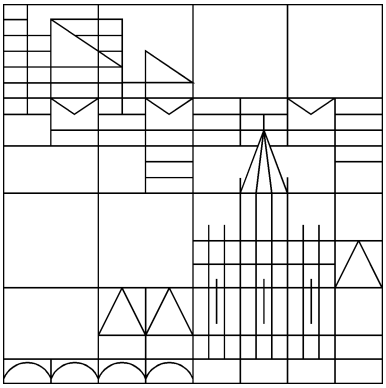
„XML-Technologien - Integration von externen, vernetzten Daten in ein offenes Hypertextsystem (KHS)“

Diplomarbeit

im Studiengang Informationswissenschaft der Universität Konstanz



SS 1999

1. Gutachter Prof. Dr. Rainer Kuhlen

2. Gutachter Prof. Dr. Wolfgang Pree

Uwe Speck

Konstanz

Kurzreferat:

Das Ziel der vorliegenden Arbeit ist die Integration von externen, vernetzten Daten (Hypertexten), in die Datenbasis des *Konstanzer Hypertext-Systems* (KHS) der FG Informationswissenschaft der UNI Konstanz. Das KHS, ein objektorientiertes WEB-basiertes Hypertextsystem, wird seit Jahren im Bereich elektronischer Kommunikationsforen eingesetzt. Im untersuchten Ansatz erfolgt die *Codierung* der zugrunde liegenden Daten mittels der *Extensible Markup Language* (XML), die ein "web-optimiertes SGML" darstellt. Als *Inhalte* kommen beispielsweise *lexikografische* Daten in Frage. Jedoch sind auch weitere Gegenstandsbereiche, wie z. B. Kommunikationsforen oder Online-Publikationen/ Präsentationen denkbar. Zunächst wird ein kurzer Überblick über die Kernkomponenten von XML (Inhalt/Struktur-, Stil-, Link-) gegeben, mit Schwerpunkt auf den Hypertext-Aspekten (XLink, XPointer). Ferner werden Programm-Module zur Verarbeitung von XML-Dokumenten, sogenannte XML-Prozessoren, anhand ihrer derzeit prominentesten *Application Programming Interfaces* (API) - *Simple API for XML* (SAX) und *Document Object Model* (DOM) - detailliert erläutert. Die theoretischen Ausführungen der Arbeit münden in der prototypischen Realisierung eines Programm-Modules (Projektname "XML4KHS") das die Integration von vernetzten Testdaten eines elektronischen Kommunikationsforums erfolgreich durchführt. Dieses Modul, das auch als "Import-Filter" bezeichnet wird, besitzt dabei eine *generische* Struktur, die sich an verschiedene Dokumenttypen anpassen läßt.

abstract:

The aim of this work is the integration of external, connected Data, means Hypertexts, into the database of the "Konstanzer Hypertext-System (KHS)" of the Information science division of the university of constance. The KHS is an objectoriented WEB-based Hypertextsystem, which is used since years in term of electronic forum. The encoding of the data is done by use of the Extensible Markup Language XML, which is a "web-optimized" SGML. The content may be lexicografical data, but other topics of use as electronical communication forums or online publications/presentations are also possible.First a short overview about the basic components of XML (content/structure, Style, link) is done, with emphasis on the hypertext aspects (Xlink, XPointer). Further, program-modules for processing of XML documents are explained in detail. The most popular standard Application programming interfaces (APIs), Simple API for XML (SAX) and Document Object Model (DOM) are used as examples. The theoretical Work guides to a prototypic realisation of a program-module (project name "XML4KHS") which delivers the integration of connected test data of an electronic communication forum with success. This module, wich is called an "import-filter", has got a generic structre, which can be adapted to different document types.

„Man freut sich, wenn man zu antworten weiß; wie gut ist das richtige Wort zur rechten Zeit!“

Die Bibel: „Sprichwörter Salomos“ Kapitel 15, Vers 23

Inhaltsverzeichnis

Inhaltsverzeichnis

1 Einleitung: Motivation und Zielsetzung

1.1 Motivation

Zahlreiche Literaturquellen weisen darauf hin, daß die Metasprache **Extensible Markup Language** (XML) - ein SGML-Derivat, das speziell für das Anwendungsgebiet des *World Wide Web* (WWW) optimiert wurde - einen hohen Stellenwert im professionellen Online-Bereich einnehmen wird.

> "Wer allerdings Daten professionell aufbereitet beziehungsweise generiert, wird an XML [als Alternative zu HTML] mittelfristig nicht vorbeikommen" [Behme 1999, S. 36]

> "Wir stehen kurz vor einer Datenexplosion - ausgelöst durch XML" [Goldfarb, Prescod 1999, S. 20].

Ein weiteres Indiz für die zunehmende Bedeutung von XML ist die Tatsache, daß namhafte Software-Anbieter wie Netscape, Microsoft, IBM, Oracle, Corel diese Metasprache in ihre Produkte integrieren. Es zeichnet sich folgendes Bild ab:

- im Bereich des *WWW* wird die Relevanz von XML teilweise so hoch eingeschätzt, daß sogar von der kommenden "Lingua Franca" des Internet/Intranet die Rede ist [Macherius 1997].
- bei *Office-Anwendungen* stellt XML eine Alternative zu klassischen, offenen Dokumentenformaten (Rtf, Pdf, PS) dar, insbesondere da XML, bei vergleichbaren Formatierungs-Möglichkeiten, die *Trennung* von Inhalt und Darstellung erlaubt.
- bei *Dokument–Management Systemen* (DMS) bietet sich die Kombination von XML und OO-DBMS (z. B. Poet) an [Behme, Mintert 1998, S. 21].

Aus **informationswissenschaftlicher Sicht** besitzt eine XML-Applikation, die man auch als "customized HTML" bezeichnen könnte, einen wesentlichen Vorteil gegenüber HTML: die Möglichkeit, den spezifisch definierten Elementtypen eine *Semantik* in bezug auf den Element-Inhalt (<ADRESSE>, <TELEFONNR>, <ISBN>, <PREIS> etc.) zu geben. Da es sich dabei um differenzierte *Meta-Information* handelt, die unter anderem erweiterte Retrieval-Möglichkeiten bietet und die KI-gestützte Analyse verbessert, kann der

Gebrauchswert der Dokumente gesteigert werden. Dadurch bietet eine XML-Anwendung einen *informationellen Mehrwert* [Kuhlen 1996, S. 80 ff.] gegenüber HTML, welches lediglich Struktur-Information zur Verfügung stellt. Hinzu kommt, daß eine XML-Applikation weitere Mehrwerte gegenüber HTML besitzt. Hierzu gehören die erweiterten Link-Möglichkeiten (insgesamt vier Typen) und neue Formatierungskonzepte (u.a. medien-angepaßte Formatierung und mehrere Formatvorlagen für ein Dokument). Aus diesen Gründen liegt bei spezifischen Dokumenttypen die Anwendung einer individuellen XML Applikation nahe[1].

