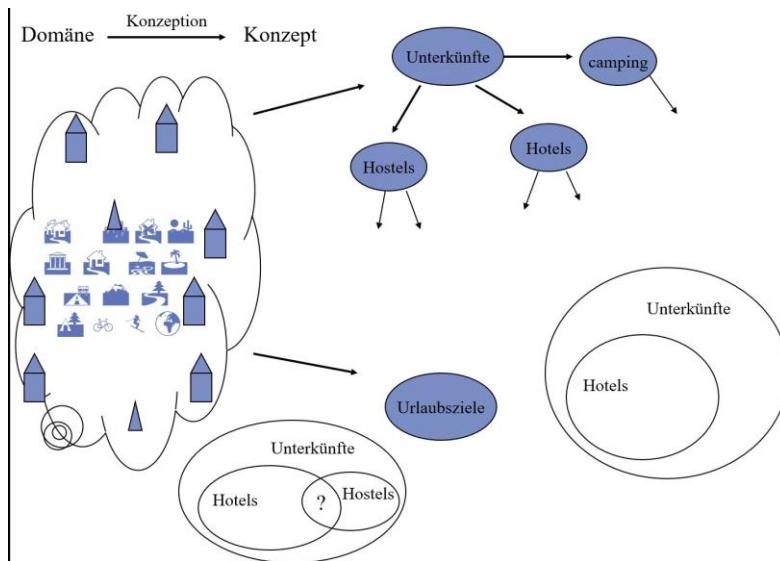


Abbildung 3.2 Domänenkonflikte

Die Abbildung zeigt im oberen Teil grafisch auf, wie Domänenkonflikte: (subsumption) aussehen. Im linken Teil der Abbildung wird ein *overlap* dargestellt.

Selbsttestaufgabe 6.3:

Was ist unter „Surjective Mappings“ zu verstehen?

Domänenkonflikte

Domänenkonflikte treten auf, wenn sich verschiedene Konzeptualisierungen bestimmter Wissensbereiche der realen Welt widersprechen und somit ein Vergleich unmöglich wird. Diese Art von Konflikten tritt besonders zutage, wenn Objekte und Klassen innerhalb des Abstraktions-Prozesses durch Kategorien voneinander abgetrennt werden, obwohl die Grenzen zwischen den entstehenden Kategorien in der Realität so nicht existieren [DOE05]. Normalerweise werden Objekte in Klassen zusammengefasst, wenn diesen bestimmte Eigenschaften gemeinsam sind, die dann nicht mehr explizit auftauchen. Dadurch wird das Auffinden von Relationen zwischen diesen Kategorisierungen erschwert. Es kann so zu unerwünschter Zusammenfassung, Überlappung und Inkonsistenz von Klassen kommen. Als Domänenkonflikte kann man daher zwei Typen unterscheiden:

Domänenkonflikte

Zum einen ist hier der Fall der Subsumtion zu nennen. Dieser liegt vor, wenn eine Klasse von Objekten alle Objekte, die in einer anderen Klasse enthalten sind, einbezieht. Für das angeführte Beispiel bedeutet dies, dass die Klasse accommodation sowohl alle accommodation-Objekte als auch die Klasse hotels umfasst. Zwar sind alle Hotels auch Unterkünfte, und bei einer Suche nach Unterkünften sollen auch alle Hotels gefunden werden. Eine entsprechende Anfragesprache muss jedoch in der Lage sein, das Datenmodell und die semantischen Eigenschaften des Vokabulars, im Beispiel das Konzept von Unterklassen, zu „verstehen“.

Der zweite Konflikttyp stellt den etwas komplexeren Fall der Überlappung vor. Wenn sich zwei Klassen teilweise überlappen (*overlap*), können in dem vorgestellten Beispiel beide Klassen *hotel* und *hostel* einige *hostels* als billige Hotels angesehen werden, während einige Hotels keinesfalls *hostels* sind und einige *hostels* kaum als Hotel gelten. Hier werden zusätzliche Entscheidungskriterien benötigt, um herauszufinden, welche Teile der Instanzen die Konzepte teilen und welche nicht.

Umgekehrt ist es für den sinnvollen Informationsaustausch nicht nur wichtig zu wissen, wenn zwei Klassen Mitglieder gemeinsam haben, sondern auch, wenn dies gerade nicht der Fall ist, d. h. Klassen per definitionem disjunkt sind. Ein Beispiel für diesen dritten Konflikttyp (*inconsistency*) wären in vorgestelltem Beispiel etwa die Klassen *hotel* und *camp site*.

Ein weiterer möglicher Konflikt auf Ebene der Domäne (*domain-level*) ist bedingt durch die unterschiedlichen Abstraktionsebenen, die dazu führen können, dass Daten in verbundener Form auftreten (*aggregation*). In dem Beispiel kann etwa eine Quelle Städte entsprechend des Landes, andere entsprechend des Kontinents gruppieren. Häufig ähnelt diese Situation dem Subsumptions-Fall, da z. B. die Klasse „holländische Stadt“ vom Konzept „europäische Stadt“ subsumiert wird.

Im folgenden Abschnitt werden verschiedene Lösungsansätze vorgestellt, die die hier vorgestellten Konflikte über die Integration auf semantischer Ebene zu beheben versuchen.

4 Lösungsansätze für die Semantische Integration

Konkrete Integrationslösungen müssen Konzepte beinhalten, die Konflikte auf allen Ebenen, auf denen sie auftreten können, überwinden [BCG02].

Der Integration auf semantischer Ebene kommt dabei eine besondere Bedeutung zu, da zum einen auf struktureller und syntaktischer Ebene Integrationsansätze und -lösungen Verwendung finden, zum anderen im Web vorrangig unstrukturierte Informationen vorliegen, die einen semantischen Ansatz erfordern.

Die semantische Integration unterstützt das Informationsmanagement, indem sie Daten aus heterogenen, verteilten Quellen beschreibbar und deren Abhängigkeiten untereinander visualisierbar macht. Damit erfolgt eine inhaltliche Integration. Daten aus unterschiedlichen Quellen werden vergleichbar und können ggf. in neue Schemata integriert werden, dadurch wird die Auswahl von Quellen hinsichtlich bestimmter Anforderungen ermöglicht.

Informationsintegration kann dabei als Verbindung von Daten- und Funktionsintegration angesehen werden. Datenintegration zielt auf die Zusammenführung heterogener Datenbestände ab. Funktionsintegration bezeichnet das Ver-

fügbarmachen lokaler Funktionen bzw. Dienste aus den einzelnen Systemen in einer einheitlichen Form [SAL03]. Das Ziel von Datenintegration ist es, die Daten aus den verteilten, heterogenen Datenquellen zu einer einheitlichen Beschreibung, dem globalen Schema, zusammenzuführen. Dabei handelt es sich um eine vereinheitlichte Sicht der zugrunde liegenden Datenquellen, welche den Benutzenden die Möglichkeit bietet, über eine einheitliche Anfrageschnittstelle Anfragen zu stellen, ohne die Heterogenität der Datenquellen und deren Ort berücksichtigen zu müssen [BCG02].

Funktionsintegration

Um Heterogenitätskonflikte aufzulösen, müssen zunächst Übereinstimmungen (sog. *matches*) zwischen den Schemata der verschiedenen Datenquellen aufgezeigt werden. Dies können 1:1 oder 1:n (bzw. n:n) *matches* sein (vgl. *Schema matching* bzw. *mapping*). Diese Abbildungsart wird derzeit noch häufig manuell mit Unterstützung verschiedener Anwendungen durchgeführt. Dieser Prozess ist sehr arbeitsintensiv und fehleranfällig. Im Allgemeinen werden 60-80 % der Ressourcen eines Data-Sharing-Projekts für die Auflösung der semantischen Heterogenität verwendet [DNH04].

Zur Verwendung kommen Datenintegrationssysteme, welche im Allgemeinen auf einer wrapper/mediator-Architektur beruhen. Basis ist ein einheitliches und semantisch ausdrucksstarkes Datenmodell zur Repräsentation der Datenquellen. Je nach konkreter Vorgehensweise spricht man vom Materialisierungsansatz, wenn Daten periodisch aus den Quellen extrahiert und in einer zentralen Datenbank abgelegt werden (*data warehouse*). So können Anfragen über diese zentrale Datenbank beantwortet werden. Werden Daten nicht in einer zentralen Datenbank abgelegt, spricht man vom virtuellen Ansatz [RAV03], der im folgenden Abschnitt näher erläutert wird.

Die Herangehensweise zur Erfassung der Semantik der Daten unterscheidet sich in den verschiedenen Ansätzen. Der Zugang zur Semantik kann über die Struktur, über die Metadaten der Datenquellen oder aber über die natürlich-sprachliche Verarbeitung des Ursprungstextes erfolgen. Letzteres findet man eher bei unstrukturierten Datenquellen, wo entsprechende Integrationssysteme unstrukturierte Anfragen unterstützen müssen (Volltextsuche). Ersteres findet man bei semistrukturierten bzw. strukturierten Quellen, bei denen die Datenintegration strukturierte Anfragen unterstützt wie in Datenbank-Systemen.

Wrapper sind Softwarekomponenten, die den Inhalt einer Datenquelle zur Vereinheitlichung in einem anderen Datenmodell oder Schema repräsentieren. Ein Beispiel dafür wäre ein XML-Wrapper für eine relationale Datenbank.

Wrapper

Mediatoren sind Softwarekomponenten, die der Vereinfachung, Reduzierung, Kombination und Erklärung von Daten dienen. Sie werden v. a. zur Bereitstellung einer gemeinsamen Anfragemöglichkeit auf unterschiedliche Datenquellen genutzt. Abbildung 4.1 stellt diesen Sachverhalt grafisch dar.

Mediatoren

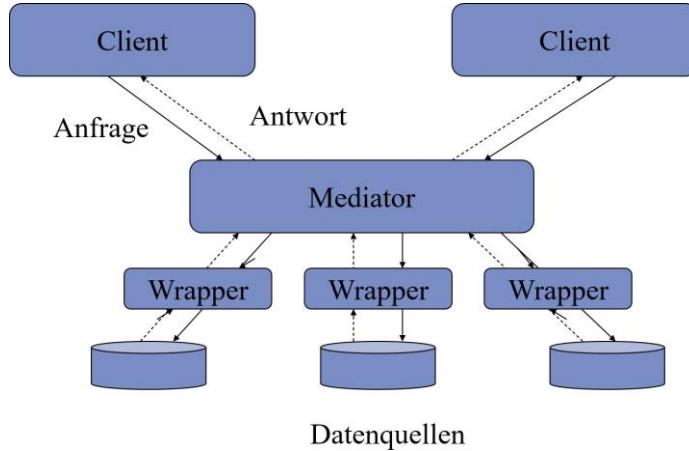


Abbildung 4.1 Wrapper und Mediatoren

Aufgabe des Mediators ist es, Anfragen an das globale Schema in Anfragen an die Quellen zu transformieren sowie die Ergebnisse zu sammeln und zu verknüpfen. Das globale Schema basiert auf einem geeigneten Datenmodell, welches z. B. als XML/RDF Repräsentation vorliegt [DMH00]. Da es sich um eine virtuelle Integration handelt, werden keine Nutzdaten separat gespeichert, sodass Anfragen an den Mediator unmittelbar in solche an die Quellsysteme umgewandelt werden (virtueller Ansatz). Dies geschieht durch Umwandlung der Anfrage in eine dem globalen Schema entsprechende interne Repräsentation und Umschreibung bzw. Zerlegung in durch die Quellsysteme ausführbare Teilanfragen. Die Ergebnisse müssen dann umgekehrt entsprechend der Definition des globalen Schemas und der Anfrage kombiniert und ggf. selektiert oder anderen Operationen unterworfen werden, um anschließend dem Nutzenden zur Verfügung gestellt zu werden [PSP01].

Die Kopplung zwischen Quelle und Mediator erfolgt über Wrappers, die dem Mediator einen einheitlichen Zugriff auf die Quellen ermöglichen, indem sie eine Abbildung zwischen dem Datenmodell des Mediators und dem der Quellen erstellen und zudem die Anfragen des Mediators in Anfragen oder Funktionsaufrufe der Quellsysteme übersetzen [BEM01].

Der Zugriff auf Ressourcen strukturierter Information von Wrappern, die aus dem konzeptionellen Modell abgeleitet werden können, kann gewährleistet werden, da in relationalen oder XML-basierten Systemen Standardschnittstellen verfügbar sind. Dagegen stellen wenig strukturierte Informationsquellen, wie zum Beispiel HTML-Seiten im Web, ein größeres Problem dar. Die relevanten Daten müssen zunächst identifiziert und extrahiert werden [RAV03]. Hierzu können entweder formale syntaktische Methoden oder Methoden automatischen Lernens verwendet werden, wobei im Fall des automatischen Lernens repräsentative Beispieleseiten mit den relevanten Dateneinträgen dem entsprechenden Lernverfahren vorgelegt werden. Als Ergebnis des automatischen Lernprozesses gewinnt man eine Menge von Auswahl-Regeln, die zur Extraktion von Informationen aus Web-Ressourcen verwendet wer-

den kann. Diese werden in eine neu geschaffene Struktur eingefügt, die für den weiteren Prozess als Grundlage dient [SvH05].

Damit liegt zwar eine Lösung für das Auslesen von Information aus schwach strukturierten Ressourcen vor, aber das Problem der Integration von Informationen aus verschiedenen Quellen bleibt weitgehend ungelöst, da Extraktionsregeln nur auf struktureller Ebene definiert sind. Man benötigt also eine Verknüpfung zwischen der semantischen Ebene der extrahierten Information und der strukturellen Ebene des jeweiligen Datenmodells.

Erfassen von Semantik über die Struktur

Ein weit verbreiteter Weg, um die Bedeutung von Information zu erfassen, ist die Beschreibung ihrer Struktur. Die Verwendung von konzeptionellen Modellen der gespeicherten Information ist von Datenbanksystemen bekannt (*Entity-Relationship-Modell*). Hier werden Objekte und Beziehungen der abzubildenden Domäne speziellen Tabellen, genauer gesagt deren Spalten oder Zeilen, zugeordnet (siehe Abbildung 4.2). Solche konzeptionellen Modelle weisen eine enge Verbindung zu der Art der Speicherung von Informationen auf, da sie hauptsächlich für die Strukturierung der abzubildenden Domäne ohne Berücksichtigung von situationsbezogenen und semantischen Unschärfen bei der Modellierung von Realweltausschnitten genutzt werden. Dadurch erfolgt eine Reduktion auf präzise, vollständig definierte Daten und Attribute, was einerseits zu Informationsverlust bzw. -veränderung führen kann. Andererseits hat die Verknüpfung von Struktur und semantischem Modell jedoch bedeutende Vorteile in Bezug auf die gemeinsame Nutzung von Informationen, da das konzeptionelle Modell bei dem Zugriff auf Information und bei dessen Validierung unterstützt [SvH05].

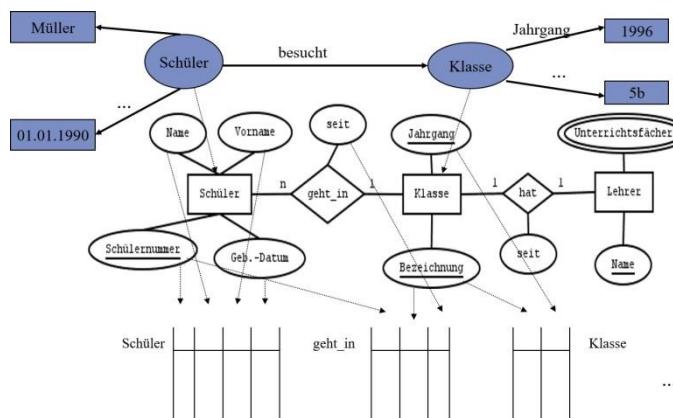


Abbildung 4.2 Erfassen der Bedeutung über die Struktur¹

¹ Beruht auf ERM (online) www.tinohempel.de/info/info/datenbank/erm.htm; letzter Zugriff 2007/02/02)

Hierzu wird es nötig sein, die unterschiedlichen Konzepte miteinander zu homogenisieren (*matchen*). Dies ist semantisch korrekt nur dann möglich, wenn allen zu integrierenden Datenquellen dasselbe semantische Modell zugrunde liegt, sodass alle Elemente und Beziehungen des einen Konzepts ihre Entsprechungen im anderen finden. Abbildung 4.3 soll diesen Sachverhalt veranschaulichen.

Es gibt unterschiedliche Ansätze zur Verknüpfung von Informationsquellen und Datenmodellen [WVV02], die im Folgenden dargestellt werden.