Dieser Ansatz wurde beispielsweise vom Institut für Deutsche Sprache (IDS) anhand lexikografischer Daten im Rahmen des Projektes *Lexikalisch-lexikologisches, korpus-gestütztes Such- und Informationssystem* (LEKSIS) [Toussi 1999] verfolgt. Hier ist man im Begriff, **hochgradig vernetzte**, lexikografische **Daten**, zu denen nicht nur Lexikon-Artikel, sondern auch Autoren-Informationen, Belegstellen usw. gehören, mittels XML in eine Hypertext-Form zu bringen. Daneben existieren in Deutschland weitere, vergleichbare Projekte, wie zum Beispiel das *Deutsche Rechtswörterbuch* (DRW) der Akademie der Wissenschaften oder der *Duden* des Brockhaus Verlages.

Das **Konstanzer Hypertext System** (KHS) der Fachgruppe Informationswissenschaft der Universität Konstanz wird seit Jahren unter anderem im Bereich elektronischer Kommunikationsforen eingesetzt. Es bietet eine WWW-Schnittstelle, die gemäß dem HTTP Protokoll die Kommunikation mit herkömmlichen Browser-Clients unter Verwendung von HTML, einer SGML-Applikation, herstellt. Dieses System, das gemäß der Definition von Kuhlen [1996, S. 432ff.] als *offenes* Hypertext-System bezeichnet werden darf, basiert technologisch auf einem objektorientierten DBMS mit angekoppeltem WEB-Server. Es wird derzeit von der Programmiersprache Smalltalk nach Java portiert.
Aufgrund der aufgeführten informationellen Mehrwerte, die eine spezifische XML-Applikation gegenüber HTML aufweist, ist es sinnvoll, XML-Technologien in das KHS zu **integrieren**. Die vorliegende Arbeit soll hierzu einen Beitrag leisten.

[1] Zu derartigen Dokumenttypen gehören beispielsweise: lexikografische Artikel, mathematische Dokumente und EDI-Dokumente.

1.2 Zielsetzung

Konkretes Ziel dieser Arbeit ist die Integration externer, vernetzter XML-Dokumente, also Hypertexte, in das KHS. Als Inhalte dieser Hypertexte kommen dabei unter anderem, wie am Beispiel der Projekte „LEKSIS“ und „DRW“ bereits angesprochen, *lexikografische* Daten in Frage. Jedoch sind auch weitere Gegenstandsbereiche, wie z. B. Kommunikationsforen oder Online-Publikationen/ Präsentationen denkbar.

Unter **„Integration“** wird im vorliegenden Kontext primär die *Integration von Dokumenten in die Datenbasis des KHS* verstanden – ein Vorgang, der auch als *Import* bezeichnet wird. Weitere integrative Maßnahmen, wie der naheliegende *Export* von Daten oder die *Visualisierung* von XML-Dokumenten mittels der von XML zur Verfügung gestellten Stil-Komponenten können in Anbetracht des zur Verfügung stehenden Zeitrahmens nur rudimentär behandelt werden.

Es werden folgende Schwerpunkte gelegt:

- der erste Schwerpunkt der vorliegenden Arbeit besteht in der Darstellung der notwendigen XML-Technologien. Wichtig erscheinen hierfür die Hypertext-Aspekte von XML („Link-Technologien“) sowie Konzepte zur Definition von XML-Applikationen wie Dokument-Typ-Definitionen (DTDs) oder XML-Schema-Sprachen. In Anbetracht künftiger integrativer Maßnahmen wie der Visualisierung von Hypertext-Daten mittels eines XML-Frontends werden auch mittelbar relevante Technologien, wie beispielsweise die XML-Stilkomponenten kurz angesprochen,
- im Rahmen des zweiten Schwerpunktes wird Information zur Verarbeitung von XML-Dokumenten, die von sogenannten XML-Prozessoren geleistet wird, erarbeitet und aufbereitet. Hierbei werden zunächst eingesetzte Standard-APIs2 erläutert. Danach werden verschiedene Standard-Produkte der XML-Verarbeitung evaluiert, um ein passendes Werkzeug für die Erstellung des prototypischen Import-Filters zu finden,

[2] API - Application Programming Interface

- der dritte Schwerpunkt behandelt die konkrete prototypische Realisierung eines Importfilters für XML-codierte Hypertexte, Arbeitstitel „XML4KHS“. In diesem Zusammenhang wird auch das Konstanzer Hypertext-System kurz beschrieben – in dem für das Verständnis des Filters notwendigen Maße.

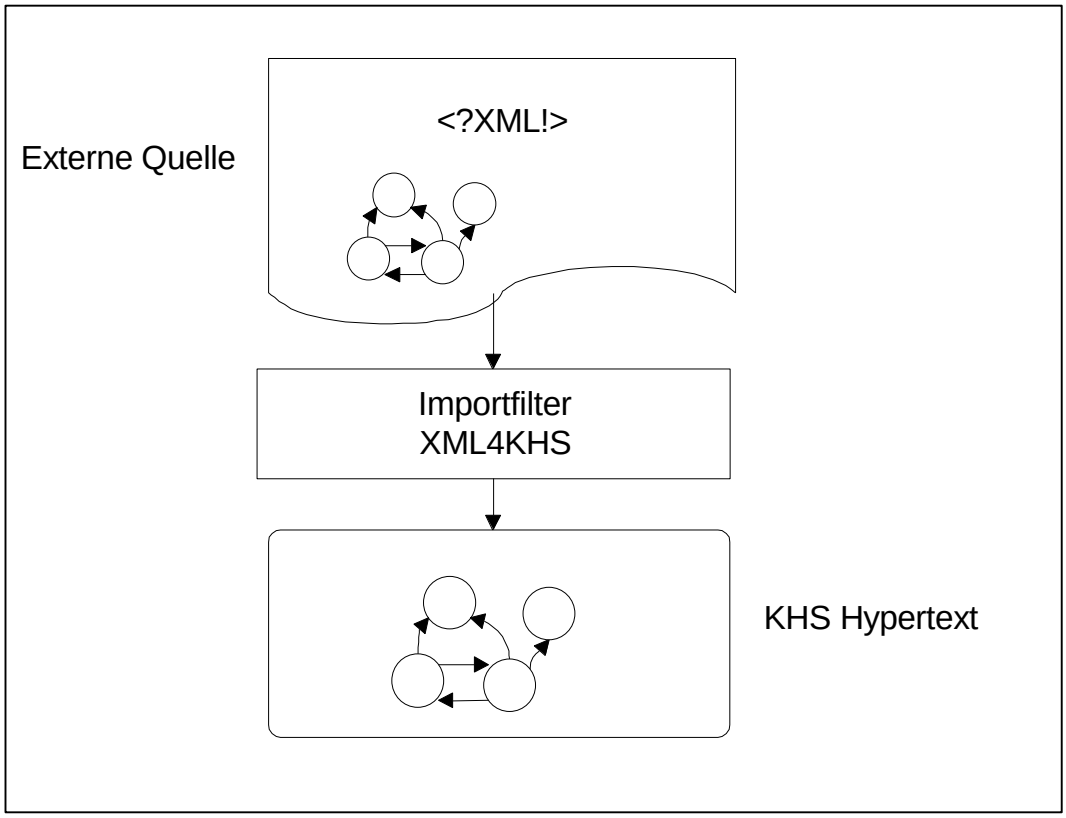


Abbildung 1: Schematische Darstellung des Importfilters. Quelle: eigen

2 XML-Technologie im Überblick

Einführend wird die “neue Dimension XML” insoweit beschrieben, als dies im Kontext der Transformation von XML-codierten Hypertexten in das KHS relevant erscheint. Im Rahmen der vorliegenden Arbeit kann die XML-Technologie dabei jedoch nur gestreift werden. Zur Vertiefung der Materie seien die **XML-Gesamtwerke** [Behme, Mintert 1998] und [Goldfarb, Prescod 1999] empfohlen. Verbindliche Informationen über diese Thematik und über den Stand der Spezifikation von XML sind über die Homepage des W3-Consortiums (URL: http://www.w3.org) erhältlich. Weitere Literaturhinweise werden im Laufe der folgenden Ausführungen gegeben.

Ursprung

Die Metasprache ***Ex**tensible **M**arkup **L**anguage* (XML) ist eine Untermenge der ***S**tandardized **G**eneralized **M**arkup **L**anguage* (SGML). Motivation für die Einschränkung von SGML, einem 1986 von der *International Organization for Standardization* (ISO) unter der Nummer 8879:1986 verabschiedeten Standard, war der Wunsch, den Anwendungsbereich des WWW zu erschließen[3]. Ausgangspunkt war folgende Problematik:
Während die SGML- *Applikation* HTML im WWW zwar technisch etabliert ist, verfügt sie über zu wenige Möglichkeiten im Bereich individueller Auszeichnung von WEB-Seiten. Demgegenüber bietet die Metasprache SGML ein großes Funktionsspektrum, allerdings mit dem Nachteil, daß die technologische Umsetzung aufwendiger ist. Das bedeutet: ein Browser, der sämtliche Optionen des SGML-Sprachumfanges und der zugehörigen Standards (Linksprache: HyTime, Stilsprache: DSSSL) unterstützen soll, muß extrem umfangreich sein [DuCharme 1999, S. 21].

Als Reaktion auf die WEB-Defizite von SGML initiierte 1996 Jon Bosak (Sun Microsystems) die Gründung einer neuen Arbeitsgruppe, der *„XML-Working Group“* des *World Wide Web Consortiums* (W3C). Das W3C ist eine 1994 ins Leben gerufene, Vereinigung[4] der weltweit

[3] [W3C 1998e, Kap. “origin and goals”]

[4] Quelle: [W3C 1999f], zur Geschichte des W3C siehe auch: http://www01.heise.de/ix/raven/Web/xml/timeline/w3c.html.

rund 350 Firmen und Institutionen angehören[5]. Das Engagement der von Bosak begründeten XML-Working Group hatte zwei entscheidende Maßnahmen zur Folge[6]:

- Verabschiedung der Sprachspezifikation von XML (V. 1.0) am 10. Februar 1998 durch das W3 Consortium,
- Erweiterung des SGML Standards ISO 8879 [Goldfarb 1998, S. 17] im Rahmen der sogenannten "WEB SGML Adaptions Annex", um gleichzeitig die Anforderungen a) "optimale Gestaltung von XML als WEB-Metasprache" und b) "XML als korrekte Untermenge von SGML" zu erfüllen.[7].

Terminologie

Im Folgenden ist teils von XML-Komponenten, teils von XML-Technologien die Rede. Beide Begriffe dienen der Beschreibung desselben Sachverhaltes: sie stehen für, teils angehende, *Standards, die in Verbindung mit XML* eingesetzt werden. Der Begriff „Technologie" weckt nach Ansicht des Autors jedoch eher die *berechtigte* Assoziation eines komplexen, technischen Konzeptes, als dies beim Begriff der XML-„Komponenten" der Fall ist.

2.1 Kern-Komponenten von XML

Zu den Kern-Komponenten von XML zählen neben der Inhalts-/Struktur-Komponente, Technologien, die die Bereiche Linking und Stil abdecken. Folgende Kern-Komponenten werden geboten:

- Inhalts-/Struktur-Komponenten: DokumentTyp Definitionen (DTD), XML-Schema-Sprachen
- Stil-Komponenten: Cascading Style Sheets (CSS level 1 + 2), Extensible Style Sheets (XSL) und Document Style Semantic and Specification Language (DSSSL),
- Link-Komponenten: Extensible Linking Language (XLL) / XPointer

Im folgenden werden die Komponenten für Inhalt und Struktur, sowie die Stil-Komponenten kurz erläutert. Aufgrund der besonderen Bedeutung für die Vernetzung von XML-Daten

[5] u. a. AT&T, Microsoft, Netscape, Sun - in Deutschland u.a. Deutsche Telekom, DFN Verein, FIZ Karlsruhe, GMD Darmstadt, SAP, Software AG (Quelle: http://www.w3.org/Consortium/Member/List, Stand 16.7.1999)

[6] siehe auch http://www01.heise.de/ix/raven/Web/xml/timeline/xml.html

[7] Die *Unterschiede* zwischen XGML und SGML werden in [W3C 1997] betrachtet.

werden die Link-Komponenten und ein weiteres Hypertext-Konzept im Kapitel 3.1 *ausführlich* behandelt.