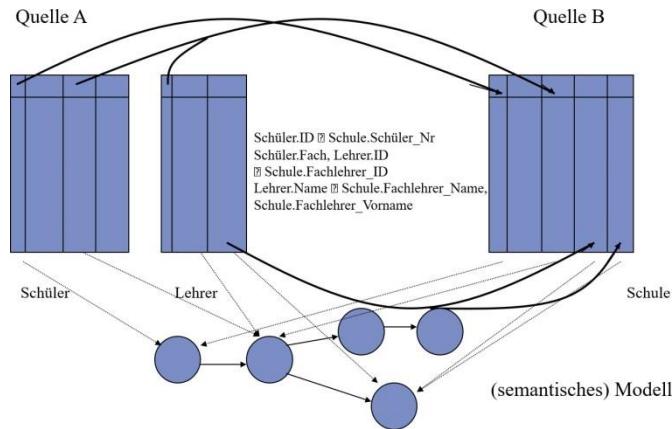


Abbildung 4.3 Matching über gemeinsames semantisches Modell

Struktur-Ähnlichkeit (structure resemblance)

Es wird ein logisches Modell erstellt und in einer Sprache codiert, die automatisches, logisches Schließen ermöglicht. Das Modell stellt eine 1:1 Kopie der konzeptionellen Struktur der Datenbank dar. Die Integration erfolgt auf der Kopie dieses Modells und kann anschließend leicht auf die Originaldaten zurückgeführt werden. Diese Methode ist etwa im SIMS-Mediator, dem MOMIS System [BCV99] und im TSIMMIS System [OUS99] implementiert. Eine geeignete Kodierung der Informationsstruktur kann bereits genutzt werden, um Hypothesen über die semantisch zusammenhängenden Strukturen in zwei Informationsquellen zu erstellen [SvH05].

Term-Definition (Definition of terms)

Um die Semantik von Ausdrücken in einem Datenbankschema greifbar zu machen, reicht eine Kopie desselben nicht aus. BUSTER [VIS02] stellt beispielsweise einen Ansatz dar, der ein Datenmodell zur Definition von Termen aus der Datenbank oder dem Datenbankschema verwendet, die nicht unmittelbar mit deren Struktur korrespondieren, sondern lediglich mit den entsprechenden Informationen verlinkt sind. Die Definition selbst kann sogar aus einer Reihe von Regeln bestehen, die den Term definieren. In den meisten Fällen wird er jedoch von Konzeptdefinitionen beschrieben [BEM01].

Struktur-Anreicherung (structure enrichment)

Hier wird ein logisches Modell konstruiert, welches die Struktur der Informationsquelle abbildet und zusätzlich Definitionen von Konzepten enthält. Es stellt momentan den verbreitetsten Ansatz dar [STT02] und verbindet die beiden oben genannten Ansätze. Zu den Systemen, die diese Methode nutzen, gehören etwa OBSERVER, KRAFT, PICSEL, DWQ und InfoQuilt ([BEF01], [PAS01]). OBSERVER verwendet Ontologien, um Informationsquellen zu beschreiben und um Beziehungen (Synonyme, Hypo-, Hypernyme) zwischen Termen unterschiedlicher Modelle zu ermöglichen. Dieser Ansatz ist beschränkt auf einfache Beziehungen [SvH05]. PICSEL und DWQ definieren die Struktur der Informationen mit typisierten Horn-Regeln. Die darin enthaltenen zusätzlichen Konzept-Definitionen werden von einem Beschreibungs-Logik-Modell (*description-logic model*) erzeugt. KRAFT legt kein spezielles Definitionsschema fest [VIS02]. InfoQuilt ermöglicht die Darstellung von komplexen Beziehungen zwischen verschiedenen Domänen, unterstützt verschiedene Funktionen und Simulationen und stellt so ein mächtiges Abfrage-Interface zur Verfügung [PAS01].

Meta-Annotation

Dies ist ein relativ junger Ansatz, der den Gegebenheiten des Webs Rechnung trägt, wo semantische Informationen häufig in Form von Kommentaren bzw. Anmerkungen hinzugefügt werden. Diese Konstrukte werden genutzt, um Zugang zur Semantik zu gewinnen. Ontobroker und SHOE sind Beispiele für diesen Ansatz [WVW02]. Da bei schwach strukturierten Informationsquellen meist kein konzeptuelles Modell vorhanden ist, ist der Ansatz, sich der Semantik über die Struktur der Datenquellen zu nähern, für das Web oft ungeeignet (vgl. [BEF01]). Das Web stellt ja gerade ein sehr umfangreiches Reservoir an nicht, oder wenig strukturierten Inhalten dar. Wie schon erwähnt, bietet sich hier jedoch der folgende Zugang an.

Zugang über natürlich-sprachliche Verarbeitung des Ursprungstextes

Hierbei finden Methoden des Information Retrieval Anwendung. Unter dem Begriff des Information Retrieval ist gewöhnlich die vage Suche auf Dokumentinhalte und deren unspezifische Bewertung zu verstehen (vgl. [RAV03]). Die einzelnen Phasen des Information Retrieval sind dabei Deskribierung der Texte, Recherche und Bewertung der Ergebnisse. Die Deskribierung beschreibt die Transformation eines Textdokuments in eine Dokumentbeschreibung aufgrund von Metadaten und Schlagworten über Stichworte aus dem Text, den sogenannten indexierten Termen. Die Analyse des Textes anhand der vorkommenden Worte wird auch Indexierung genannt. Statistische, wortbasierte Verfahren sind dabei am gebräuchlichsten [SvH05].

Information Retrieval

Deskribierung

Die Aufgabe, relevante Informationen zu einem bestimmten Thema durch inhaltliche Suche auf Dokumenten zu finden, wird somit gelöst durch die Indexierung

freier Text-Dokumente mit gewichteten Ausdrücken, die mit ihrem Inhalt verbunden sind. Die etwa von Suchmaschinen im Web eingesetzten Systeme zur Informationsgewinnung erfassen Eigenschaften und Charakteristika von Dokumenten, ohne diese zu verwalten.

Da der semantische Inhalt eines Dokuments in den indexierten Termen beinhaltet ist, stellt deren Wahl und Erstellung bei der Behandlung der Semantik des Textes den entscheidenden Schritt dar. Dokumente werden durch den Abgleich der Terme von deren Dokumentbeschreibungen mit einem semantischen Modell, das kontextuelle Zusammenhänge und Beziehungen der Terme berücksichtigt, vergleichbar. Damit steht ein Verfahren, analog zum *Matchen* von Strukturelementen aus dem vorherigen Abschnitt zur Verfügung. Die Aufgabe, die beabsichtigte Bedeutung der Terme im jeweiligen Zusammenhang zu ermitteln, wird als Disambiguierung (*disambiguation*) bezeichnet.

Bei der Analyse eines Dokuments und der Generierung von Dokumentbeschreibungen können zunächst Methoden wie die Auswahl der Terme anhand lediglich einfacher linguistischer Analysen oder die Berücksichtigung von Beugungsformen genannt werden. Eine Bewertung anhand der Häufigkeit eines Worts führt jedoch oft zu unbefriedigenden Ergebnissen. Beispielsweise kann ein Wort sowohl als Verb als auch als Adjektiv (*fabricated units vs they fabricated units*) eingesetzt werden, wodurch die gefundenen Terme in ihrer Relevanz bzgl. der gestellten Benutzeranfragen stark differieren können. Daher analysieren komplexere linguistische Verfahren Worttypen und Satzstrukturen derart, dass Worte in ihrem Zusammenhang erkannt werden. Hierzu werden bestimmte linguistische Eigenschaften der Wörter wie Subjekt, Präposition etc. berücksichtigt. Eine zusätzliche Verbesserung der Erfassung semantischer Zusammenhänge kann durch Betrachtung des jeweiligen Kontextes, in dem ein Term vorkommt, erreicht werden. Dies erfolgt unter Berücksichtigung und Entscheidung zwischen verschiedenen möglichen Interpretationen, basierend auf dem Vorkommen anderer Wörter in diesem Zusammenhang. Daraus kann sich zugeleich ein Hinweis auf eine bestimmte Bedeutung ergeben. Die Ausnutzung dieser

impliziten Strukturen wird als verborgenes semantisches Indexieren (*latent semantic analysis*) bezeichnet (vgl. [BEF01]).

Die Entscheidung für eine bestimmte Bedeutung basiert häufig auf einem natürlichsprachlichen Wortschatz, etwa in Form eines Lexikons oder eines Thesaurus. Zudem kann ein Thesaurus zu Termen noch verschiedene Beziehungen wie Vorentscheidungen, Querverweise, Ober- und Unterbegriffe u.s.w. speichern, um Fälle, in denen spezialisierte Vokabulare in Dokumenten verwendet werden, zu berücksichtigen (siehe Abbildung 4.4).

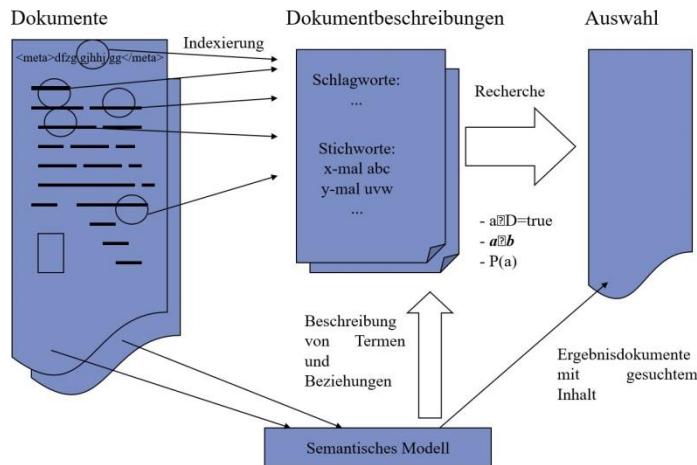


Abbildung 4.4 Erfassen der Bedeutung über natürlichsprachliche Verarbeitung

Diese Beziehungen werden zudem von domänenspezifischen Lexika oder semantischen Netzen zur Verfügung gestellt. Ausdrucksstärkere Terme zur Gewinnung von komplexer indexerter Information erhält man durch Verwendung von Ontologien, die das Anwendungsgebiet geeignet strukturieren und so komplexere Zusammenhänge darstellen können.

Nachdem die Dokumente durch eine Deskribierung entsprechend aufbereitet sind, können Anfragen an die Dokumente gestellt werden. Bei der Recherche werden die gewonnenen Terme (Stichworte, Schlagworte) verwendet und mit verschiedenen Operatoren, Kontextinformationen oder Wahrscheinlichkeiten versehen, um die Menge der Ergebnisdokumente effektiv einzuschränken.

Eine Auswertung der Recherchen kann nach verschiedenen Retrieval-Modellen erfolgen. Im einfachsten Fall, dem Boole'schen Retrieval, gibt der Wert true an, dass die recherchierten Terme im Dokument vorkommen. Mehrere gefundene Dokumente haben ggf. dieselbe Relevanz.

Boole'sches Retrieval

Im Vektorraum-Modell werden die Such- und Dokument-Terme jeweils als Vektoren im mehrdimensionalen Suchraum aufgefasst. Mit vorgegebenen Ähnlichkeitsmaßen können auch Dokumente mit „benachbarten“ Vektoren gefunden werden, falls die Such-Terme nicht direkt getroffen werden. Es wird also ein mehr oder weniger vages Ergebnis geliefert, welches durch Wahrscheinlichkeiten der Terme verfeinert werden kann.

Vektorraum-Modell

In probabilistischen Modellen werden Wahrscheinlichkeiten hinsichtlich der Relevanz eines Dokuments in Abhängigkeit zur Anfrage berechnet. So sollen beim Ranking die gefundenen Dokumente entsprechend ihrer Relevanz sortiert werden; ggf. kann der Nutzer beim *Relevance Feedback* bestimmte Dokumente entweder als irrelevant oder als relevant markieren und so die Rankingsortierung hin zu relevanteren Dokumenten verschieben.

Relevance Feedback

Eine Besonderheit des Information Retrieval im Web stellt die Tatsache dar, dass nicht von bestehenden Dokumentmengen ausgegangen werden kann, sondern diese im Web eingesammelt werden müssen (*crawling*). Hierzu werden die Link-Referenzen zwischen den Dokumenten ausgenutzt. Die ermittelte Link-Struktur kann zu Ranking-Zwecken verwendet werden. Generell stellen sich an Retrieval-Techniken im Web spezielle Anforderungen, zu denen nicht nur eine möglichst vollständige Abdeckung der verfügbaren Information, sondern auch die Aktualität der Indizes zählen. Ersteres ist angesichts des Umfangs des Web-Inhalts kaum realisierbar. Tatsächlich werden nur etwa 60 % der direkt erreichbaren Dokumente und nur ein Bruchteil der in Web-Datenbanken vorgehaltenen, und ggf. über dynamisch generierte Webseiten ausgegebenen Information gefunden. Die Aktualität der Indizes liegt bei etwa 50-150 Tagen, d. h. jüngere oder in dieser Zeit geänderte Webinhalte werden u. U. nicht gefunden [RAV03]. Die Methoden des Information Retrieval finden ihre Grenzen zum einen bei spezialisierten Texten, insbesondere der wissenschaftlichen Art, da die dort verwendete Terminologie gar nicht oder kaum mit bestehenden linguistischen Verfahren erfassbar ist und zum anderen bei komplexerer Indexierung. Hier fehlt ein eindeutiges semantisches Modell, welches Terme und ihre Beziehungen beschreibt.

Selbsttestaufgabe 6.4:

Wie würden Sie „Disambiguierung“ definieren?

Semantische Modelle

Klassifizierung

Spezielle Vokabulare treten, wie oben gezeigt, oft in Klassifikations- und Bewertungsausdrücken auf. Sie dienen der Reduzierung des in der Informationsquelle gespeicherten Datenaufkommens. Statt alle Merkmale eines durch einen Datensatz repräsentierten Objekts zu beschreiben, wird eine Klasse verwendet, die Objekte mit denselben Eigenschaften beinhaltet. Dieser Ausdruck entspricht oft einer Klassifizierung, welche außerhalb der Informationsquelle spezifiziert wird, z. B. die Verwendung von Produkt-Kategorien bei e-Commerce oder der Bezug auf standardisierte Flächen- nutzungsarten in geographischen Informationssystemen [TUM06]. Wie bereits erwähnt, können so wichtige Information verloren gehen.

Information sharing

Neben der großen Bedeutung für die gemeinsame Informationsnutzung (*information sharing*) im Web, ist der Einsatz von heterogenen Klassifikationsschemata auch für die Integration von Information bedeutsam. Information Sharing erfordert sowohl das Auffinden geeigneter Informationsquellen als auch den Zugang zu den darin enthaltenen Daten, d. h. die gefundenen Informationsquellen müssen mit dem anfragenden System zusammenarbeiten können. Man spricht hier von der informationellen Interoperabilität [WVV02].

Hierzu sind auf niedrigeren Ebenen formale Sprachen wie XML oder RDF nötig, die sowohl die Struktur der Information als auch Meta-Informationen über die Na-

tur der Information und der konzeptionellen Struktur, die der Informationsquelle zugrunde liegen, erfassen und beschreiben können. In einem weiteren Schritt müssen die Informationsquellen mithilfe der erwähnten Sprachen mit Information zur Semantik versehen werden. Diese semantische Information muss auf einem Vokabular basieren, welches eine übereinstimmende und formale Spezifikation der Konzeptualisierung der Domäne zum Ausdruck bringt [SvH05]. Bei der Integration von Information können diese geeigneten semantischen Modelle verwendet werden, um die Semantik der Informationsquelle zu beschreiben und offen zu legen.

In Hinblick auf die Integration von Datenquellen können semantische Modelle somit zur Identifizierung und Verbindung von sich semantisch entsprechenden Informationskonzepten verwendet werden (*semantic matching*). Diese Ebene umfasst insbesondere das Ausstatten der Informationsquellen mit zusätzlicher semantischer Information und der Verwendung von gemeinsam genutzten Term-Definitionen. Dies wird bereits in aktuellen Ansätzen gemeinsamer Informations-Nutzung in Form von Meta-Notationen und Term-Definitionen implementiert [SvH05]. Neben der Beschreibung und Offenlegung der Semantik von Informationsquellen können semantische Modelle auch als globales Anfragemodell und Grundlage für die Verifikation von Integrationsschritten dienen [WVV02].

5 Semantische Modelle darstellen und vergleichen

Wesentliche Voraussetzung für die Integration von Information ist die Vergleichbarkeit von Information auf semantischer Ebene. Dem liegt die Vergleichbarkeit der Bedeutungen von Ausdrücken zugrunde, die als Elementnamen eines Schemas und als Attributwert verwendet werden. Ein einfaches Beispiel für die Problematik, Ausdrücke in Relation zu einer bestimmten Bedeutung zu setzen, zeigt sich bei der Verwendung von Wörterbüchern. Je nach Kontext kann ein Wort unterschiedliche Bedeutungen haben (sog. Homonyme). Andererseits können unterschiedliche Worte in einem bestimmten Kontext Bedeutungsgleichheit bzw. Bedeutungsähnlichkeit besitzen (sog. Synonyme). Durchsucht man Dokumente nach bestimmten Ausdrücken ohne Berücksichtigung ihrer Bedeutung, kann man demzufolge sowohl irrelevante, also zu viele Ergebnisse als auch zu wenige Ergebnisse erhalten. Basis für Integration und Transformation von Datenwerten stellen die semantischen Beziehungen zwischen diesen Ausdrücken dar [SvH05].

Wissensrepräsentation

Die Integration von Information wirft somit die Frage nach der Natur der verwendeten semantischen Modelle auf. In der Regel liegen einzelnen Informationsdarstellungen unterschiedliche Modelle zugrunde. Maßgeblich ist hierbei deren formale Repräsentation, d. h. die Art der verwendeten Sprache. Kern derselben ist ein im letzten Abschnitt vorgestelltes semantisches Modell der Wissensrepräsentation [WVV02].

Es gibt unterschiedliche Möglichkeiten der Konzeptualisierung und der Kontext-Wissens-Darstellung. Diese reichen mit wachsender semantischer Ausdrucksstärke von der einfachen informellen Beschreibung von Ausdrücken in natürlicher Sprache (Glossar) über einfache Hierarchien von Ausdrücken oder komplexe Netzwerke über Hierarchien von Konzepten, komplexen Konzeptbeschreibungen (Ontologien) bis hin zu streng formalen Ansätzen mit der Ausdrucksstärke von Prädikatenlogik [VIS02].

Die einzelnen Ansätze sollen im Folgenden etwas näher beschrieben werden (siehe Abbildung 5.1). Hierzu wird als Beispiel der Begriff trip betrachtet, der nach dem Wörterbuch nachfolgende Bedeutungen besitzen kann:

1. a journey for some purpose (he took a trip to the shopping center)
2. a hallucinatory experience induced by drugs (an acid trip)
3. an accidental misstep threatening or causing a fall
4. tripper, trip: a catch mechanism/switch (the pressure activates the tripper and releases the water)
5. a light or nimble tread (he heard the trip of women's feet overhead)
6. an unintentional but embarrassing blunder (recite a poem without a single trip)

Abbildung 5.1 Beispiel 5 [SvH05]

Namen und Bezeichner

semantischer Kontext

Die im zweiten Abschnitt angesprochene Möglichkeit der Definition von Terminologien mittels DTD bzw. XML-S in XML-Dokumenten stellt letztendlich nichts anderes als eine Hierarchie von Termen dar. Allgemeiner formuliert repräsentiert die Organisation von Termen in Netzwerken durch zweiwertige Relationen die einfachste Art, einen semantischen Kontext auszudrücken.

Vor allem auf dem Gebiet des Information Retrieval ist eine Reihe von Methoden zur Kontextklärung entwickelt worden, die weitere Ausdrücke verwendet, um mehr Information über die beabsichtigte Bedeutung eines Ausdrucks zu liefern. Ein Ansatz dazu kann in der Verwendung von Mengen von Synonymen anstelle einzelner Ausdrücke gesehen werden. Ein Synonyme-Set enthält alle Ausdrücke, denen eine Bedeutungsgleichheit bzw. -ähnlichkeit zu eigen sind. In dem oben angeführten Beispiel wären demnach trip und journey in einem Synonyme-Set mit der ersten Bedeutung, ein Synonym-Set mit der zweiten Bedeutung würde z. B. hallucination enthalten.

Mithilfe von Ähnlichkeitsmaßen, welche allen Mitgliedern der Synonyme-Sets zweier zu vergleichender Ausdrücke zugewiesen werden, können Ausdrücke mit gleicher bzw. ähnlicher Bedeutung gefunden werden, da ihre Synonyme-Sets einige Ausdrücke gemeinsam haben. Sie verhindern außerdem falsche Übereinstimmungen zwischen Ausdrücken unterschiedlicher Bedeutung, da deren Synonyme-Sets annähernd disjunkt wären.

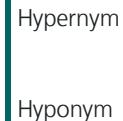
Selbsttestaufgabe 6.5:

Erklären Sie den Begriff der „Ontologie“ wie er bislang hier definiert und vorgestellt wurde.

Termnetzwerke

Nutzt man weitere Relationen wie etwa Hyper-, Hypo-, Holo-, Mereonyme zu anderen Termen kann man Netzwerke aufbauen, die den Kontext eines Terms weit ausführlicher beschreiben.

Den Oberbegriff eines Begriffes nennt man Hyponym. Unter Hyponym versteht man den Unterbegriff eines Begriffs. Dabei ist der Begriffsumfang des Hyponyms kleiner als der des Hyperonyms, aber der Begriffsinhalt des Hyponyms ist größer als der des Hyperonyms.



Als Holonym wird ein Begriff bezeichnet, der das Ganze einer „Teil-von-Beziehung“ zwischen zwei zusammengehörigen Begriffen darstellt. Die Umkehrung dieser Relation bezeichnet man als Meronymie. So stellt der Begriff Hand ein Holonym von dem Begriff Finger dar, und der Begriff Finger ist ein Meronym von dem Begriff Hand.



Zusammen mit den Termen, die sie verbinden, bilden diese Relationen umfassende Netzwerke zwischen Termen und deren Beziehungen. Die verbreitetste Form solcher Netzwerke sind Thesauri (Synonymwörterbücher), die hauptsächlich Hyper- und Hyponym-Relationen verwenden, um Hierarchien von Ausdrücken zu bilden. Auch hier können Ähnlichkeitsmaße helfen, semantische Zusammenhänge aufzudecken. Zur Ermittlung der Ähnlichkeitsmaße kann man unter anderem die Pfadlänge zwischen zwei Ausdrücken sowie statistische Informationen nutzen.



Konzepthierarchien

Das Problem von Term-Netzwerken ist, dass sie kein formales Prinzip zum Hierarchieaufbau anbieten. Daher können sich immer noch verschiedene mögliche Interpretationen eines Ausdrucks denselben Platz in einer Hierarchie teilen. In Bezug auf das Beispiel in Abbildung 5.1 stehen trek und tumble (Sturz wg. trip u. a. Stolpern) auf gleicher semantischer Ebene im Hinblick auf trip. Nutzt man anstelle einer Hierarchie von Ausdrücken eine Hierarchie von Konzepten zur Beschreibung der Bedeutung der Ausdrücke, kann jedes Konzept durch die Eigenschaften seines Vorgängers in der Hierarchie, die es erbt, definiert werden. Die geerbten Definitionen können beim Vergleich der Bedeutung zweier Konzepte benutzt werden, um mehr und genauere Informationen zu bekommen.

Ontologien

Fügt man den Instanzen der Konzepte bestimmte Kennzeichen und Einschränkungen hinzu (*features, constraints*), kann man deren Bedeutung noch näher und

aussagekräftiger bestimmen. Im oben angeführten Beispiel könnte definiert werden, dass jeder trip Attribute wie z. B. eine Richtung (*destination*) und eine Dauer (*duration*) besitzt, dass er aus verschiedenen Teilen bestehen kann, entweder im Sinne von Abschnitt/Etappen (*stages*) oder im Sinne von Etappe/Runde (*legs*), und dass er verschiedene Funktionen erfüllen kann, wie etwa einen Besuch (*visit*). Ein formaler und verbreiteter Ansatz nutzt RDF-S, um Konzepte entsprechend zu beschreiben. Ein RDF-Schema enthält Definitionen von Klassen mit entsprechend zugehörigen Eigenschaften, die durch *constraint-properties* enger gefasst werden. Default-Werte und Einschränkungen des Wertebereichs sind jedoch immer noch von zu geringer Ausdrucksstärke.

Die Definition von Klassen mittels logischer Formeln, Regelmengen oder komplexer Axiomsysteme in Logik erster Ordnung bietet hier weit mächtigere Repräsentationsmöglichkeiten. So wurden spezielle Repräsentationsformalismen entwickelt, die epistemologische (erkenntnistheoretische) Primitive anbieten, mit denen Konzepte hinsichtlich der charakteristischen Merkmale ihrer Instanzen definiert werden können. Zu den am häufigsten verwendeten gehören framebasierte Repräsentationen und Beschreibungslogiken (*description logics*). Framebasierte Systeme definieren einen Rahmen mit fester Struktur zur Beschreibung der Eigenschaften der Instanzen von bestimmten Konzepten, während *description logics* eine Untermenge der Prädikatenlogik darstellen. Sie bieten eine flexible logische Sprache zur Definition notwendiger und hinreichender Bedingungen an, die bestimmte Instanzen erfüllen müssen, um zu einem Konzept zu gehören ([BEF01], [PAS01]).

Alle erwähnten Ansätze zur Beschreibung von Semantiken, die auf charakteristischen Merkmalen von Instanzen beruhen, können benutzt werden, um die intendierte Bedeutung von Information zu vergleichen. Auf dem Gebiet des fallbasierten Schließens (*reasoning*) sind Ähnlichkeitswerte definiert worden. Diese ermöglichen den Vergleich von Konzepten, die als auf Attribut-Wert-Paaren basierende „Fälle“ repräsentiert werden. Für framebasierte Sprachen sind Vergleichsalgorithmen vorgeschlagen worden, die durch die Berücksichtigung der Struktur der Konzeptausdrücke semantische Übereinstimmungen ermitteln können. Im Fall der 1. Ordnung-Axiomsysteme (Prädikatenlogik 1. Ordnung) kann logisches Schließen genutzt werden, um festzustellen, ob eine Repräsentation eine andere impliziert oder ob zwei Repräsentationen äquivalent sind und daher dieselbe Bedeutung haben [SvH05].

Da der Vergleich von Semantiken, die auf dem Prinzip allgemeiner Deduktion beruhen, sich oft als schwierig erweist, bietet die Beschreibungslogik spezialisierte Schließverfahren, um zu ermitteln, ob die Definition eines Konzepts ein Spezialfall eines anderen ist (oder von diesem subsumiert wird). Diese Möglichkeit macht Beschreibungslogiken zu einem mächtigen Werkzeug zum Beschreiben und Vergleichen von Semantiken und zum verbreitetsten Ansatz für die Darstellung von Ontologien ([BEF01], [ANB04]).

6 Zusammenfassung

Die Erschließung und Nutzung der umfangreichen Informationsquellen im Web ist nur mit maschineller Unterstützung möglich. Hierzu ist es erforderlich, die vorhandenen Informationen für maschinellen Zugang und Verarbeitbarkeit zu öffnen. Wesentliche Voraussetzung hierfür ist die Integration heterogener, verteilter Informationen und die Erschließung ihrer Semantik. Für die Erfassung der Semantik der vorliegenden Informationen ist eine geeignete Darstellungsform notwendig. Syntaktische Standards wie XML und RDF und die Verwendung von strukturierten Metadaten zur

„Anreicherung“ der entsprechenden Dokumente mit semantischem Inhalt ermöglichen in Verbindung mit semantischen Modellen und den entsprechenden (Ontologie-)Sprachen den Zugang entsprechender Software-Tools zur Semantik der Web-Inhalte.

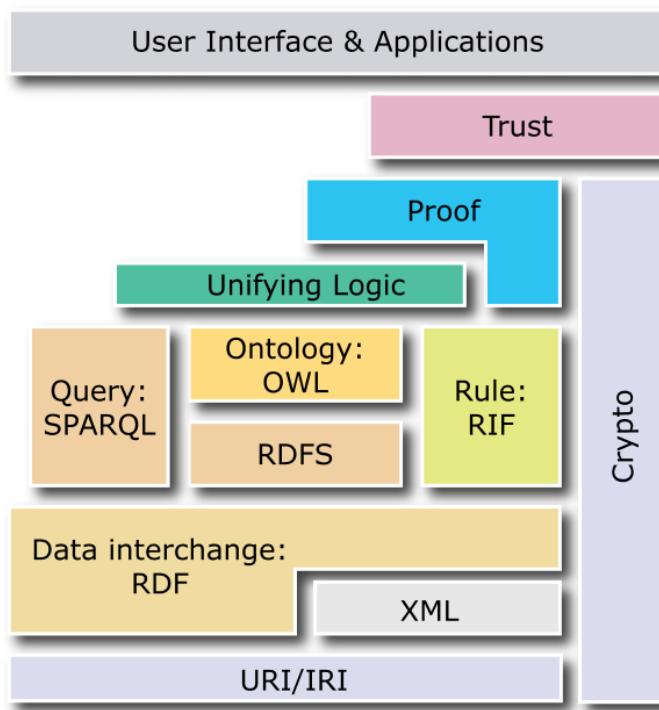


Abbildung 6.1 Semantisches Schichtenmodell [BRA07]

Der Austausch von Informationen zwischen verschiedenen Anwendungen erfolgt über XML- bzw. RDF-basierte Sprachen. So können etwa Fakten, Aussagen oder Anfragen mithilfe von RDF-Statements repräsentiert werden. Auf Basis eines gemeinsamen semantischen Modells kann eine übereinstimmende Interpretation der Bedeutung der entsprechenden Terme gewährleistet werden. RDF-S stellt bereits eine primitive Ontologiesprache dar. Dennoch sind ausdrucksstärkere Sprachen, Darstellungsmöglichkeiten anwendungsspezifischen deklarativen Wissens durch Regelmengen und formale Logik sowie Mechanismen zur Validierung und Beweisführung erforderlich. Um Sicherheit und Qualität der verarbeiteten Informationen

gewährleisten zu können, ist eine weitere Ebene, die die Vertrauenswürdigkeit der Operationen sicherstellt, notwendig.

Die Heterogenität der verteilten Informationsquellen im Web stellt die größte Herausforderung hinsichtlich deren Integration dar. Es müssen Konflikte auf syntaktischer, struktureller und semantischer Ebene aufgelöst werden. Der Integration auf semantischer Ebene kommt dabei eine besondere Bedeutung zu. Zum einen sind, basierend auf den vorgestellten Standards und Datenbankanwendungen, Integrationsansätze und -lösungen auf struktureller und syntaktischer Ebene bekannt und finden Verwendung, wobei aber keine inhaltliche Integration möglich ist. Zum anderen wird im Web vorrangig unstrukturierte Information vorgefunden, die einen semantischen Ansatz erforderlich macht. Dieser kann auf dem Zugang zur Semantik über die Struktur oder die natürlich-sprachliche Verarbeitung des Ursprungstextes basieren. Aufgrund der erwähnten Unstrukturiertheit vieler Web-Inhalte kommen vorrangig Methoden des Information Retrieval zum Einsatz, wobei komplexe linguistische Verfahren und Sammlungen von Wortschätzern sowie unterschiedlich komplexe semantische Modelle zur Darstellung von Kontext-Wissen und Konzeptualisierungen Verwendung finden.

Je nach Grad der Ausdrucksstärke lassen sich verschiedene Ansätze unterscheiden, um Semantiken zu beschreiben und zu vergleichen. Sie können einfache Hierarchien von Ausdrücken oder komplexe Netzwerke sowie Hierarchien von Konzepten bis hin zu komplexen Konzeptbeschreibungen (Ontologien) umfassen. Die am häufigsten verwendeten Repräsentationen komplexer Konzepte sind framebasierte Darstellungen und Beschreibungslogiken. Vergleichsalgorithmen und logische Schließverfahren ermöglichen es, unterschiedliche Repräsentationen hinsichtlich ihres semantischen Inhalts zu vergleichen und ggf. zusammenzufassen.

Obwohl zu den verschiedenen Konflikttypen innerhalb der semantischen Integration bereits Lösungsansätze erarbeitet und teils auch umgesetzt worden sind, kann man nicht davon sprechen, dass das Ziel eines semantischen Webs schon erreicht wäre. Einzelne isolierte Anwendungen vorrangig im universitären und wissenschaftlichen Umfeld demonstrieren zwar das Potential und verschiedene Werkzeuge und Quasi-Standards geben die Möglichkeit der Etablierung weiterer nützlicher Anwendungen. Dennoch sind trotz aller Fortschritte weiterhin sehr heterogene, spezialisierte und voneinander abgegrenzte Wissensbereiche einem semantischen Zugang geöffnet. Sowohl die fehlende Verwendung einheitlicher Ontologien als auch der Übergang von bestehenden, darstellungsorientierten zu inhaltlich orientierten Informationsquellen stellen mögliche Hindernisse dar.

Die Offenheit und Heterogenität des Webs, die die Notwendigkeit eines semantischen Ansatzes begründet haben, stellen somit auch die größte Herausforderung bei dessen Realisierung dar. So sind erste nennenswerte Erfolge am ehesten in In-

tranet-Umgebungen großer Organisationen, Universitäten oder Unternehmen zu erwarten [AvH03].

Einige Kritiker bezweifeln überhaupt die Möglichkeit der Etablierung eines semantischen Webs. Einerseits werden Zweifel hinsichtlich Annahmebereitschaft, Interessenlagen und Gewohnheiten der Nutzer ([PET06], [MCC06]) sowie der Nutzbarkeit der derzeit im Web vorzufindenden Informationen vorgebracht [MCC05]; andererseits herrschen Bedenken bezüglich des aktuellen Stands und Grenzen der Software-Entwicklung ([PEB06], [HEP06]).

Weitere Problemfelder eröffnen sich durch Datenmissbrauch, Überwachung und Zensur. Die Notwendigkeit, in Ontologien Zusammenhänge der Welt zu kategorisieren, wirft die Frage auf, wer diese mit welchem ideologischen und kulturellen Hintergrund erzeugt und wie dabei ein Konsens gefunden werden kann.