2.1.1 Inhalts-/Struktur-Komponenten

Neben den *Dokument Typ Definitionen* (DTD) werden *XML-Schema-Sprachen* zur Definition des Inhaltes und der Struktur von XML-Dokumenttypen[8], sogenannter XML-*Applikationen*, verwendet. Hierzu gehört zum einen das Festlegen der Elementtypen („Tags“) und der zugehörigen Attribute, ferner wird auch die Struktur der zugehörigen Dokumente, d. h. die möglichen Kombinationen bzw. Schachtelungen von Elementen mittels des sogenannten Inhaltsmodells spezifiziert. Auch die Deklaration von sogenannten Entitäten wird vorgenommen[9].

Eine Gegenüberstellung der zwei alternativen Konzepte liefert folgendes Ergebnis:

2.1.1.1 XML-DTDs und XML-Schema-Sprachen im Vergleich

Die Syntax von XML-DTDs ist in der XML Sprachspezifikation eigens aufgeführt. Die DTD-Syntax weicht dabei von der Syntax, auf deren Basis XML-Dokumente erstellt werden, ab. Anders verhält es sich bei XML-Schemata: Diese werden mittels der gleichen Syntaxregeln wie XML-Dokumente erstellt, sind also XML-Applikationen.

XML-DTDs wurden bereits mit der XML-Recommendation verbindlich festgelegt, demgegenüber befinden sich XML-Schema-Sprachen, von denen übrigens etwa eine Hand voll verschiedener Konzepte miteinander konkurriert (DDML, DCD, SOX, XML-Data, W3C XML Schemas [vgl. Bourret 1999] - allesamt noch in der Entwurfsphase – keine dieser Sprachen liegt bislang (Stand 18.8.1999) als verbindliche W3C Recommendation vor.

---

[8] oft auch als Dokument-Klassen bezeichnet

[9] Entitäten sind Platzhalter analog der von HTML bekannten: ä = ä, &szlig=ß. Diese Entitäten werden wegen ihrer Lesbarkeit engl. „Parsed Entities“ genannt. Daneben existieren sogenannte „*Unparsed* Entities“ - *nicht* zu analysierende Zeichenfolgen wie beispielsweise Bild-Ressourcen.

Dabei bieten Schema-Sprachen gegenüber DTDs ein erheblich erweitertes Leistungsspektrum. Dieses wird von Ron Bourret [1999], der auch in Veröffentlichungen des W3C angeführt wird[10], folgendermaßen auf den Punkt gebracht:

"Schema language goals

- **use XML grammar**, tools, and technology,
- partially or completely replace DTDs,
- include human-readable documentation,
- define **reuse mechanism**, including namespace support,
- define **data typing** mechanism".

Ergänzend sei bemerkt, daß die verschiedenen Ansätze teilweise auch Optionen bieten, die aus dem Bereich der Software-Entwicklung/Daten-Modellierung bekannt sind. So bietet zum Beispiel die Schema-Sprache „XML-Data" die Möglichkeit, Beziehungen zwischen XML-Elementen mittels eines „Fremdschlüssel-Konzeptes" herzustellen und diese auch mit Kardinalitäten zu versehen[11] – analog des im Bereich Datenbank-Modellierung üblichen Entity Relation-Ship Modells. Das Konzept „*Schema for object oriented XML* (SOX)" der Firma VEO Systems bietet als Möglichkeit zur Wiederverwendbarkeit (reuse mechanism) Vererbungs-Konzepte, wie sie in der objektorientierten Programmierung geläufig sind.

Durch die genannten Vorzüge bedingt, ist es wahrscheinlich, daß XML-Schemata Sprachen mittelfristig das DTD Konzept ablösen werden.

Literatur

XML-DTDs werden bei [Behme, Mintert 1998] und [Goldfarb, Prescod 1999] erklärt – *die* W3C Referenz zu diesem Thema stellt die XML 1.0 Sprachspezifikation [W3c 1998e] dar.
Erläuterungen zu *XML-Schema-Sprachen* finden sich im Hinblick auf XML-Data bei [Pardi 1999, S. 217ff.; Goldfarb 1999, Prescod S. 484 ff] im Hinblick auf das *Schema for object oriented XML* (SOX) in [VEO 1999]. Eine Übersicht sämtlicher bedeutender XML-Schema Konzepte bietet [Bourret 1999]. W3C-Empfehlungen zu XML-Schema-Sprachen sind in

[10] [W3C 1999d, Kap. references (normative), Schlagwort XSchema)]

[11] vgl. [W3C 1998g, Kap. „One to many Relations", „Multipart keys" und „Constraints & additional properties"]

[W3C 1998f] (SOX), [W3C 1998g] (XML-Data), [W3C 1999d; W3C 1999e] („W3C“ XML-Schema) zu finden.

2.1.2 Stil-Komponenten

Die Stilsprache *Cascading Style Sheets* CSS fand bereits bei HTML Verwendung[12] und besitzt - gemessen an den zwei weiteren Stil-Komponenten XSL und DSSSL verhältnismäßig geringe Möglichkeiten. Sie unterstützt lediglich Formatierungen wie Schriftart, Farbe, Schriftgröße, Ausrichtung usw. für einzelne Elemente und kann dabei auch hierarchische Beziehungen (=Schachtelungen) von Elementen berücksichtigen.

Demgegenüber unterstützen die neu entwickelte *Extensible Style Language* (XSL) und die aus SGML stammende *Document Style Semantic and Specification Language* (DSSSL) einen mächtigeren Funktionsumfang. Beide enthalten neben einer vergleichbar leistungsfähigen Komponente zur Erzeugung von *Stil*-Information eine sogenannte *Transformations*-Komponente. Letztere wird über eine *integrierte Programmiersprache* zur Verfügung gestellt. Die Transformations-Sprache erzeugt dabei in der Regel sogenannte Flow-Objects, die typografischen Objekten[13] entsprechen [Behme, Mintert 1998, S. 117]. Im Falle von DSSSL wird Scheme, ein LISP Dialekt, eingesetzt - XSL unterstützt ECMA-Script, die standardisierte Version von Java-Script.

Als weitere erhebliche Erweiterungen von XSL bzw. DSSSL gegenüber den Cascading Style Sheets seien die Abfrage-Möglichkeiten mittels XSL-Pattern (XSL-Variante)[14] und SDQL (DSSSL-Variante) [15] genannt.

XSL ist übrigens eine XML-Applikation, CSS und DSSSL sind es nicht. Weitere Informationen zu allen drei Stilsprachen können bei [Behme, Mintert 1998] sowie speziell zu XSL [Pardi 1999], entnommen werden.

---

[12] vgl. [Muenz 1999] Der Standard CSS liegt mittlerweile in den Ausvbaustufen Level 1 und Level 2 vor.