7 Literaturverzeichnis

- [RAV03] E. Rahm, G. Vossen (Hrsg.) Web & Datenbanken. dpunkt.verlag, 2003
- [AvH03] Antoniou, G. and Harmelen van, F.: A Semantic Web Primer. s.l. : The MIT Press, 2003.
- [SvH05] Stuckenschmidt, H. and Harmelen, van F.: Information Sharing on the Semantic Web. Springer-Verlag, Berlin Heidelberg New York, 2005.
- [BEF01] Bertino, E. and Ferrari, E.: XML and Data Integration. In: IEEE INTERNET COMPUTING: 75-76, 11/12 2001.
- [DOE05] Dörk: Ontologiebasierte Informationsintegration. Seminararbeit. Seminar: Texttechnologien und Semantic Web. Ernesto William De Luca .Otto-von-Guericke Universität Magdeburg, 2005.
- [BCG02] Boucelma, O., Castano, S., Goble, C., Josifovski, V., Lacroix, Z. and Ludächer, B.: Report on the EDBT. Panel on Scientific Data Integration. In: SIGMOD Record, 31(4): 107-112, December 2002.
- [DNH04] Doan, A., Noy, A. and Halevy, A.: Introduction to the Special Issue on Semantic Intgration. In: SIGMOD Record, 33(4): 11-13, December 2004.
- [DMH00] Decker, S. Melnik, S. Harmelen, F. Van, Fensel, D., Klein, M., Broekstra, J., Erdmann, M., Horrocks, I.: The Semantic Web: The Roles of XML and RDF. In: IEEE INTERNET COMPUTING: 63-74, 09/10 2000.
- [PSP01] Panti, M., Spalazzi, L., Penzerini, L.: Cooperation Strategies for Information Integration. In: Cooperative Information Systems Proceedings of the 9th International Conference, CoopIS 2001, Trento, Italy, September 2001, Springer-Verlag, Berlin Heidelberg New York, 2001.
- [BEM01] Berlin, J. and Motro, A.: Autoplex: Automated Discovery of Content for Virtual Databases. In: Cooperative Information Systems Proceedings of the 9th International Conference, CoopIS 2001, Trento, Italy, September 2001, Springer-Verlag, Berlin Heidelberg New York, 2001.
- [WVV02] Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-Based Integration of Information- A Survey of Existing Approaches. In: Ontology-based Information Integration Tutorial at the 13th International Conference on Knowledge Engeneering and Knowledge Management (EKAW). Sigüenza, Spain, 01.-04.10.2002 Universität Bremen, FB Mathematik und Informatik, September 2002.
- [BCV99] Bergamaschi, S., Castano, S., Vincini, V.: Semantic Integration of Semistructured and Structured Data Sources. In SIGMOD Record, 28(1): 54-59, March 1999.

- [OUS99] Ouksel, A. M., Sheth, A.: Semantic Interoperability in Global Information Systems A brief introduction to the research area and the special section. In: SIGMOD Record, 28(1): 5-12, March 1999.
- [VIS02] Visser, U. and Stuckenschmidt, H.: Interoperability in GIS-Enabling Technologies. In: Ontology-based Information Integration Tutorial at the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW). Sigüenza, Spain, 01.-04.10.2002 Universität Bremen, FB Mathematik und Informatik, September 2002.
- [STT02] Stuckenschmidt, H., Timm, I.: Information Integration on the Semantik Web. In: Ontology-based Information Integration Tutorial at the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW). Sigüenza, Spain, 01.-04.10.2002 Universität Bremen, FB Mathematik und Informatik, September 2002.
- [PAS01] Patel, S., Sheth, A.: Semantic Information Requests Using Domain Modeling and Resource Characteristics. In Cooperative Information Systems Proceedings of the 9th International Conference, CoopIS 2001, Trento, Italy, September 2001, Springer-Verlag, Berlin Heidelberg New York, 2001
- [TUM06] Tu, S., Abdelguerfi, M. : Web Services for Geographic Information Systems. In IEEE INTERNET COMPUTING: 12-14, 09/10 2006
- [ANB04] Antoniou, G., Boley, H. (eds) Rules and RuleMarkup Languages for the Semantic Web. Proceedings of the Third International Workshop RuleML 2004 Hiroshima, Japan, November 2004 Springer-Verlag, Berlin Heidelberg New York, 2004
- [PET06] C. Petrie It's the Programming, Stupid. In IEEE INTERNET COMPUTING: 96-97, 05/06 2006
- [MCC06] R. McCool Rethinking the Semantic Web, Part II In IEEE INTERNET COMPUTING: 96-97, 01/02 2006
- [MCC05] R. McCool Rethinking the Semantic Web, Part I In IEEE INTERNET COMPUTING: 88-90, 11/12 2005
- [PEB06] C. Petrie, C. Bussler Industrial Semantics and Magic. In IEEE INTERNET COMPUTING: 96-98, 07/08 2006
- [HEP06] M. Hepp Semantic Web and Semantic Web Services - Father and Son or Indivisible Twins? In IEEE INTERNET COMPUTING: 85-88, 03/04 2006
- [BRA07] Bratt, S.: Semantic web, and other technologies to watch. <https://www.w3.org/2007/Talks/0130-sb-W3CTechSemWeb/>, 2007.
- [SAL03] K. Sattler, F. Leymann Information Integration & Semantic Web. Datenbank-Spektrum: 5-6, 6/2003

8 Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 2.1 Heterogenität auf syntaktischer Ebene | 5 |
| Abbildung 2.2 Beispiel 4 [MCC05] | 6 |
| Abbildung 2.3 multilateraler Konflikt | 8 |
| Abbildung 2.4 Grafische Darstellung des multilateralen Konfliktes..... | 8 |
| Abbildung 3.1 Datenkonflikte | 10 |
| Abbildung 3.2 Domänenkonflikte | 11 |
| Abbildung 4.1 Wrapper und Mediatoren | 14 |
| Abbildung 4.2 Erfassen der Bedeutung über die Struktur..... | 15 |
| Abbildung 4.3 Matching über gemeinsames semantisches Modell..... | 16 |
| Abbildung 4.4 Erfassen der Bedeutung über natürlichsprachliche Verarbeitung..... | 19 |
| Abbildung 5.1 Beispiel 5 [SvH05]..... | 22 |
| Abbildung 6.1 Semantisches Schichtenmodell [BRA07] | 25 |

9 Lösungen der Selbsttestaufgaben

Lösung zu Selbsttestaufgabe 6.1

Heterogenitätskonflikte können auf folgenden Ebenen auftreten:

- Datenmodellebene
- Datenschemaebene
- Dateninstanzebene

Lösung zu Selbsttestaufgabe 6.2

Diese Konflikte sind bedingt durch die Verwendung unterschiedlicher Modellierungselemente zur Repräsentation von Information derselben Art. In konzeptionellen Datenmodellen sind diese Grundelemente Dateneinheiten (entities), Attribute und Daten, in RDF-Ressourcen sind es Ressourcen, Eigenschaften und Datentypen/Literale. Die Vermischung dieser Modellierungselemente bei der Repräsentation kann zu Konflikten führen.

Lösung zu Selbsttestaufgabe 6.3

Es liegt „Surjective Mappings“ vor, wenn ein Wert einer Quelle auf mehrere Werte einer anderen Quelle abgebildet werden muss, z. B. „Luxury“ und die Angabe von Sternen. Das Problem liegt demnach darin, dass die der Abstraktion zugrunde liegenden Werte-Systeme nicht dieselbe Anzahl an Werten besitzen.

Lösung zu Selbsttestaufgabe 6.4

Mit Disambiguierung ist die Auflösung sprachlicher Mehrdeutigkeit und die Ermittlung der beabsichtigten Bedeutung der Terme im jeweiligen Zusammenhang gemeint.

Lösung zu Selbsttestaufgabe 6.5

Ontologien sind ein Mittel zur Konzeptionalisierung bzw. zur Beschreibung von komplexen Konzepten über das Wissen einer Domäne und sind eine wesentliche Methode für die interoperable Repräsentation im Semantic Web.

000 000 000 (00/23)

00000-0-00-S1

Alle Rechte vorbehalten
© 2023 FernUniversität in Hagen
Fakultät für Mathematik und Informatik

Prof. Dr. Matthias Hemmje

01877

Dokumenten- und Wissensmanagement im Internet

Kurseinheit 7:
Ontologiesprachen für das Internet

Fakultät für
Mathematik und
Informatik

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere das Recht der Vervielfältigung und Verbreitung sowie der Übersetzung und des Nachdrucks, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Kein Teil des Werkes darf in irgendeiner Form (Druck, Fotokopie, Mikrofilm oder ein anderes Verfahren) ohne schriftliche Genehmigung der FernUniversität reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden. Wir weisen darauf hin, dass die vorgenannten Verwertungsalternativen je nach Ausgestaltung der Nutzungsbedingungen bereits durch Einstellen in Cloud-Systeme verwirklicht sein können.

Der Inhalt dieses Studienbriefs wird gedruckt auf Recyclingpapier (80 g/m², weiß), hergestellt aus 100 % Altpapier.

Inhaltsverzeichnis

| | |
|---|-----|
| Inhaltsverzeichnis..... | III |
| 1 Einleitung | 4 |
| 2 Einführung: Ontologie-basierendes Informationsmanagement..... | 4 |
| 3 Ontologien | 6 |
| 4 Die Rollen von Ontologien bei der Informationsintegration | 19 |
| 5 Fazit | 26 |
| 6 Literaturverzeichnis..... | 27 |
| 7 Abbildungsverzeichnis | 30 |
| 8 Lösungen der Selbsttestaufgaben | 31 |

1 Einleitung

Dies ist die siebte Kurseinheit des Kurses 1877 „Dokumenten- und Wissensmanagement im Internet“.

Als Ausgangspunkt des Ontologie-basierenden Informationsmanagements differenzieren die Studierenden zwischen drei Konzepten der semantischen Heterogenität. Der Begriff der Ontologie sowie ihr Aufbau, auch im Vergleich zur Taxonomie, werden näher betrachtet und ihre Rolle bei der Reduktion von semantischer Heterogenität durch Konzeptualisierungen und gemeinsam genutztes Vokabular beleuchtet.

Die Kursteilnehmenden unterscheiden und evaluieren verschiedene Anwendungsbiete für Ontologien, wie beispielsweise in der Systementwicklung, wo Standardisierungen erforderlich sind, um die Kompatibilität und Interoperabilität zwischen verschiedenen Systemen zu erreichen.

Anschließend untersuchen die Absolventen und Absolventinnen des Kurses die verschiedenen Rollen von Ontologien bei der Informationsintegration, und zwar als Repräsentationsarchitekturen, als mögliches Query-Modell und zur Verifikation von Informationsquellen.

Sie befassen sich mit der Frage nach einem einheitlichen und ausdrucksstarken Framework für die Informationsnutzung, das mithilfe von entsprechenden Tools die Voraussetzung für die Abfrage, die Interpretation und die Integration der verschiedenen Informationsressourcen schafft, und können diese einschätzen.

Abschließend werden der Übersetzungsprozess und das Ontologie-Alignment betrachtet, wodurch die Interoperabilität zwischen semantischen (Web-)Applikationen gewährleistet werden soll. Die Begriffe Ontology Mapping und Ontology Merging sind nach der Bearbeitung ebenfalls bekannt und können kontrastiert werden.

2 Einführung: Ontologie-basierendes Informationsmanagement

Der rege genutzte Informationsaustausch über das World Wide Web (im Folgenden: WWW) hat eine enorme Bedarfssteigerung nach inhaltsbasiertem Wissen ausgelöst. Beispielsweise ergibt eine Stichwortsuche über Suchmaschinen zu viele und zu heterogene Ergebnisse oder aufgrund der Globalisierung wird eine effektivere Kollaboration verschiedener Organisationen notwendig. Im weiteren Verlauf dieser Kurseinheit wird den zentralen Fragen nachgegangen, wie eine maschinenauswertbare Semantik dargestellt werden kann, um die Sachverhalte eines Anwendungsbereichs hinreichend aussagekräftig zu beschreiben und warum eine Standardisierung des Vokabulars für die gemeinsame Informationsnutzung notwendig ist.

Semantische Heterogenität

Das im WWW vorhandene Wissen stellt für sich allein betrachtet noch keine Information dar. Damit Wissen zu Information wird, müssen auch die wechselseitigen ideellen Beziehungen klar sein. Um Wissen als Information im Kontext einer Anwendungsdomäne, einer Aufgabe oder eines Problems zu nutzen, müssen die Verbindungen, die das Zusammenwirken von Begriffen beschreiben, leicht abrufbar, aktualisierbar und manipulierbar sein [KUR93]. Wenn man im schwach strukturierten WWW vorhandenes Wissen als eine gemeinsame Informationsbasis nutzen möchte, muss man sich zunächst des komplexen Begriffs der Heterogenität bewusst sein.

Heterogenität

Neben der strukturellen, syntaktischen und systemtechnischen Heterogenität verkörpert die semantische Heterogenität, die sich auf die unterschiedlichen Konzepte und ihre Interpretationen bezieht, das größte Problem für die automatische Wissensverarbeitung [CUO00]. Während Menschen aufgrund ihrer kognitiven Fähigkeiten den Kontext einer Wissensressource auf vielfältige Arten ab- bzw. herleiten können, sind technische Systeme nur dann dazu in der Lage, wenn die Ressourcen vorher formal explizit beschrieben worden sind. Die semantische Heterogenität kann in drei verschiedene Typen von Konzepten klassifiziert werden (vgl. Kurseinheit 1) [CUO00]:

1. semantisch äquivalente Konzepte:

Bei semantisch äquivalenten Konzepten können sich etwa verschiedene Begriffe auf ein und dasselbe Konzept beziehen oder verschiedene Eigenschaften für das gleiche Produkt von zwei Systemen modelliert werden.

2. semantisch unabhängige Konzepte:

Bei semantisch unabhängigen Konzepten kann es zu Verständnisproblemen kommen, wenn der gleiche Begriff für unterschiedliche Konzepte verwendet wird. Als Beispiel kann „Käfer“ entweder für ein „Tier“ oder für ein „Auto“ stehen.

3. semantisch abhängige Konzepte:

Bei semantisch abhängigen Konzepten wird die Heterogenität anhand von Generalisierung und Spezifizierung beschrieben. Als Beispiel dazu kann hier der Fall angeführt werden, wenn ein System nur Früchte kennt, ein anderes aber Äpfel, Orangen und Bananen etc. unterscheiden kann. Ebenso können Skalierungskonflikte auftreten, wenn etwa ein System „Kind“ als Menschen im Alter zwischen 5 und 12 Jahren klassifiziert, ein anderes System aber „Kind“ als Menschen zwischen 3 und 10 Jahren klassifiziert und ein drittes System etwa als Menschen unter 140 cm klassifiziert.

Interoperabilität

Die durch die semantische Heterogenität entstehenden Konflikte erschweren die Interoperabilität zwischen Softwareagenten oder Webservices, die Information nicht nur wiederverwenden sondern auch verarbeiten wollen. Missverständnisse, hervorgerufen durch Begriffe und ihre Attribute, kann man durch Vereinbarungen

und Standardisierungen eliminieren. Ontologien reduzieren das Problem der Heterogenität und können daher für die gemeinsame Informationsnutzung eingesetzt werden [BCB05].

Selbsttestaufgabe 7.1:

Was wird unter der semantischen Heterogenität verstanden?

Selbsttestaufgabe 7.2:

Nennen und erklären Sie die drei Konzepte der semantischen Heterogenität.

3 Ontologien

Der Begriff Ontologie

Ontologie

Gegenstand einer Ontologie (gr. óntos = Sein, lógos = Lehre) ist die Untersuchung von Kategorien von Dingen, die in bestimmten Interessensgebieten existieren oder existieren können. Das Ergebnis einer solchen Untersuchung ist ein Katalog von Typen von Dingen, von denen man annimmt, dass sie aus der Sicht einer Person, die eine Sprache L benutzt, um über einen Interessensbereich I zu sprechen, innerhalb dieses Interessensbereichs I existieren [SOW03].

Die Forschungsarbeiten auf dem Gebiet der Künstlichen Intelligenz haben den Begriff „Ontologie“ für die Repräsentation und maschinelle Verarbeitung von komplexen Wissensstrukturen aus der Philosophie entliehen und adaptiert. Daher hat der Begriff „Ontologie“ innerhalb der Informatik eine ganz andere Bedeutung als in der Philosophie. Die meistzitierte Definition einer Ontologie stammt von Tom Gruber [GRU93]:

„An Ontology is an explicit specification of a conceptualisation.“

Konzeptualisierung

Eine Konzeptualisierung bezieht sich auf ein abstraktes Modell darüber, wie sich Menschen die Dinge in der realen Welt vorstellen, wie z. B. einen Stuhl. Eine explizite Spezifizierung gibt den Konzepten und Relationen des abstrakten Modells explizit Namen und Definitionen [VIS02]. Grubers Definition ist oft noch um drei weitere Eigenschaften erweitert worden:

„An ontology is an explicit, formal specification of a shared conceptualization of a domain of interest.“

„Formal“ meint in diesem Sinne, dass die Ontologie maschinenlesbar sein soll.

„Shared“ bezeichnet, dass die Ontologie Wissen erfasst, das auf gegenseitiger Übereinstimmung beruht. Das Wissen ist nicht privat, sondern wird von einer Gruppe gemeinschaftlich akzeptiert und genutzt.

A ‘domain of interest’ gibt an, dass man nicht daran interessiert ist, die ganze Welt zu modellieren, sondern nur die für die Arbeit und Aufgabe relevanten Teile.