[13] Absatz, Tabelle, Überschrift, Fußnote, …

[14] XSL Patterns sind eine spezielle Version der Extensible Query Language (XQL). vgl. [Pardi 1999, S. 187 ff.]

[15] Standard Document Query Language (SDQL) vgl. [Behme, Mintert 1998, S. 116 oben]

2.2 Periphere Komponenten

Neben diesen Kern-Komponenten existieren weitere, periphere Komponenten wie die *XML-Query Language* (XQL), *XML-Namespaces* und das *Document Object Model* (DOM - level 1 und 2).
Darüber hinaus sind schon jetzt zahlreiche (teils vom W3C bereits standardisierte) XML-*Applikationen*[16] verfügbar. Hierzu gehören das *Resource Decription Framework* (RDF), *Channel Defintion Format* (CDF), *Synchronized Multimedia Integration Language* (SMIL), *Scalable Vector Graphics Fornat* (SVG), *e-Commerce XML* (cXML), *Chemical Markup Language* (CML). Informationen hierzu können bei [W3C 1999b] entnommen werden - speziell zu cXML und anderen XML-E-Commerce-Konzepten [Heise 1999].

2.3 Stand der Spezifikation

Die meisten Komponenten von XML werden vom W3C spezifiziert. Einige Standards, die auch in Verbindung mit SGML eingesetzt werden, unterliegen der Standardisierung durch die ISO.

Lediglich folgende Standards sind bislang definitiv verabschiedet (Stand Juni 1999):

- XML-Sprachspezifikation (1.0),
- CSS (level 1 + level 2), DSSSL[17],
- XML-Namespaces,
- DOM (level 1).

Die weiteren oben genannten Komponenten, befinden sich noch in der *Entwurfsphase*. Je nach Stand der Entwicklung liegen sie - stets unverbindlich - als Notes, Working Drafts oder *Proposed* Recommendations vor (Erklärungen s. Anhang „W3C – Technical Reports“). Verbindliche Informationen über den Stand der Spezifikation bietet die Übersicht „W3C Technical Reports and Publications“ [W3C 1999b].

[16] XML-Applikation: spezifische, mittels XML definierte, Auszeichnungssprache.

[17] DSSSL wurde als SGML-Standard von der ISO unter der Nr. ISO/IEC 10179 verabschiedet

2.4 Zukunftsaussichten

Wie bereits im Kapitel “Motivation” erwähnt, wird XML eine große Verbreitung und Bedeutung zugeschrieben, nicht nur im ursprünglichen Einsatzgebiet des World Wide Web. Die Tatsache quasi täglich hinzukommender Erweiterungen deutet darauf hin, daß XML eventuell sogar SGML verdrängen könnte. Diplomatisch schreibt C. Goldfarb - wie bereits erwähnt, Erfinder von SGML - zu diesem Thema:

> “Some claim that XML will replace SGML because there will be so much free and low-cost software. Others assert that XML users, like HTML users before them, will discover that they need more of SGML and will eventually migrate to the full standard. Both assertions are nonsense […] XML and SGML **don’t even compete**” [Goldfarb 1998, S. 23].

3 XML Technologie ausführlich – Hypertext und XML

3.1 Link-Komponenten

Die Link-Technologien von XML unterteilen sich in zwei Bereiche: mittels *der Extended Link Language* (XLL) werden XML-Links, auch **XLinks** genannt, definiert, während **XPointer** der Definition von Ziel-Adressen, sogenannten Locators dienen. Die XPointer-Komponente ist der XLL-Komponente untergeordnet. Dies erklärt sich daraus, daß XPointer nicht ohne XLink eingesetzt werden können, umgekehrt ist dies jedoch möglich.

Beide Konzepte wurden maßgeblich durch bewährte Standards beeinflußt, eine XLink-Vorlage W3C [W3C 1999h, Kap. 1.3] schreibt hierzu:

> „Three standards have been especially influential:
> **HTML:** Defines several SGML element types that represent links.
> **HyTime:** Defines inline and out-of-line link structures and some semantic features, including traversal control and presentation of objects.
> **Text Encoding Initiative Guidelines (TEI P3):** Provides structures for creating links, aggregate objects, and link collections out of them.
> Many other linking systems have also informed this design, especially Dexter, FRESS, MicroCosm, and InterMedia“.

3.1.1 Vorbemerkung

Derzeit sind die XML-Linking Standards XLL und XPointer noch nicht definitiv vom W3C verabschiedet, sondern liegen lediglich in Form unverbindlicher W3C Arbeitsentwürfe, sogenannter Working Drafts[18] (WD), vor. Somit ist die letztendliche Ausführung der Link-Technologien noch ungewiß. Dies ist wahrscheinlich der Hauptgrund dafür, daß die Linking-Technologie von XML derzeit noch in *keinem* kommerziellen XML-Produkt implementiert ist, sondern lediglich in Browser-*Prototypen* wie Fujitsu's "Hybrick" oder Ludwigs "Link- an XML-XSL-XLL Browser"[19].

[18] siehe Anhang „W3C – Technical Reports“

[19] Fujitsu Hybrick: http://www.fsc.fujitsu.com/hybrick/ - Ludwigs Link: http://pages.wooster.edu/ludwigj/xml/index.html

Dennoch können die XLL/Xpointer-Working Drafts bereits jetzt als weitgehend festgelegt betrachtet werden. Goldfarb, Prescod formulieren diese Tatsache folgendermaßen:

> "Die grundliegenden [XLink/XPointer-] Konzepte sind fertig und werden sich nicht ändern, möglicherweise aber ihre Ausarbeitung" [Goldfarb, Prescod 1999, S. 510].

Im Folgenden beschränken sich die Ausführungen auf solche Komponenten, die von Goldfarb und Prescod als stabil eingestuft wurden.

3.1.2 XLinks (XLL)

Xlinks stellen den angehenden Standard zur Definition eigener Link-Elemente dar. Syntaktisch werden Link-*Elemente* gemäß dem aktuellen XLink-Working Draft (26.7.1999) durch das Attribut **xlink:type** gekennzeichnet. Damit wird die Kennzeichnung xml:link aus dem Jahre 1998 erwartungsgemäß ersetzt[20].

Grundsätzlich werden XLinks in zwei Grundtypen unterteilt*:*

1. Einfache Links,
2. Erweiterte Links.

In Verbindung mit Xlinks werden auch sogenannte Link-Gruppen eingesetzt. Diese definieren - um es vorwegzunehmen - eine Gruppierung von XLink-*Dokumenten*. Link-Gruppen werden in Kapitel 3.1.2.4 gesondert behandelt.

3.1.2.1 Einfache Links

Einfache XML-Links funktionieren zunächst wie das bereits aus HTML bekannte Prinzip der Verknüpfung von zwei Ressourcen. Die Position des Link-Elementes selbst stellt einen Endpunkt (i.d.R. ein Hotword) dar, - in der WC-Spezifikation verallgemeinernd als

[20] Der Umstellung von xml:link zu xlink:type ging eine Diskussion in der XML-DEV Mailing List im Mai 1998 voraus. Grund: Die Zeichenfolge "xml" ist nicht erlaubt. (XML-Recommendation [W3C 1998e] Abschnitt: 2.3, Absatz: 3 "(('x' | 'X')('m' | 'M') ('l' | 'L')) are reserved for standardization in this or future versions of this specification"). Diese Frage wurde mit dem - mittlerweile umgesetzten - Vorschlag von Eve Maler (einem Mitglied der XML Working Group des W3C) beantwortet:

> "we voted to change the xml:link attribute to xlink:form**,** though this doesn't appear in the spec yet. You'll see it in the next draft. (No, I'm not sure when that will be. :-)" [Maler 1998].

“participating Ressource” bezeichnet. Diese Eigenschaft wird als “inline”, im Sinne von: „der Verzweigungspunkt ist an der Stelle des Linkelementes (in der gleichen Zeile) positioniert“, bezeichnet. Es sind jedoch auch simple Links denkbar, bei denen die Postion des Verzweigungspunktes von der des Link-Elementes *abweicht*. Solche Links, die als „out-of-line“ bezeichnet werden, werden im Kapitel „Exkurs: Inline und Out-of-line Links“ näher erläutert. Das **Attribut xlink:type** erhält bei einfachen Links den Wert **„simple“**.

Beispiele für einfache Links:

a) URL als Locator

```
<MySimpleLink xlink:type ="simple" xlink:href="MyXmlDoc.xml"/>
```

enspricht in HTML

```
<a href="MyXmlDoc.xml"></a>
```

b) XPointer als Locator

```
<MyXpointerLink xlink:type="simple" xlink:href=id(x709x).child(1,Name)>
```

Verweist auf das 1. Element vom Typ “Name”, das dem Element mit der id x709x untergeordnet ist.

3.1.2.2 Erweiterte Links

Erweiterte Links stellen ein größeres Funktionsspektrum als einfache Links zur Verfügung. Sie können:

- n-direktional sein, d. h. mittels eines Links kann auf quasi beliebig viele Ressourcen verwiesen werden,
- den zugehörigen Verzweigungspunkt (Hotword, Icon, …) an der *exakten* Postion des Linkelements definieren (inline), oder einen Verzweigungspunkt, dessen Position von der des Link-Elementes *abweicht*, definieren (out-of-line),
- in Ressourcen, auf die keine Schreibrechte existieren (“read only”) positioniert werden!,

- fremde Ressourcen[21] in das aktuelle Dokument einbetten (“embedding links”).

Erweiterte (engl extended) Links bündeln mehrere Links mit Adressangaben (engl. locators). Formal umschließt das zugehörige Extended-Link-Element die betreffenden Locator-Elemente (href=…). Die einzelnen Adressierungs-Elemente dienen dabei der Spezifikation von Enden (Ausgangspunkt/ Endpunkt)[22] eines Links. Das **Attribut xlink:type** erhält bei erweiterten Links den Wert **„extended“**.

Beispiele zu erweiterten Links können dem Kapitel 3.1.4 „XLinks und XPointer: Fallbeispiele“ entnommen werden.

3.1.2.3 Exkurs: Inline und Out-of-line Links

Neben der Unterscheidung von einfachen und erweiterten Links werden sogenannte inline- und out-of-line-Links, je nach Wert des **Attributes xllink:inline** unterschieden. Als inline bzw. out-of-line können sowohl einfache als auch erweiterte Links gekennzeichnet werden.

Inline Links sind bereits aus HTML bekannt. Charakteristisch für diese Links ist die Tatsache, daß die Position des Link-Elementes (<A href=... >Hotwordtext</A>) gleichzeitig die Position des Hotwords darstellt. Demgegenüber stellen **out-of-line-Links** eine wesentliche Erweiterung gegenüber den aus HTML bekannten inline-Links dar. Die Erläuterung dieses Prinzips ist in mancher Literaturquelle etwas salopp, um nicht zu sagen: die Aussagen widersprechen sich teilweise. Ein Grund für diese Tatsache mag darin liegen, daß speziell das XLINK-Working Draft [W3C 1999h] sehr knapp (und teilweise ein wenig kryptisch) formuliert ist und wenig anschauliche Beispiele enthält (im Gegensatz z. B. zum WD über Xpointer).
Das Spektrum der *Mutmaßungen* zu diesem Thema reicht von:

[21] = ganze Dokumente oder Dokument-Teile

[22] Ein Endpunkt eines Links kann in speziellen Fällen *gleichzeitig* Ausgangs- und Endpunkt sein!

> "der Unterschied wird alleine dadurch festgemacht, *wohin* ein Link verweist – demnach ist ein link, dessen Ziel außerhalb des Quell-Dokumentes liegt, out-of-line" (verkürzte Übersetzung aus) [Homer 1999, S. 94].

bis zu:

> "Ein Inline link ist wie im A-Element in HTML einer, der Bestandteil des Linkelementes ist. Im Gegensatz dazu ist das beim Out-of-line link nicht der Fall. Vielmehr enthält hier *ein im Linkelelement eingeschlossenes Element* die eigentliche Adreßangabe" [Behme, Mintert 1998, S. 89].

Die beschriebene Divergenz der Aussagen brachte den Autor dazu, eigene Nachforschungen anzustreben. Wichtige Hinweise erhielt er dabei von Justin Ludwig, Autor von [Ludwig 1999], der, als Entwickler eines Xlink-verarbeitenden Systems, bereitwillig sein Wissen teilte[23]. Durch das Experimentieren mit Fujitsu's Experimental-Browser „Hybrick" konnten die Kenntnisse ebenfalls vertieft werden.

Das XLink WD von 1998 [W3C 1998a] schreibt im Glossar zum Thema out-of-line

> **"out-of-line link**: A link whose content does not serve as one of the link's participating resources. Such links presuppose a notion like extended link groups, which indicate to application software where to look for links. Out-of-line links are generally required for supporting multidirectional traversal and for allowing read-only resources to have outgoing links".