Eine Ontologie stellt ein Vokabular zur Darstellung eines Wissensbereichs bereit, indem

Prädikate verwendet werden, welche die Beziehungen zwischen den Entitäten herstellen.

Aufbau einer Ontologie

Das Problem der semantischen Heterogenität kann reduziert werden, wenn alle Begriffe, die eine Ontologie beschreiben, möglichst eindeutig und nicht-redundant definiert werden und weltweit gültig sind [MET03]. Die einfachste vorstellbare Ontologie (siehe Abbildung 3.1: Ontologien und ihre Verwandten [SHA06]), nämlich ein kontrolliertes Vokabular, wird dargestellt durch eine Liste von Begriffen, die von einer Registrierungsinstanz verwaltet wird. Wörterbücher und Lexika beispielsweise beschreiben Begriffe informell eindeutig und werden allgemein für ein besseres Verständnis zwischen Menschen und Organisationen genutzt.

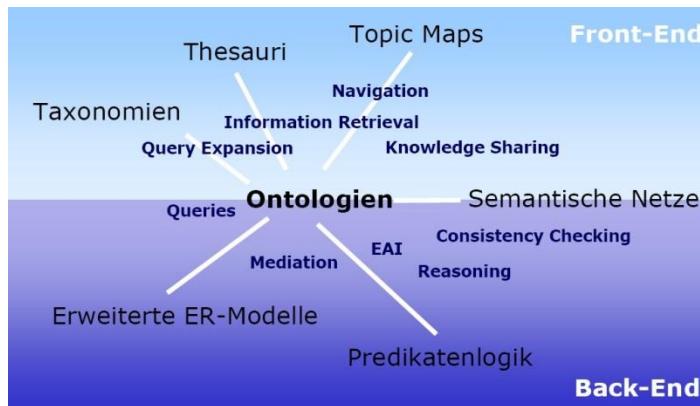


Abbildung 3.1 Ontologien und ihre Verwandten [SHA06]

Im grundsätzlichen Aufbau besteht eine Ontologie aus mehreren Bestandteilen. Um Strukturen zu erstellen werden im Folgenden die minimalen Bestandteile einer Ontologie genannt:

- **Konzepte (Klassen):** Konzepte fassen real existierende Instanzen mit gemeinsamen Eigenschaften zusammen.
- **Instanzen (Begriffe):** Instanzen stellen die eigentlichen Objekte dar.
- **Relationen:** Relationen verbinden Konzepte und Instanzen miteinander.

Einige Beispiele für Ontologien, basierend auf diesen Bestandteilen, werden im Folgenden erläutert.

Taxonomie

Die Taxonomie ist eine der ältesten wissenschaftlichen Disziplinen in der Medizin und der Biologie (vgl. Abbildung 3.2: Auszug aus einer Insektenklassifikation). Mit Taxonomien (griech. Taxis = Ordnung, -nomia = Verwaltung) werden Mengen von Konzepten von kontrollierten Vokabularen eines Themengebiets definiert und in eine hierarchische Beziehung gesetzt, um eine möglichst präzise Beschreibung und Repräsentation eines Themengebiets zu erhalten. Es bestehen etwa internationale Regelwerke zur Nomenklatur für die Zoologie (*International Code of Zoological Nomenclature - ICZN*).

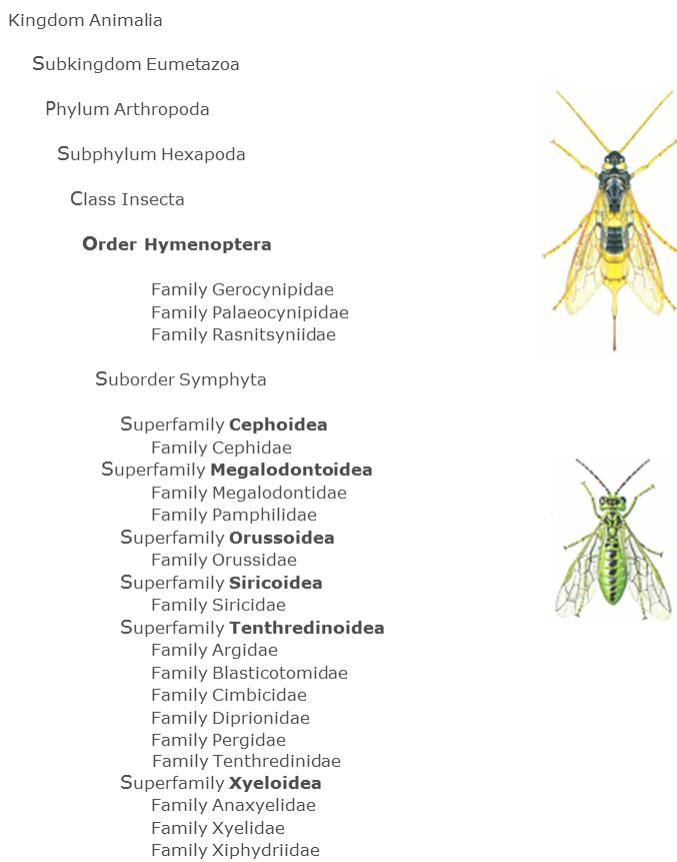


Abbildung 3.2 Auszug aus einer Insektenklassifikation

Entity-Relationship-Modelle

Innerhalb der Informatik bilden die Schemata von Datenbanksystemen eine konzeptuelle Datenbeschreibung in Form von Entity-Relationship-Modellen oder UML-Diagrammen. Sie bieten eine Beschreibung von Begriffen und ihren Beziehungen an, die direkt in eine Ontologie umgesetzt werden können.

Um dafür ein Beispiel zu nennen (vgl. Abbildung 3.3), kann eine zwischen den Klassen ‚Studenten‘ und ‚Vorlesung‘ definierte Assoziation ‚ hören‘ direkt in eine entsprechende semantische Relation in einer zugehörigen Ontologie umgesetzt werden können.

Im Vergleich zu Ontologien sind aber die Beschreibungen konzeptueller Datenbankschemata als eingeschränkter zu bezeichnen in Bezug auf die zur Verfügung stehenden sprachlichen Ausdrucksmittel [SUR04]. Darüber hinaus sind konzeptuelle Datenbankschemata im Allgemeinen nicht auf eine gemeinsame Nutzung durch verschiedene Gruppen von Benutzenden ausgerichtet, sondern vielmehr auf eine Beschreibung der Inhalte einer Datenbank. Ferner sind konzeptuelle Datenbankschemata typischerweise nicht Bestandteil des laufenden Systems.

konzeptuelle
Datenbankschemata

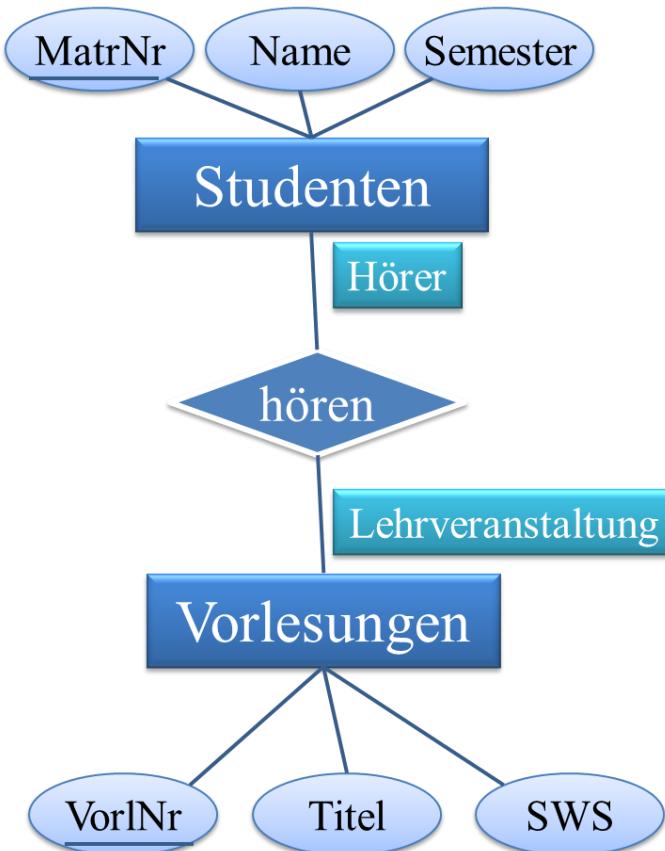


Abbildung 3.3 ER-Modell anhand einer Vorlesung

Ein Beispiel für eine Ontologie, die Bestandteil des laufenden Systems ist, stellt die FOAF-Ontologie (*Friend of a friend*) dar, ein Projekt zur maschinenlesbaren Modellierung sozialer Netzwerke. Personen können hier über ein FOAF-Dokument Format-Angaben über eine Person veröffentlichen. In Abbildung 3.4 ist ein Koautoren-Netzwerk als Beispiel dargestellt. Die FOAF-Dokumente beruhen auf einem spezifizierten FOAF-Vokabular, das mit der Ontologiesprache (RDF/OWL) beschrieben wird. Außer der Abstraktion des FOAF-Vokabulars werden noch ein paar mathematische Regeln verwendet, die den FOAF-Entwurf unterstützen.

FOAF-Ontologie

Ähnlich wie Webseiten können FOAF-Dokumente mit anderen FOAF-Dokumenten kombiniert werden und so eine zusammenhängende informative Datenbasis erzeugen.¹

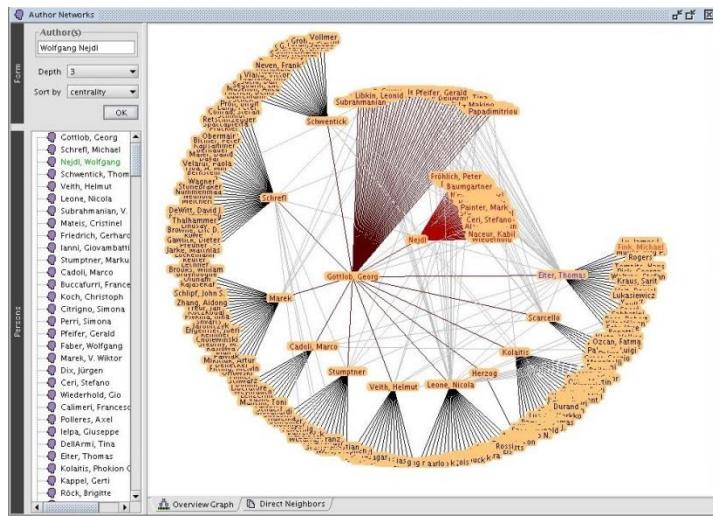


Abbildung 3.4 FOAF Beispiel anhand einer Co-Autorenbeziehung (<http://ezdl.de>)

Semantic Web

Ontologien im *Semantic Web* bestehen in der Regel aus vernetzten Taxonomien und einer Reihe von Inferenzregeln, anhand derer Maschinen logische Schlüsse ziehen können.

Selbsttestaufgabe 7:3:

Erklären Sie den Begriff der Taxonomie.

Konzeptualisierungen und gemeinsam genutztes Vokabular

Wie schon Benjamin Lee Whorf in seinen Beiträgen zur Metalinguistik und Sprachphilosophie erläutert hat, ist der menschliche Wahrnehmungsapparat ganz darauf ausgerichtet, Bereiche von Phänomenen zu ignorieren, die so selbsterklärend und normal sind, dass sie für unser alltägliches Leben keine besondere Rolle spielen [WHO69]. Dies führt dann dazu, dass Vokabular für die Sprechenden einen Hintergrundcharakter besitzt. Nach Whorf ist als Voraussetzung für die Kommunikation eine Übereinstimmung dieser Hintergrundcharaktere einer Sprache notwendig. In der Informatik unterstützen formale Ontologien die Kommunikation zwischen menschlichen und/oder maschinellen Akteuren und erleichtern somit den Austausch und das Teilen von Wissen in Unternehmen. Zur Reduktion der semantischen Heterogenität sind zwei Anforderungen zu stellen.

¹ <http://xmlns.com/foaf/0.1/>

1. Einerseits ist die Einigung einer Gruppe von Anwendern auf die jeweiligen Begriffe und deren Zusammenhänge auf der Wissensebene nötig.
2. Andererseits ist die Einigung auf ein Vokabular nötig, das den Verweis auf die Begriffe und so die Repräsentation der Konzepte eindeutig identifiziert [DJK04].

Konzeptualisierung

Konzepte

Die semantische Grundlage für die Kommunikation sind Konzepte ([OGR23], [TRA96]). Ein Konzept ist eine gedankliche Vorstellung bzw. die subjektive Interpretation über ein reales Ding in der Welt. Die Gesamtheit aller Konzepte bildet mit den Symbolen eine Brücke zwischen der impliziten gedanklichen Vorstellung eines Weltbildes und des physikalischen expliziten Modells der Welt.

Problematisch kann sich dabei die Zuordnung des Symbols auf den Gegenstand erweisen, wenn diese nicht mehr eindeutig ist. So kann das Symbol „Käfer“ sowohl dem Konzept „Auto“ als auch dem Konzept „Tier“ zugeordnet werden [JAN03]. Hier sei auf die Problematik der Homonyme und Synonyme hingewiesen.

Homonyme und Synonyme

Die Semantik ergibt sich aus der Verknüpfung eines Symbols mit einem Konzept, das etwas beschreibt. Eine Konzeptualisierung im Sinne von Gruber hat nun die Aufgabe, die Semantik im Umfeld der technischen Informationssysteme so eindeutig zu formulieren, dass kein Interpretationsspielraum übrig bleibt und trotzdem keine Semantik verloren geht.

Sicherlich muss beachtet werden, dass die natürliche Sprache keinen hinreichenden Zeichenvorrat zur Verfügung stellt, um alle denkbaren Konzepte so eindeutig zu benennen [DJK042]. Wie kann nun die minimale ontologische Vereinbarung, die die Entwicklung einer Ontologie so wenig wie möglich einschränkt, um die Freiheit zur Spezifikation zu ermöglichen, erzielt werden?

Uniform Resource Locator

Für den Ausschluss von Homonymen und Synonymen findet eine allgemein anerkannte Vereinbarung eines Namens als eindeutige Referenz auf das Konzept Verwendung. Eine solche Referenz ist z. B. ein *Uniform Resource Locator* (URL) wie er in dem RDF-Konzept eingesetzt wird. Die ontologische Zustimmung beruht auf der Benutzung eines konsistenten Vokabulars [GRU93]. Eine Minimierung der ontologischen Zustimmung kann durch eine sehr schwache semantische Ontologie spezifiziert werden und nur die Begriffe eines Vokabulars werden dabei definiert, die unbedingt notwendig sind für die Kommunikation. Das RDF-Konzept ist ein Beispiel für eine schwach spezifizierte Theorie.

Gemeinsam genutztes Vokabular

DF verwendet *Uniform Resource Identifiers* (URIs) zum Identifizieren der Ressourcen und verknüpft sie so mit einer eindeutigen Definition. Allerdings

RDF-Schema RDF-Triples

können verschiedene RDF-Instanzen dieselben Sachverhalte in unterschiedlicher Weise ausdrücken. Mit RDF-Schema (RDFS) kann ein Vokabular erstellt werden, das Gruppen von verwandten RDF- Ressourcen und ihre Beziehungen zwischen diesen Ressourcen auf einer höheren Abstraktionsebene flexibel beschreibt. Gleichzeitig bestehen RDFS-Triples aus Klassen, Klasseneigenschaften und Werten, sodass aufgrund von Typeigenschaften die Eindeutigkeit von Sachverhalten sichergestellt werden kann. Mit RDFS kann ein domänenspezifisches Vokabular erstellt werden. Ein Abgleich von Aussagen verschiedener Domänen erfordert noch zusätzliche Axiome wie etwa `subClassOf`, `disjointWith` und Klassenbeschreibungen wie `unionOf`, `intersectionOf` etc., die von der Ontologiewebsprache OWL bereitgestellt werden.

Vokabularen

| Die sinnvolle Definition von Vokabularen, die allgemein von der Mehrzahl verstanden und benutzt werden, muss kohärent und konsistent sein für die Durchführung von qualifizierten Suchanfragen und für den Austausch von Informationen durch Softwareagenten. Die sich ein gemeinsames Vokabular teilenden Agenten müssen sich aber nicht die Wissensbasis teilen. Denn jeder kennt Dinge, die ein anderer möglicherweise nicht kennt. Zudem muss ein Agent, der einer Ontologie zustimmt bzw. verpflichtet ist, nicht unbedingt alle Fragen beantworten, die man mit einem gemeinsam genutzten Vokabular stellen kann [GRU93]. Indem das in der Ontologie definierte Vokabular benutzt wird, stimmt man mit einer Ontologie überein. Sie ist eine Garantie für Konsistenz, aber nicht für die Vollständigkeit des verwendeten Vokabulars.

Aufgrund der natürlichen Umweltveränderungen entstehen auch zukünftig neue Begrifflichkeiten, die von Ontologien für bestimmte Aufgaben oder Repräsentationen aufgenommen werden müssen. Deshalb müssen Ontologien offen für Erweiterungen sein, ohne dass eine Revision der existierenden Definitionen notwendig wird [GRU93]. RDF-Schema und OWL-Full sind offen für weitere Vokabular-Definitionen. Das Hinzufügen von Vokabeln erweitert den Bereich der gemeinsam genutzten Vokabeln und kann dadurch die ontologische Zustimmung erhöhen.