[W3C 1999a, Kap. 3 "Terminology"] stellt dieses Konzept folgendermaßen dar:

> **"inline/out of line:** In order to make it possible to express links all of whose ends are read-only, many hypermedia systems provide a way to encode links in some place external to the document(s) containing any of their ends. A link that makes use of this capability is said to be stored "out of line", while one whose own location is one of its ends is "inline"".

[23] Anmerkung von Justin Ludwig in einer Email vom 11. Juni 1999 "I know it [comparison of inline and out-of-line links] can be very confusing, even (or especially!) after dilligent study".

Mit out-of-line-Links werden also zwei Erweiterungen gegenüber inline-Links geboten:

- multidirektionale Verzweigung ("traversal"),
- Zugriff auf read only Ressourcen.

Multidirektionale Verzweigung

Die multidirektionale Verzweigung wird von out-of-line Links über die Angabe von verschiedenen Ressourcen in Verbindung mit einem *extended* Link realisert. Der Unterschied gegenüber inline Links ist dabei, daß jede der beteiligten Ressourcen als Traversierungs-*Start*punkt auftreten kann. Das definierende Link-Element *selbst* markiert keinen Endpunkt des Links, so wie dies beispielsweise bei HTML-Links der Fall ist, wo die Position des Link-Elementes gleichzeitig einen Endpunkt des Links (hier: Startpunkt = i. a. "Hotword"). Dieser Effekt wird im Teil b) des Kapitels „XLinks und XPointer: Fallbeispiele" illustriert.

Zugriff auf Read-Only Ressourcen

Diese Technik, die bereits von HyTime, einem *SGML*-Link Standard, ermöglicht wurde, wird XML auch im WEB Bereich etablieren. Die **Realisierung** des Zugriffs auf Read-Only Ressourcen kann über Link-Groups nach dem sogenannten *Hub-Document-Prinzip* erfolgen [W3C 1998a, Kap. 5][24]. Hierzu wird beispielsweise in einer Link-Group des aktuellen Dokumentes ein Verweis auf eine lokale Datei (z. B. C:\Comments\HubDoc.xml) eingetragen. Dadurch wird in diesem Dokumenten nach (out-of-line)Links gesucht, die vom aktuellen Dokument ausgehen.

Eine weitere Vision wird von Goldfarb und Prescod ausgemalt: Danach bestünde auch die Möglichkeit, den *standardisierten* Zugriff auf weltweite (out-of-line)*Link-Datenbanken* zu installieren [Goldfarb, Prescod 1999, S. 515].

Es sei angemerkt, daß out-of-line-Links in seltenen Fällen die skurrile Form annehmen können, lediglich einen Startpunkt und *keinen* Endpunkt zu bezeichnen. Diese Form, welche die Einzige „out-of-line" Variante von *simple* Links darstellt, wird als „one ended" bezeichnet.

[24] "For example, should a group of documents be organized with a single "hub" document containing all the out-of-line links, it might make sense for each non-hub document to contain an extended link group containing only one reference to the hub document". [W3C 1998a, Kap. 5]

3.1.2.4 Gruppen von Dokumenten mit erweiterten Links

Gruppen von Dokumenten mit erweiterten Links, mißverständlich oft als "Gruppen von (erweiterten) Links" bezeichnet, nehmen eine wichtige Funktion bei der Verarbeitung von out-of-line Links wahr. Sie weisen eine XML-Anwendung auf weitere Dokumente hin, wleche (out-of-line) Links die das aktuellen Dokument betreffen, enthalten. Link-„Gruppen"-Elemente umschließen dabei Link-Elemente der Form „Dokument".

Hierzu ein Beispiel:

```
<group xlink:type="group" steps="2">
    <groupdoc xlink:type="document" href="hubdoc.xml" />
    <groupdoc xlink:type="document" href="fm.xml" />
</group>
```

In den mit dem Attribut **xlink:type="document"** ausgezeichneten Elementen werden weitere Dokumente referenziert, in denen nach Links gesucht wird, die vom aktuellen Dokument *ausgehen*[25]. Das Attribut steps gibt an, inwiefern die Extended Link groups der referenzierten Dokumente berücksichtigt werden. Hat es den Wert „1", so wird lediglich in den referenzierten Dokumenten selbst nach Extended Links gesucht. Der Wert „2" bewirkt die Suche in den von diesen Dokumenten wiederum referenzierten Dokumenten. Der Wert läßt sich beliebig erhöhen, - mittels „step" wird somit die "Rekursions-Tiefe" der Suche nach out-of-line-Links vorgegeben.

3.1.2.5 Syntax: Attribute von XLinks

Je nach Funktion können verschiedene Kategorien von Attributen unterschieden werden:

1. **Grundfunktion des Links,**
2. **beteiligte Ressourcen,**
3. **Semantik des Links,**
4. **Link Verhalten.**

[25] es wird *nicht*, wie vielleicht vermutet werden könnte, nach Links gesucht die auf das aktuelle Dokument verweisen.

1. Grundfunktion des Link-Elementes

Attribut-Name	xlink:type, vormals xml:link[26]
Beschreibung	spezifiziert die Grundfunktion (“form”) des Links-Elementes: e
Mögliche Werte	“extended” (einfach), “simple”(erweitert), locator (Bestandteil eines erweiterten Link Elementes), “group” (Gruppe von Xlink Dokumenten), “document” (Bestandteil einer Gruppe von Xlink Dokumenten)

2. Beteiligte Ressourcen

Attribut-Name	inline
Beschreibung	gibt an, ob ein inline oder out-of-line-Link vorliegt
Mögliche Werte	true, false

3. Semantik der beteiligten Resourcen

Attribut-Name	content-role (inline links) / role (out-of-line links)
Beschreibung	Bezeichnet die Semantik (“Rolle”) einer beteiligten Ressource[27] (Link auf: Kommentar/Lexikoneintrag/Autor …), die von der verarbeitenden Anwendungssoftware berücksichtigt werden kann.
Mögliche Werte	beliebige Zeichenfolgen – in Anführungstrichen eingeschlossen
Attribut-Name	content-title (inline links) / title (out-of-line links)
Beschreibung	Bezeichnet einen Titel einer beteiligten Ressource, der von der Anwendungssoftware gegebenenfalls angezeigt werden kann.
Mögliche Werte	beliebige Zeichenfolgen – in Anführungstrichen eingeschlossen

4. Link-Verhalten

Attribut-Name	show
Beschreibung	bestimmt, wie im Falle der Traversierung eines Links die beteiligten

[26] Die im XLink Working Draft vom 26.7.1999 [W3C 1999h] vorgeschlagene Kennzeichnung **xlink:type** löst die Notation **xml:link** des ersten WD vom 3.3.1998 ab.