Top-Level-Ontologien

| Da die Erstellung einer Ontologie mit der Standardisierung der Terminologie sehr mühsam ist, wurden verschiedene Top-Level-Ontologien entwickelt, die sehr allgemeine Konzepte beschreiben und so die Nachhaltigkeit beim Einsatz von Ontologien gewährleisten. Ein Beispiel ist die EuroWordNet Top-Ontologie, die die verschiedenen Sprachkonzepte der europäischen Länder über eine sprachunabhängige Ebene abgleicht. Das Ziel von EuroWordNet ist der Aufbau einer multilingua- len Datenbasis, die ein Vokabular für mehrere Sprachen bereitstellt.

Upper-Ontologie

| Dublin Core ist ein anderes Beispiel für die Bereitstellung einer Menge von standardisierten Metadaten für die Veröffentlichung von Literatur. Sie stellt als Vokabular eindeutige URLs für die wichtigsten Metadatentypen zur Verfügung. Spezifizierte Metadaten-Vokabularen für die Beschreibung von Ressourcen wie beispielsweise Autor, Datum, Inhalt etc. verbessern die automatische Suche sowie die

Abfrage von unstrukturierten Informationen und Dokumenten. Cyc-Ontologie ist ein kommerzielles Produkt, wobei ein kleiner Teil der Upper-Ontologie, der die Konzepte des Alltagswissens enthält, frei verfügbar ist. DOLCE stellt eine weitere frei verfügbare Ontologie-Bibliothek von existierenden Ontologien dar.

Die Spezifikation eines Wissenskontexts

Die Spezifikation eines Wissenskontexts bedeutet, dass man sich gegenseitig darauf verständigen und einigen muss, welche Bedeutung die Entitäten haben, die den Inhalt einer Web-Informationsressource repräsentieren. Es haben sich in verschiedenen Gebieten verschiedene Ontologien entwickelt, die überlappende Informationen enthalten. Nach Jasper und Uschold [USJ99] unterscheiden sich Ontologien nach dem Grad der Formalität und nach dem Maß der Definitionstiefe

Grad der Formalität

Der Grad der Formalität bestimmt den Grad der semantischen Interoperabilität, der Fähigkeit zur Zusammenarbeit heterogener Informationssysteme nicht nur syntaktischer, sondern auch inhaltlicher Art. Die Formalität einer Sprache, die eine Ontologie repräsentiert, lässt sich in vier Bereiche einteilen:

semantische
Interoperabilität

- **Total-informell**

Eine Ontologie wird einfach in natürlicher Sprache beschrieben, viele Glossare passen in diese Kategorie.

- **Semi-informell**

Die Ontologie wird in einer strukturierten Form mittels einer natürlichen Sprache dargestellt, die Mehrdeutigkeiten reduziert. Dies kann z. B. eine Taxonomie, also ein Wortschatz sein, der in einer Generalisierungs-Spezialisierungs-Hierarchie strukturiert ist.

- **Semi-formal**

Eine semi-formale Sprache ist eine künstlich formal definierte Sprache, die die Beschreibung der Begriffe sowohl informell in einer natürlichen Sprache als auch formal in einer für Computer interpretierbaren Wissensrepräsentationssprache unterstützt [FFR97].

semi-formale
Sprache

- **Regelsysteme**

Eine Taxonomie, die mit einem Regelsystem implementiert ist, ermöglicht die schnelle und einfache Darstellung von Wissen in Regeln. Als Beispiele können hier Expertensysteme und Produktionsysteme angeführt werden.

Regelsystem

- **Framekonzepte**

Klassenframes

Frames repräsentieren ein mögliches Modell zur Darstellung von Konzepten [LOE03]. Durch die Kombination der in den Frames enthaltenen Informationseinheiten (Framename, Slots und Filler) wird Wissen über die Konzepte zur Verfügung gestellt. In diesem Fall spricht man auch von Klassenkonzepten bzw. Klassenframes. Einzelframes werden als Instanzen von Klassenframes bezeichnet. Filler sind so genannte Standardwerte, die Slots einnehmen können. Als ein Beispiel dafür könnte folgendes dienen:

- Klassenframe: Apfel
- Slots: Form, Farbe, Größe, Material, Geschmack
- Filler: rund, rot, apfelfroß, Fruchtfleisch, süßsauer

Die Framekonzepte ähneln dem Klassenkonzept der objekt-orientierten Programmierung (OOP).

- **Semantische Netzwerke**

In semantischen Netzwerken werden Konzepte als Knoten, Relationen als Kanten visualisiert. Der Vorteil liegt in der Widerspiegung der semantischen Nähe zwischen den Konzepten innerhalb eines semantischen Netzes. Inhaltlich ähnliche Konzepte liegen nah aneinander, während unterschiedliche Konzepte durch eine Reihe von Zwischenknoten verbunden sind. Diese Eigenschaft ermöglicht einen besonders schnellen Zugriff auf die mit einer etwaigen Anfrage inhaltlich verwandten (assoziativen) Informationen.

- **Rigoros formal**

Description logics

Eine auf der Prädikatenlogik basierende formale Sprache treibt den Standardisierungsprozess voran und ermöglicht die Entwicklung von Schlussfolgerungskalkülen, welche gültige Eigenschaften von Mengen von Ausdrücken ableiten, sodass Fragen der Vollständigkeit und Korrektheit von formalen Spezifikationen überprüft werden können. Beschreibungslogiken (*description logics*) sind eine Familie von Sprachen zur Wissensrepräsentation. Die meisten Beschreibungslogiken sind eine Untergruppe der Prädikatenlogik erster Stufe, im Gegensatz zu dieser aber entscheidbar. Dies ermöglicht Schlussfolgerungen, d. h. aus vorhandenem Wissen kann neues Wissen generiert werden. Beschreibungslogiken richten sich nach dem *Open World Prinzip*. Das Open World Prinzip verwendet weder eindeutige Bezeichner noch wird angenommen, dass mangelndes Wissen über einen Sachverhalt (etwa bei

Prädikatenlogik

fehlenden Einträgen in einer Datenbank) automatisch die Negation dieses Sachverhalts impliziert. Statt in einem solchen Fall also den Wert „falsch“ zurückzugeben², gibt das Open World Prinzip den Wert „weiss nicht“ zurück. Das Open World Prinzip geht also grundsätzlich davon aus, dass Informationen unvollständig spezifiziert sind. und ist daher gut geeignet für die Beschreibung von Wissen, das sich iterativ erweitern kann [DRS06]. KIF (*Knowledge Interchange Format*) ist entwickelt worden, um verschiedenen wissensbasierten Systemen den Wissensaustausch zu ermöglichen. Es basiert auf dem Prädikatenkalkül.

Knowledge
Interchange Format

Je höher der Grad der Formalität einer Sprache ist, desto stärker ist der Standardisierungsprozess [DIE94], desto strukturierter ist die Information und desto nützlicher kann eine Ontologie sein [GUA98].

Das Ausmaß der Definitionstiefe

Das Ausmaß der Definitionstiefe ist von dem Zweck abhängig, die eine Ontologie erfüllen soll. Es gibt kleinere leichtgewichtige Ontologien, die keine komplexen Strukturen enthalten wie z. B. Produktkategorisierungen. Die schwergewichtigen Ontologien können sich aus mehreren kleinen leichtgewichtigen Ontologien durch *Merging* bilden oder es sind umfangreiche Ontologien wie die so genannten Toplevel- Ontologien DOLCE oder CyC, die viele komplexe Schlussfolgerungen unterstützen müssen.

leichtgewichtige
Ontologien

schwergewichtige
Ontologien

Im Folgenden werden kurz Webstandards vorgestellt, die Wissen implementierungsabhängig von einer Programmiersprache darstellen. Die Sprachen werden anhand der Ausdrucksstärke, der Inferenzmächtigkeit und der Entscheidbarkeit miteinander verglichen [OWL04].

Webstandards

- XML (*eXtensible Markup Language*) stellt eine Syntax für strukturierte Dokumente zur Verfügung, nutzt aber keine semantische Eingrenzung der Bedeutung dieser Dokumente.
- XML-Schema ist die Sprache, um die Struktur von XML-Dokumenten festzulegen und erweitert XML auch um Datentypen.
- RDF (*Resource Description Framework*) ist ein Datenmodell für Ressourcen und Relationen zwischen diesen. Es stellt eine einfache Semantik für das Datenmodell zur Verfügung, die in XML repräsentiert werden kann.

² Das Gegenstück zum Open World Prinzip wird Closed World Prinzip genannt.

- RDF-Schema stellt ein Vokabular zur Beschreibung der Eigenschaften und Klassen von RDF-Ressourcen mit einer Semantik für verallgemeinernde Hierarchien solcher Eigenschaften und Klassen bereit.
- DAML+OIL: DAML-ONT (*Darpa Agent Markup Language-Ontology*) und OIL (*Ontology Inference Layer*) fließen in DAML+OIL zusammen. Die Beschreibungssprache kann als eine Ontologie-Infrastruktur für das *SemanticWeb* betrachtet werden. OIL basiert auf Konzepten, die in Beschreibungslogik und frame-basierten Systemen entwickelt worden sind. Sie ist somit entscheidbar und unterstützt effizientes *Reasoning*. OIL ist kompatibel mit RDFS [BKD00].
- OWL: Die *Web Ontology Language* ist eine Spezifikation des W3C und basiert historisch auf DAML+OIL. Sie fügt mehr Vokabeln zur Beschreibung von Eigenschaften und Klassen hinzu (Disjunktheit, Kardinalität, Äquivalenz etc). Innerhalb der OWL-Familie kann man drei Dialekte mit zunehmender Ausdrucksstärke unterscheiden:
 - OWL-Lite dient zum schnellen und einfachen Erstellen von Taxonomien und hat eine geringere formale Komplexität als OWL-DL.
 - OWL-Description Logic (OWL-DL) entspricht OIL und unterstützt diejenigen Nutzer, die eine maximale Ausdrucksmächtigkeit bei gleichzeitiger Erhaltung der Berechenbarkeit benötigen. Alle Schlussfolgerungen sollen garantiert berechenbar und entscheidbar sein. OWL-DL entspricht der Beschreibungslogik, deren Inferenzmächtigkeit durch hinreichende und notwendige Bedingungen eingeschränkt wird.
 - OWL-FULL ist die ausdrucksmächtigste logische Sprache, die prädikatenlogische Ausdrücke höheren Grades ermöglicht. Sie erlaubt es einer Ontologie, die Bedeutung von vordefiniertem RDF- bzw. OWL-Vokabular zu erweitern, allerdings ohne eine Garantie der Verarbeitbarkeit.

Mehr über die Ausdrucksmächtigkeit von semantischen *Markup*-Sprachen erfährt man in den Beiträgen *A Comparison of (Semantic) Markup Languages* [GIR02] und *A Roadmap to Ontology Specification Languages* [COG00].

Selbsttestaufgabe 7.4:

In welche vier Bereiche lässt sich die Formalität einer Sprache, die eine Ontologie repräsentiert, einteilen?

Anwendungsgebiete für Ontologien

Die Vorteile von Ontologien liegen zum einen in dem einheitlichen Sprachgebrauch, der den Datenaustausch zwischen verschiedenen Akteuren unterstützt. Zum anderen sind sie maschinenlesbar und verarbeitbar und ermöglichen somit die Darstellung komplexer Beziehungen zwischen Objekten in maschinenlesbarer Form. Die Anforderungen an die Formalität und Ausdrucksmächtigkeit einer Ontologie sind vom jeweiligen Anwendungsgebiet abhängig. Nach Jasper lassen sich die Anwendungsszenarien für Ontologien in vier klassische Gruppen gliedern:

- **Kommunikation**

Ontologien reduzieren die semantischen und terminologischen Begriffsverwirrungen und ermöglichen dadurch ein gemeinsames Verständnis und gemeinsame Kommunikation zwischen Menschen mit ihren verschiedenen Ansichten von Konzepten. Ontologien können als ein normatives Modell eingesetzt werden, in dem z. B. die unterschiedlichen Begrifflichkeiten für Geschäftsaktivitäten und Geschäftsobjekte vereinheitlicht werden, um eine automatische Zusammenführung von Geschäftsprozessen zu ermöglichen. Ontologien können ferner Beziehungsnetzwerke herstellen. Ein Beispiel ist die FOAF-Ontologie, die soziale Netzwerke ermöglicht und so zu hierarchiefreiem Wissensaustausch sowie zur Wissensgenerierung über das Internet als sog. „Mitmach-Netz“ beiträgt. Die Integration von Wissen, in dem Ontologien benutzt werden, kann an dem *ffPoirot*-Beispiel, einer mehrsprachigen Anwendung zur Bekämpfung von Mehrwertsteuer-Betrug (www.ffpoirot.org), dargestellt werden, wo Anwender eigene Erfahrungen zu finanziellen Problemen hinzufügen oder Wissen dazu abfragen können.

- **Technische Systementwicklung**

Standardisierungen im Umfeld der Informations- und Kommunikationstechnologie sind erforderlich, um die Kompatibilität und Interoperabilität zwischen verschiedenen Systemen zu erreichen. So ist die Kompatibilität der Daten für Zwecke, die auf Vergleichen beruhen, wie es bei Klassifikationen etwa üblich ist, notwendig, um Redundanzen und unnötig mühsame Arbeit zu reduzieren (ISO/TC)³. Mit Ontologien wird ein standardisiertes Grundvokabular definiert, welches später als Basis für die Anforderungsspezifikation und für die Entwicklung von Software dient. Neben der Wiederverwertbarkeit und Wartbarkeit unterstützt dieses Verfahren auch die Dokumentation der Software. Die uneingeschränkte Wiederverwendbarkeit von Ontologien ist das große Ziel vieler Top-Level-Ontologien, um so die Zuverlässigkeit und Konsistenz zu erhöhen. Ontologien sind offen für neue Konzepte und können leicht erweitert werden [USG96].

³ <https://www.iso.org/committee/45020.html>

- **Interoperabilität**

Zwei Applikationen sind interoperabel, wenn sie die terminologische Semantik in ihren korrespondierenden Konzepten gemeinsam nutzen und teilen [GRU05]. Sollen zwei Programme, als Beispiel können hier Web-Suchmaschinen oder Software-Agenten genannten werden, miteinander kommunizieren, so müssen sie entweder selbst die Interpretationsvorschrift für die Daten in sich tragen, also datenabhängig sein oder aber sie liefern diese in Form von Metadaten aus einer beiden Seiten zugänglichen Ontologie mit. Das *Knowledge-Interchange-Format* (KIF) ist entwickelt worden, um den automatisierten Wissensaustausch zwischen verschiedenen wissensbasierten Systemen zu ermöglichen. Viele Ontologiesprachen bauen auf dem KIF-Format auf oder sind kompatibel zum KIF-Format, sodass verschiedene Wissensspezifikationen (Ontologien) auf verschiedene, im Sinne von relationalen, objektorientierten, deduktiven oder anderen Datenbank-Schemata mittels KIF als Mediator übertragen werden können [GUA98].

- **Automatisches Schließen (Information Retrieval)**

| Information Retrieval |

Der Nachteil klassischer Information Retrieval Ansätze ist, dass eine rein syntaktische Suche nach Begriffen stattfindet, ohne dabei die Bedeutung der Wörter innerhalb der einzelnen Dokumente zu berücksichtigen. Durch die Benutzung der eindeutig spezifizierten Begriffe einer Ontologie für die Suchanfrage, etwa durch den Einsatz linguistischer Ontologien wie EuroWordNet, wird das Ergebnis der Suche eindeutiger und nicht so diffus.

Beim automatischen Schließen können Programme logische Schlüsse schon aufgrund der bekannten Ableitungsregeln der formalen Ontologie ziehen – diese müssen also nicht stets von neuem übermittelt werden. So steht z. B. hinter der Preisfeststellung für die Deutsche Bahn ein Regelwerk, das dem Kunden unbekannt ist, weil es wahrscheinlich ziemlich kompliziert ist. Ein Reasoner, der deklarative Regeln und automatische Schlussverfahren einsetzt, würde es ermöglichen a) dieses Regelwerk dem Kunden mitzuteilen und b) die Preisfestlegung dem Kunden zu erklären. Der Kunde hat so die Möglichkeit, mehr Kontrolle über die Auswahl von günstigen Zugverbindungen zu haben [BRY05].

Selbsttestaufgabe 7:5:

Nennen Sie zwei Anwendungsbeispiele für Ontologien.

4 Die Rollen von Ontologien bei der Informationsintegration

Im letzten Abschnitt sind die Vorteile der ontologiebasierten Informationsnutzung erläutert worden. In diesem Abschnitt werden nun die drei Rollen beschrieben, die Ontologien in der Informationsintegration einnehmen können:

- Ontologien werden hauptsächlich zur Repräsentation und Visualisierung von zum Teil unterschiedlich strukturierten Informationen eingesetzt.
- Ontologien können als ein Anfrage-Modell eingesetzt werden, das die Suche und das Browsing von Informationen vereinfacht, indem die Struktur der Ontologie selbst genutzt wird.
- Ontologien werden auch als Verifikationsmodell genutzt, denn durch die Spezifikation wird die Softwareentwicklung mit Ontologien konsistent.

Repräsentationsarchitekturen

Eine der Hauptrollen von Ontologien ist die Repräsentation ihrer Instanzen. Sie dienen als Informationsquellen, die ihre Instanzen als Ergebnis von Suchanfragen an den Nutzer liefern. Es gibt drei verschiedene Architektur-Ansätze, mit denen die zum Teil unterschiedlich strukturierten Informationsquellen (HTML-Seiten, Datenbanken, Textdateien) mit einer einheitlichen Semantik für den Nutzer dargestellt werden können. Man unterscheidet drei Arten von Ansätzen:

- Single-Ontology-Ansatz
- Multiple-Ontology-Ansatz
- Hybrid-Ansatz

Der Single-Ontology-Ansatz

Der Single-Ontologie-Ansatz benutzt eine einzige globale Ontologie und stellt ein gemeinsam zu nutzendes Vokabular für ein bestimmtes Fachgebiet bereit. Alle integrierten Informationsquellen in einem System müssen ihre Informationen mit dem einheitlichen Vokabular der globalen Ontologie präsentieren. Dies gelingt nur, wenn die zu übersetzenden Informationen eine nahezu gleiche Sicht auf das zu präsentierende Fachgebiet haben, weil sonst der Gültigkeitsbereich der Ontologie verletzt wird und so wie der Homonyme und Synonyme entstehen können. Aufgrund des einheitlichen Vokabulars ist der Vergleich der Informationsressourcen sehr einfach, hat aber den Nachteil der engen Kopplung bezüglich der ontologischen Übereinstimmung. Es ist fraglich, ob nicht trotzdem bestimmte terminologische Fehlanpassungen wie Hyponyme, Hyperonyme etc. auftreten können [SEK05]. Änderungen in der Informationsressource ziehen Korrekturen in der globalen Ontologie nach sich, wodurch die Skalierbarkeit der Ontologie, also die

Frage nach der einfachen Erweiterbarkeit des Integrationssystems mit neuen Informationsquellen, leidet. Die Wiederverwendung von Ontologien wird nicht unterstützt.

Ein allseits bekannter Ansatz der globalen Ontologie-Integration ist SIMS (Search In Multiple Sources), das 1993 erschienen ist und den Zugriff und die Integration verschiedener dynamischer Informationsquellen über einen Wrapper/Mediator ermöglicht [BCB05]. Suchanfragen werden mit einer SQL-ähnlichen Sprache wie LOOM mit den Begriffen der globalen Ontologie durchgeführt, die zunächst in das Konzept der ursprünglichen Informationsquelle übersetzt werden müssen. Anschließend wird der Queryplan generiert, wobei die Suchanfrage in zwei Unteranfragen zerlegt wird, deren Ausführungsplan noch optimiert wird.

Multiple-Ontology-Ansatz

mehrreiche Ontologien

Die Inflexibilität bei Informationsänderungen, die zu enge Kopplung und mangelnde Wiederverwendbarkeit von Ontologien in *Single-Ontology*-Ansätzen haben zu der Entwicklung des mehrfachen-Ontologie-Ansatzes geführt, wo jede Ressource durch ihre eigene Ontologie repräsentiert wird. OBSERVER (*Ontology Based System Enhanced with Relationship for Vocabulary heterogeneity Resolution*) ist ein Ansatz, der viele Informationsquellen über ihre speziellen Ontologien verwaltet. Die Architektur basiert auf Wrappern, Ontologie-Servern und einem *Inter-Ontologie-Relationship-Manager* (IRM).

Jede Ressource kann jede beliebige Ontologiesprache zur Repräsentation verwenden, ohne sich an ein gemeinsames Vokabular binden zu müssen. Dies vereinfacht zwar die Skalierung des Systems, ebenso hat das Hinzufügen einer zusätzlichen Ressource keine Änderungen des gesamten Systems zur Folge, dafür muss aber der Vergleich der Ressourcen sowohl auf der semantischen als auch auf der terminologischen Beschreibungsebene durchgeführt werden.

Die Auswertung der Ontologievergleiche wird vom IRM durchgeführt, der die semantischen Beziehungen zwischen den verschiedenen Ontologien mit den Begriffen der Synonyme, Hyponyme, Hypernyme, Subsumption, Disjunktheit, Overlap etc. identifiziert [BCB05]. Die Suchanfragen, die in Begriffen der lokalen Ontologie formuliert sind, müssen in die Anfragesprache aller anderen beteiligten lokalen Ontologien übersetzt werden. Der 1:1-Mapping-Ansatz lässt die Komplexität des Ontologie-Mappings auf $O(n^2)$ steigen, wobei n die Anzahl der Ontologien ist [SEK05]. Der Mangel eines gemeinsam genutzten Vokabulars erhöht die semantische Heterogenität, die eigentlich mit Ontologien beseitigt werden sollte.

Hybridansatz

Dieser Ansatz stellt eine Verbindung zwischen den beiden bereits vorgestellten Ansätzen zur Beschreibung von Informationen dar. Jedem Informationssystem wird seine eigene lokale Ontologie zugeordnet, wobei alle lokalen Ontologien auf

einem gemeinsamen globalen Vokabular basieren. Das Vokabular gibt die allgemeinen Begriffe systemweit vor, die dann für die einzelnen lokalen Ontologien entsprechend den Konzeptualisierungen zu komplexen Begriffen kombiniert werden. Die Flexibilität und Skalierbarkeit wird durch die lokalen Ontologien unterstützt, während das globale Vokabular als „Lingua Franca“ die Vergleichbarkeit der Applikationsontologien gewährleistet. Werden die verschiedenen Hybridansätze vernetzt, muss auch wieder ein ontologisches Mapping durchgeführt werden. Für die Integration und das Mapping der Ontologien wird ein Werkzeug bereitgestellt, dass die Informationen aus den heterogenen Informationssystemen integriert und kombiniert.

Das auf dem Hybridansatz basierende System COIN hat eine wrapper-mediatorbasierte Architektur. Wrapper kapseln die Informationssysteme, um eine einheitliche Schnittstelle auf beliebige Informationssysteme zu bieten. Die Mediatoren müssen die Heterogenitätskonflikte beseitigen und kombinieren bzw. in die Information integrieren [WHO69]. Als nachteilig ist hier der Wiedereinsatz von Ontologien zu nennen, da sie erst dem globalen Wortschatz angepasst werden müssen. Bei dem agentenbasierten System KRAFT oder Infosleuth wird die lose Kopplung und damit die Wiederverwendbarkeit von Ontologien unterstützt, indem Agenten den Austausch der Daten und ihre Konsolidierung im dynamisch verteilten System übernehmen [BCB05]. Die Suchanfragen in einem zentralen mediatorbasierten System, die in lokalen Ontologien erstellt werden, müssen wie beim OBSERVER für die unterschiedlichen Ressourcen passend umgeschrieben werden.

Hybridansatz

Query-Modell

Das von den Ontologien bereitgestellte Vokabular dient als ein stabiles, konzeptuelles und von Datenbank-Schemata unabhängiges Interface für Datenbanken. Die Ontologiesprachen sind ausdrucksstark genug, um die Komplexität von Abfragen in entscheidungsunterstützenden Anwendungen zu verarbeiten [BCB05]. Suchanfragen werden an ein mediatorbasiertes Schema in Form einer Ontologie, die den Wissensbereich eines Systems beschreibt, gestellt. Wenn Ontologien als ein Query-Modell verwendet werden, dann besteht für die Nutzer die Möglichkeit, Suchanfragen mit den Begriffen der globalen Ontologie wie zum Beispiel SIMS zu stellen, ohne Berücksichtigung von Begrifflichkeiten oder der Sprache an sich, die die verteilten Informationsquellen enthalten. Dies ist der Grund dafür, dass Suchanfragen in einer „high-level“-Sprache wie etwa Loom oder einer SQL-ähnlichen Sprache geschrieben werden können. Diese Queries enthalten keine Informationen, die auf die Quellen hinweisen, die entweder relevant für ihre Ausführung sind oder die eine Quellensuche ermöglichen. Das System reformuliert die Suchabfragen in Unterabfragen entsprechend für jede Informationsquelle.

Query-Modell

Verifikation

Jede Informationsquelle ist eine Instanz eines Ontologiesystems. Eine Ontologie kann Auskunft über Ressourcen geben, etwa darüber wie oft und durch wen sie

aktualisiert worden ist. Solche Informationen dienen der Zugriffsüberprüfung und Zugriffssteuerung auf Instanzen von Ontologien.

Der Austausch von semantikerhaltenden Informationen erfordert eine überprüfbarer Methode, die die logisch äquivalenten Konzepte in den unterschiedlichen Ontologien erkennt und diese dann in die Zielontologie überführt.

Die Korrektheit und die Überprüfung dieser Transformation ist selbst mit der ausdrucksstärksten Ontologiesprache problematisch [CGL01], weil das Ergebnis des *Mappings* der unterschiedlichen Ontologien stark von der Übereinstimmung der Konzepte abhängt. Die Korrektheit einer Entscheidung bei der Suche nach einer Übereinstimmung, etwa die Frage, ob das Ergebnis einer lokalen Suchanfrage eine Untergruppe von dem Ergebnis einer globalen Suchanfrage ist (in diesem Fall spricht man vom sogenannten Problem des *Query-Containments* [CGL98]), ist Aufgabe der Verifikation. Mit Ontologien können beispielsweise Software-Agenten überprüfen, ob es sich bei den erhaltenen Informationen um die gewünschten Informationen handelt und anschließend mit Inferenzregeln die Kohärenz der Ontologie überprüfen [HEL01]. Die Qualität einer Verifikation hängt in hohem Maße von der Vollständigkeit des Vokabulars der beteiligten Ontologien und von ihrem Formalisierungsgrad ab.

Ein Framework für die Informationsnutzung

Gesucht wird ein einheitliches und ausdrucksstarkes Framework, das mithilfe von entsprechenden Tools die Voraussetzung für die Abfrage, die Interpretation und die Integration der verschiedenen Informationsressourcen wie HTML-, XML-, RDF-, RDFS- und andere Dateien schafft. Die meisten Framework-Architekturen basieren auf dem Hybridansatz, weil mit dem einheitlichen und standardisierten Vokabular die Suchanfragen, ihre Reformulierungen, die Übersetzungen zwischen den Ontologien und die Verifikation wesentlich vereinfacht werden. Eine auf dem Hybridansatz basierende Infrastruktur beinhaltet folgende drei Schichten:

- Die unterste Ebene hält die Daten, Metadaten und Ressourcen bereit, die mit HTML, XML oder RDF repräsentiert sein können.
- Die mittlere Ebene stellt die *Middleware-Schicht*, die bestimmte Dienste bereitstellt [KSC05], die
 - Mapping-Regeln zwischen XML und RDF und anderen Sprachen ermöglichen,
 - die Konformität dieser *Mappings* überprüfen,
 - die Reformulierungen der Suchanfragesprachen für die verschiedenen Informationsquellen bereitstellen und

- diese Suchanfragen miteinander kombinieren.
- Die oberste Ebene stellt für diese Dienste die wiederverwendbaren, standardisierten und stabilen Ontologien und das globale Vokabular zur Verfügung.

Ein Framework für die ontologiebasierte Informationsnutzung sollte neben dieser Infrastruktur zwei Mechanismen bereitstellen:

- Methoden zur Gewinnung von inhaltsbasierten Informationen aus verteilten Ressourcen (*Information Retrieval*) sowie
- Methoden zur Übersetzung der Konzeptualisierungen verschiedener Ressourcen.

Für die Informationsgewinnung kann man verschiedene *Reasoning* Systeme verwenden, mit denen Subsumptions-Relationen zwischen verschiedenen Ontologien automatisch berechnet werden können. Dafür wird allerdings die Existenz des gemeinsam genutzten Vokabulars vorausgesetzt. Das *Reasoning* quer über die Ontologien benötigt also ein gemeinsam zu nutzendes Basisvokabular (Reduktion auf die Syntax). Die Beschreibung der Konzepte in beiden Ontologien erfolgt mit Begriffen der logischen Ausdrücke über dem gemeinsam genutzten Vokabular (Reduktion auf die Logik).

Reasoning

Die Methoden zur Übersetzung und die Methoden zur Informationsgewinnung sind stark miteinander verflochten, da zum einen für die Suche nach relevanten Informationen eine Übersetzung der vom Benutzer verwendeten Terminologie notwendig ist und zum anderen die gefundenen Informationen ebenfalls in die vom Benutzer verwendeten Terminologie zurückübersetzt werden müssen. Die Übersetzungsmethoden ermitteln und lösen dabei die Konflikte zwischen den Definitionen der Objekt-Klassen, die im Widerspruch stehen

Der Übersetzungsprozess und Ontologie-Alignment

Das ontologische Übersetzungsproblem tritt dann auf, wenn Ontologien oder Teile davon mit entsprechenden Tools oder Sprachen wiederverwendet werden sollen, die sich aber von den verfügbaren Ontologien unterscheiden [GRU93]. Die verschiedenen Sprachen und Tools sind unterschiedlich in ihrem Grad der Formalität und ihrer Ausdrucksmächtigkeit. Zudem haben sich auch viele verschiedene Ontologien entwickelt, die entweder in den klassischen Ontologiesprachen wie CycL, Flogic, KIF, LOOM oder Ontolingua implementiert worden sind oder in den Markup-Sprachen für das Semantik Web, zu denen DAML+OIL, OWL, RDF, RDFS, XOL etc. zu zählen sind [CGP05].

Um die Interoperabilität zwischen den semantischen (Web-)Applikationen zu gewährleisten, wird eine Mediation zwischen diesen verschiedenen Wissens-Repräsentationen benötigt. Auch für das *Querying* und für die Datenintegration ist ein

Mappingprozess, der die Beziehungen zwischen den beteiligten Ontologien beschreibt oder ein *Merge*, der eine neue Ontologie aus den Quellressourcen erstellt, notwendig. Erst die Existenz eines gemeinsam genutzten Vokabulars ermöglicht die Übersetzung zwischen verschiedenen Informationen auf einer semantischen Basis. Man kann nach dem Kriterium der zeitlichen Ordnung beim Mapping drei Phasen unterscheiden [SEK05], wobei nicht alle Phasen in jedem *Mapping-Tool* notwendig und manche Phasen nur optional sind:

1. Ontologien importieren

Die in verschiedenen Sprachen spezifizierten Ontologien werden in das *Mapping-Tool* importiert, das dann das *Mapping* spezifiziert.

2. Ähnlichkeiten finden

Viele Systeme benutzen einen *Match-Operator*, um semi-automatisch Ähnlichkeiten zwischen den Schemata oder Ontologien zu finden. Es werden Merkmale bzw. Attribute von Entitäten gesucht und der Suchraum kann eingeschränkt werden, sodass nur sinnvolle Entitäten für einen Vergleich zugelassen werden [SES06].

3. Mapping/Merge spezifizieren

Nach dem Auffinden der Ähnlichkeiten zwischen Ontologien müssen diese spezifiziert werden, dies geschieht in erster Linie manuell, eventuell auch mit einem Hilfstoß wie PROMPT. PROMPT ist ein Tool für den interaktiven *Merging*-Prozess, indem eine kohärente Ontologie erzeugt wird, die die Versionen der Ursprungsontologien enthält. Meistens finden noch iterative Rücksprünge zur vorherigen Phase statt, weil ein Entitätenpaar normalerweise mehrere Ähnlichkeitswerte besitzt. Tools können im Gegensatz zu manuellen Verfahren präzisere Ähnlichkeitsmessungen durch eine simple Durchschnittsbildung oder durch komplexe Aggregierungsfunktionen mit Gewichten für jede einzelne Ähnlichkeit durchführen.

Ontologie-Aligning

Ontologie-Aligning

Verändert man die originalen Ontologien nicht, impliziert dies, dass nur ein Teil der Ontologie-Integration durchgeführt werden kann, weil große Differenzen die Adaption der Ontologien erfordern. Wenn zwei Ontologien verbunden werden durch ein *Ontologie-Aligning*, dann bleiben die ursprünglichen Ontologien mit einer zusätzlichen Anzahl von Links erhalten, eine Sicht, die auch als *bridge ontology* bezeichnet wird und die eine gegenseitige Wiederverwendung der Informationen in den Ontologien erlaubt [SEK05]. Je höher der Grad der Wiederverwendbarkeit des gemeinsam genutzten Vokabulars und der Ontologien ist, umso mehr ist eine semantikerverhaltende Integration von Informationen möglich.

Ontologie-Engineering

Klassische Entwicklungsmethoden für wissensbasierte Systeme sind meistens zentralisiert in ihrem Wissenssystem selbst. Im Gegensatz dazu versucht das Ontologie-Engineering ein unvollständiges und sich laufend veränderndes System zu strukturieren. Wünschenswertes Ziel ist es, dass Agenten, Webservices und ontologiebasierte Peer-to-Peer-Systeme mit Ontologien arbeiten, die alle auf ein gemeinsames Vokabular zurückgreifen. Um dieses Ziel zu erreichen, sollte eine strukturierte Entwicklungsmethode für Ontologien die Übereinstimmung der zu entwickelnden Ontologie sowohl innerhalb der Teammitglieder als auch zwischen den Teams berücksichtigen. Ebenso muss die Weiterentwicklung der existierenden Ontologien durch andere möglich sein, wie auch die entwickelte Ontologie in anderen Ontologien und Applikationen wiederverwendbar sein sollte. Darum ist eine minimale ontologische Vereinbarung unerlässlich [GRU93], in der festgehalten wird, dass soweit wie möglich das standardisierte vorhandene Vokabular für die Beschreibung der Ontologien zu benutzen ist.

Die bekanntesten Entwicklungsmethoden, wie beispielsweise die von Uschold und Gruninger [USG96], beginnen alle mit der Identifikation des Zwecks und des Einsatzbereichs sowie dem Aufbau der Ontologie durch Erfassung und Klassifizierung von Konzepten, formaler Kodierung und Integration von existierenden Ontologien mit iterierender Evaluation und Dokumentation. Nach der Erfassung und Strukturierung des Wissenskontexts schlägt Uschold die Codierung in eine formale Sprache vor. Um eine minimale Kodierungsabweichung zu erreichen, die die Konzeptualisierung auf dem Wissenslevel unabhängig von der symbolischen Kodierung erfordert [GRU93], wird in dem „ME-THONTOLOGY“-Framework [FGJ97], das sowohl eine Definition und Standardisierung eines Ontologie-Lebenszyklus als auch die Methoden und Techniken für ihre Entwicklungsarbeit bereitstellt, noch eine Schicht (intermediate representations) eingefügt.

Dieses Vorgehen ermöglicht es dem Ontologie-Entwickler, sich ganz auf die Konzeptualisierungen des Problembereichs zu konzentrieren, ohne Rücksicht auf die Implementierungssprache nehmen zu müssen. Die Implementierung wird anschließend automatisiert mit einem Übersetzer in die verschiedenen Sprachen durchgeführt. So können mit kompetenten Fragen (*competency questions*) die Anforderungen für die ontologiebasierte Anwendung definiert und mit prototypischen Instanzen für eine Testmenge die Evaluierung auf dem Wissenslevel durchgeführt und die Benutzbarkeit einer Ontologie verbessert werden [BFG98].

kompetente Fragen

Selbsttestaufgabe 7.6:

Nennen Sie die drei Rollen von Ontologien bei der Informationsintegration.

Selbsttestaufgabe 7.7:

Warum kann die Überprüfung der Überführung von Ontologien in eine Zielontologie problematisch sein?

5 Fazit

Ontologien sind im World-Wide Web inzwischen alltäglich geworden. Die Ontologien im Web reichen von umfangreichen Taxonomien, die die Webseiten kategorisieren, als Beispiel sei hier Yahoo genannt, dem folgen online frei verfügbare Nachschlagewerke wie Wikipedia bis hin zu Produktkategorien für den Verkauf, wie es Amazon.com anbietet [NMG01]. Es haben sich Initiativen gebildet wie z. B. die *Standard Upper Ontology Working Group* (SUO), die einen Standard für Upper-Ontologien entwickeln, um die Dateninteroperabilität, Informationssuche und Retrieval, automatisierte Schlussfolgerungen durch Inferenzen und die natürliche Sprachverarbeitung in Programmen zu ermöglichen und zu unterstützen. Die Standardisierung der Ontologieinhalte, wie sie Upperlevel-Ontologien bereitstellen, ist enorm wichtig für die Produktion von benutzbaren Ontologien hinsichtlich der Integrität und Konsistenz, weil sie wiederverwendet und innerhalb von Applikationen eingesetzt werden können.

Damit Ontologien die ihnen zugesetzten Rollen im semantischen Web erfüllen können, sind die vom WWW-Consortium (W3C) empfohlenen Datenmodelle RDF, RDFS und OWL weit verbreitet eingesetzt worden. Mit dem standardisierten Vokabular von RDFS/OWL konnte man das Problem der semantischen Heterogenität reduzieren. Es gibt bereits einige praktische Anwendungen, die mit Einsatz von verschiedenen Techniken die Modellierung von Ontologien und die Wissensintegration erlauben, wie beispielsweise Protégé oder OntoStudio. Diese Anwendungen haben sich allerdings noch nicht zu einer Standard-Applikation im semantischen Web durchsetzen können. Die Evolution der ontologiebasierten Anwendungen, die Modellierung von Ontologien, die Verfahren zum *Alignment* zwischen verschiedenen Ontologien wie auch die Reduzierung des *Overheads* für die Spezifikation von Metadaten sind nach wie vor die Schwerpunkte der heutigen Forschung [EHH06].

6 Literaturverzeichnis

- [KUR93] Raimond Kurzweil. Die Suche nach dem Wissen. In: Das Zeitalter der Künstlichen Intelligenz. Carl Hanser Verlag 1993.
- [CUO00] Cui, Z. and O'Brien, P.: Domain Ontology Management Environment. In : Proceedings of the 33rd Hawaii International Conference on System Sciences. 2000.
- [BCB05] Buccella, A., Cechich, A. and Brisaboa, N. R.: Ontology-Based Data Integration Methods: A Framework for Comparison. 2005.
- [SOW03] Sowa, J.F. Ontology, Definition and Scope, last modified 2003:
<http://www.jfsowa.com/ontology/index.htm>
- [GRU93] Gruber, T.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. 1993.
- [VIS02] Visser, U. and Schlieder, Ch.: Modelling Real Estate Transactions: The Potential Role of Ontologies. Ashgate. 2002.
- [BHS05] Bloehdorn, St., Haase, P., Sure, Y. and Voelker, J. (Universität Karlsruhe): Report on the integration of ML, HLT and OM. Karlsruhe. 2005.
- [STS04] Studer, R.(Hrsg.) and Staab, S. (Hrsg.) *Handbook on Ontologies – International Handbooks on Information Systems.* Springer Verlag. Heidelberg. 2004.
- [MET03] What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology and a meta-model?
In: <http://www.welchco.com/02/14/60/03/01/1501.HTM>.
- [SUR04] Sure, Y.: Ontologien. Institut AIFB Karlsruhe: KompetenzNetzwerk Wissensmanagement. 26.05.2004.
- [WHO69] Whorf, B. L.: Sprache - Denken – Wirklichkeit. Beiträge zur Metalinguistik und Sprachphilosophie. Hrsg. Von Peter Krausser. Rowohlt Verlag. Hamburg 1969.
- [DJK04] Dostal, W., Jeckle, M., Kriechbaum, W.: Semantik und Web Services. Vokabulare und Ontologien. In: Java-Spektrum. Ausgabe 3/2004. S. 51 - 54.
- [OGR23] Ogden, Ch. K. and A. Richards, I. A.: The Meaning of Meaning. London. 1923. In: Online-Artikel XML & Web Services Magazin. Ausgabe 4.
- [TRA96] Trabant, J.: Elemente der Semiotik. Tübingen. 1996 (Stuttgart: Beck'sche 1976).

- [JAN03] Janowicz, K.: Fahrt in die Havel DAML+OIL - Über den praktischen Nutzen von Ontologien. XML & Web Services Magazin. 2003. S. 34-38.
- [DJK042] Dostal, W., Jeckle, M., Kriechbaum, W.: Semantik und Web Services. Beschreibung von Semantik. In: Java Spektrum. Ausgabe 2/2004. S. 45 – 49.
- [USJ99] Uschold, M., Jasper, R.: A Framework for Understanding and Classifying Ontology Applications. In: Proceedings of the IJCAI99 Workshop on Ontologies and Problem-Solving Methods (KRR5). Stockholm. 1999.
- [FFR97] Farquhar, A., Fikes, R., Rice, J.: The Ontolingua Server: A Tool for Collaborative Ontology Construction. Stanford University. Proceeding AAAI'97/IAAI'97. Proceedings of the fourteenth national conference on artificial intelligence and ninth conference on Innovative applications of artificial intelligence. 1997.
- [LOE03] Lönneker, B.: Konzeptframes und Relationen: Extraktion, Annotation und Analyse französischer Corpora aus dem World Wide Web. Hamburg. 2003.
- [DRS06] Drummond, N. and Shearer, R.: The Open World Assumption. Presentation. The University of Manchester. 2006.
- [DIE94] Dieter, F.: Sinn und Unsinn formaler Spezifikationssprachen für wissensbasierte Systeme. Künstl. Intell. (KI) 8 (1994) H. 4 S. 26-34.
- [GUA98] Guarino, N.: Formal Ontology and Informationssystems. In: Proceedings of the 1st International Conference. Trento. 1998.
- [OWL04] OWL Web Ontology Language Overview, <http://www.w3.org/TR/2004/REC-owl-features-20040210/>. 2004.
- [BKD00] Broekstra, J., Klein, M., Decker, St., Fensel, D. and Horrocks, I.: Adding formal semantics to the Web: Building on top of RDF Schema. In: Proceedings of the Workshop ECDL 2000 Workshop on the Semantic Web. 2000.
- [GIR02] Gil, Y., Ratnakar, V.: A Comparison of (Semantic) Markup Languages. In: Proceedings of the 15th International FLAIRS Conference. Special Track on Semantic Web. USA. 2002.
- [COG00] Corcho, O., Gomez-Pérez, A.: A Roadmap to Ontology Specification Languages. 2000. In: Journal IEEE Intelligent Systems. Volume 17/1. USA
- [USG96] Uschold, M., Gruninger, M.: ONTOLOGIES: Principles, Methods and Applications. AIAI-TR-191. In: Knowledge Engineering Review. Volume 11/2. 1996.
- [GRU05] Grüninger, M.: Model-theoretic Approaches to Semantic Integration (Extended Abstract). In: Dagstuhl Seminar 05371: Principles and Practices of Semantic Web Reasoning. 2005.

- [BRY05] Bry, F.: Reasoning ist der Motor des Semantic Web. In: www.semantic-web.at. 2005.
- [SEK05] SEKT: Semantically Enabled Knowledge Technologies: D4.2.2 State-of-the-art survey on Ontology Merging and Aligning V2. Report. 2005.
- [CGL01] Calvanese, D., de Giacomo, G., Lenzerini, M.: A Framework for Ontology Integration. Università di Roma. In: Proc. of 2001 Int. Semantic Web Working Symposium (SWWS 2001). Pages 303-316.
- [CGL98] Calvanese, D., de Giacomo, G., Lenzerini, M.: On the Decidability of Query Containment under Constraints. In: Proc. of the Principles on Database Systems. PODS. 1998. Pages 149 – 158.
- [HEL01] Helic, D.: Aspects of Semantic Data Modeling. In: Hypermedia Systems. Graz University of Technology. 2001. S. 54.
- [KSC05] Koffina, I., Serfiotis, G., Christophides, V., Tannen, V., Deutsch, A.: Integration XML Data-Sources using RDF/S. Zitiert in: Dagstuhl Seminar 2005.
- [SvH05] Stuckenschmidt, H., van Harmelen, F.: Information Sharing on the Semantic Web. Springer-Verlag. Heidelberg. 2005.
- [CGP05] Corcho, O., Gómez-Pérez, A.: A Layered Model for Building Ontology Translation System. International Journal on Semantic Web and Information Systems (IJSWIS). . Spanien. 2005.
- [SES06] Sure, Y., Ehrig, M., Studer, R.: Automatische Wissensintegration mit Ontologien. Institut AlFB Karlsruhe. 2006.
- [BFG98] Blázquez, M., Fernández, M., García-Pinar, J. M., Gómez-Pérez, A.: Building Ontologies at the Knowledge level using the Ontology Design Environment. KAW. Canada. 1998.
- [FGP99] Fernández-López, M., Gómez-Pérez, A., Pazos Sierra, J.: Building a Chemical Ontology Using Methontology and the Ontology Desing Environment. In: IEEE Intelligent Systems. Volume: 14,1. Pages: 37-46 Madrid 1999.
- [FGJ97] Fernández-López, M., Gómez-Pérez, A., Juristo, N.: METHONTOLOGY: From Ontological Art towards Ontological engineering. In: AAAI. Technical Report. 1997.
- [NMG01] Noy, N. F., McGuiness, D. L.: Ontology development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880. Standford University. 2001.
- [EHH06] Ehrig, M., Haase, P., Hartmann, J., Oberle, D., Tempich, C., Studer, R., Sure, Y.: Ontology Management und Semantische Portale. In: Studer, R.: Wissensmanagement. Jahresbericht. Universität Karlsruhe. 2006.

- [SHA06] Sure, Y., Hitzler, P., Ankolekar, A.: Intelligente Systeme im WWW: Semantic Web:RDF/S und Ontologien. Institut AIFB, Universität Karlsruhe. 2006. <https://docplayer.org/3261414-Intelligente-systeme-im-world-wide-web-web-ontology-language-owl-rdf-syntax.html>

7 Abbildungsverzeichnis

| | |
|--|----|
| Abbildung 3.1 Ontologien und ihre Verwandten [SHA06]..... | 7 |
| Abbildung 3.2 Auszug aus einer Insektenklassifikation | 8 |
| Abbildung 3.3 ER-Modell anhand einer Vorlesung | 9 |
| Abbildung 3.4 FOAF Beispiel anhand einer Co-Autorenbeziehung (http://ezdl.de) | 10 |

8 Lösungen der Selbsttestaufgaben

Selbsttestaufgabe 7.1

Was wird unter der semantischen Heterogenität verstanden?

Die semantische Heterogenität die Unterschiede in Bedeutung, Interpretation und Art der Nutzung, speziell von Informationen. Es ist damit das größte Problem für die automatische Wissensverarbeitung.

Selbsttestaufgabe 7.2

Nennen und erklären Sie die drei Konzepte der semantischen Heterogenität.

Bei **semantisch äquivalenten Konzepten** können sich etwa verschiedene Begriffe auf ein und dasselbe Konzept beziehen.

Bei **semantisch unabhängigen Konzepten** kann es zu Verständnisproblemen kommen, wenn der gleiche Begriff für unterschiedliche Konzepte verwendet wird.

Bei **semantisch abhängigen Konzepten** wird die Heterogenität anhand von Generalisierung und Spezifizierung beschrieben.

Selbsttestaufgabe 7.3

Erklären Sie den Begriff der Taxonomie.

Mit einer Taxonomie werden Mengen von Konzepten von kontrollierten Vokabularen eines Themengebietes definiert und in eine hierarchische Beziehung gesetzt, um eine möglichst präzise Beschreibung und Repräsentation eines Themengebiets zu erhalten. Eine Taxonomie ist also ein Klassifikationsschema.

Selbsttestaufgabe 7.4

In welche vier Bereiche lässt sich die Formalität einer Sprache, die eine Ontologie repräsentiert, einteilen? Benennen und erläutern Sie sie.

- Total-informell

Eine Ontologie wird einfach in natürlicher Sprache beschrieben, viele Glossare passen in diese Kategorie.

- Semi-informell

Die Ontologie wird in einer strukturierten Form mittels einer natürlichen Sprache dargestellt, die Mehrdeutigkeiten reduziert. Dies kann z. B. eine Taxonomie, also ein Wortschatz sein, der in einer Generalisierungs-Spezialisierungs-Hierarchie strukturiert ist.

- Semi-formal

Eine semi-formale Sprache ist eine künstlich formal definierte Sprache, die die Beschreibung der Begriffe sowohl informell in einer natürlichen Sprache als auch formal in einer für Computer interpretierbaren Wissensrepräsentationssprache unterstützt [FFR97].

- Rigoros formal

Eine auf der Prädikatenlogik basierende formale Sprache treibt den Standardisierungsprozess voran und ermöglicht die Entwicklung von Schlussfolgerungskalkülen, welche gültige Eigenschaften von Mengen von Ausdrücken ableiten, sodass Fragen der Vollständigkeit und Korrektheit von formalen Spezifikationen überprüft werden können.

Selbsttestaufgabe 7.5

Nennen Sie zwei Anwendungsbeispiele für Ontologien.

- **Kommunikation:** Ontologien reduzieren die semantischen und terminologischen Begriffsverwirrungen und ermöglichen dadurch ein gemeinsames Verständnis
- **Interoperabilität:** Nutzung von Ontologien für die Definition einer gemeinsam genutzten terminologische Semantik

Selbsttestaufgabe 7.6

Nennen Sie die drei Rollen von Ontologien bei der Informationsintegration.

- Ontologien werden hauptsächlich zur Repräsentation und Visualisierung von zum Teil unterschiedlich strukturierten Informationen eingesetzt.
- Ontologien können als ein Anfrage-Modell eingesetzt werden, das die Suche und das Browsing von Informationen vereinfacht, indem die Struktur der Ontologie selbst genutzt wird.
- Ontologien werden auch als Verifikationsmodell genutzt, denn durch die Spezifikation wird die Softwareentwicklung mit Ontologien konsistent.

Selbsttestaufgabe 7.7

Warum kann die Überprüfung der Überführung von Ontologien in eine Zielontologie problematisch sein?

Die Korrektheit und die Überprüfung einer Transformation sind selbst mit der ausdrucksstärksten Ontologiesprache problematisch, weil das Ergebnis des Mappings der unterschiedlichen Ontologien stark von der Übereinstimmung der Konzepte abhängt.

000 000 000 (00/23)

00000-0-00-S1

Alle Rechte vorbehalten
© 2023 FernUniversität in Hagen
Fakultät für Mathematik und Informatik