[27] Definition Ressource: Dokument oder Teil eines Dokumentes. Im obigen Kontext zählen hierzu auch die Ausgangspunkte von Links:Hotwords, Icons, usw. .

	Ressourcen visualisiert werden.	
Mögliche Werte	**embed**	"embedding link" - In der Quell-Ressource (Ursprung des Links) , d. h. an der Position des Linkelementes wird die Ziel-Ressource angezeigt (eingebettet).
	replace	Die Ziel-Ressource ersetzt das aktuelle Dokument.
	new	Das aktuelle Dokument bleibt unberührt, die Ziel-Ressource erscheint in einem neuen Kontext (z. B. neues Browser Fenster)

Attribut-Name	actuate	
Beschreibung	bestimmt, welche Ereignisse die Traversierung eines Links auslösen	
Mögliche Werte	**auto**	Der betreffende Link wird bei Aktivierung eines *weiteren* links, welcher in der Regel in demselben (extended) Link-Element enthalten ist, ebenfalls ausgelöst. Im Falle von embedded links (s. Attribut show) kann dieser Wert bewirken, daß die fremde Ressource beispielsweise beim *Eintreten* in das Quell-Dokument angezeigt wird [Behme 1998, S. 89]).
	user	Der Link kann nur über explizite User-Anforderung (z. B. Anlicken eines Hotwords) ausgelöst werden.

3.1.3 XPointer

Vorbemerkung

Xpointer werden an dieser Stelle wesentlich knapper als Xlinks behandelt. Der Grund ist die geringere Komplexität dieses Konzeptes, das im W3C Working Draft [W3C 1999i] umfangreich kommentiert ist. Das vormalige WD [W3C 1998c] wurde mit diesem, jüngst erschienenen, Dokument grundlegend überarbeitet.

XPointer bieten erweiterte Adressierungen[28] (engl. „Locators") von Links im Vergleich zu HTML. Bei HTML kann auf WEB-Dokumente lediglich wie folgt verwiesen werden:

- die Angabe einer URL (Bsp: http://www.MeineDomaene.de/doc.html) verweist auf den Anfang eines Dokuments.

[28] i. d. R. Ziel-Adresse

- die Angabe einer URI samt *Fragmentbezeichner* (Bsp: http://www.MeineDomaene.de/doc.html**#47**) verweist auf eine *Textmarke* innerhalb eines Dokuments.

Dem gegenüber ermöglichen Xpointer erweiterte Möglichkeiten. Zur Adressierung können folgende Ausdrücke verwendet werden (die Kapitelangaben beziehen sich auf [W3C 1999i]):

- Xpath Relative Axis (Angabe[29] des relativen Pfades), Kap 4.3
- Xpath Absolute Axis (Absolute Pfadangabe), Kap. 5.3
- Range Axis (Angabe eines Bereiches), Kap. 5.2.1, 5.5.3
- String Axis (Angabe einer Zeichenfolge). 5.2.2

Ein Beispiel für XPointer wurden bereits im Kapitel „Einfache Links“ erläutert, weitere sind im folgenden Kapitel zu finden.

3.1.4 XLinks und XPointer: Fallbeispiele

Anmerkung: Es wird im Falle der XPointer die Notation des Working Drafts von 1998 verwendet, - die Notation vom 9. Juli 1999 konnte nicht berücksichtigt werden, da diese erst kurz vor Abgabetermin der vorliegenden Arbeit erstmalig erschien. Der verwendete XLink-Standard entspricht - wenn nicht gegenteilig angemerkt - dem Stand des WD vom 26. Juli 1999.

a) Erweiterte Links (inline)

Folgendes Beispiel für einen inline Extended-Link wurde Ludwig [1999, Kap. 4.1.2] entnommen: Hier werden verschiedene multimediale Daten, die zu dem Katalog-Artikel

[29] Die vorgenommene deutsche Übersetzung „*Angabe* des …“ ist eine *Interpretation* des Begriffs „Axis“. Die genaue Definition dieses Ausdruckes findet sich in [W3C 1999i, „Glossar“].

„Franks bun-size Beef“[30] eines Hotdog-Standes gehören, innerhalb eines extended Links zusammengefaßt.

```
<HotdogLink     xlink:form[31]="extended"
                xlink:role="hotdog display"
                xlink:content-role"link text"
                xlink:inline="true">
        <LocalHotdogContent>Bun-sizebeef frank</LocalHotdogContent>
        <HotdogContent  xlink:form="locator"
                        xlink:href="gifs/bsbeef.gif"
                        xlink:role="gif"
                        xlink:show="embed"
                        xlink:actuate="auto"/>
<HotdogContent  xlink:form="locator"
                        xlink:href="descriptions/bsbeef.txt"
                        xlink:role="description text"
                        xlink:title="Bun-size Beef Frank"
                        xlink:show="replace"
                        xlink:actuate="user"/>
</HotdogLink>
```

Ein Browser könnte diesen Code folgendermaßen darstellen: Als *Hotword-Text* dient „Bun-size Beef frank“. Die Bild-Datei mit dem *Pfad* „gifs/bsbeef.gif“ (xlink:href="gifs/bsbeef.gif") wird zur Illustration des Produktes *automatisch* (xlink:actuate="auto") *eingebettet* (xlink:show ="embed"). Durch Anklicken des Hotwordes durch den *Benutzer* wird ein Traversierungsvorgang ausgelöst (xlink:actuate="user"). Daraufhin wird der *Beschreibungstext* (xlink:role="description text"), der in der Datei „descriptions/bsbeef.txt“ enthalten ist, im Browser angezeigt. Dabei *ersetzt* dieser den aktuellen Fenster-Inhalt (xlink:show="replace").

b) Erweiterte Links (out-of-line), erweiterte Link Gruppen

Die nachfolgende Darstellung illustriert die Anwendung von Extended Links, Xpointern und Extended Link Groups. Als Grundlage dienten drei XML-Dokumente (bernoulli.xml,

[30] Frei übersetzt: Franks „Riesen“-Rinder-Steak

[31] [Ludwig vermutete, daß das Attribut xml:link des XLink WD von 1998 durch xlink:form abgelöst würde, wie von Eve Maler seinerzeit in [Maler 1998] vorgeschagen. Nach dem jüngsten WD [W3C 1999h] lautet die korrekte Syntax jedoch wie beschrieben xlink:type.

BernoullisP.xml, fm.xml) die dem Xlink-Beispiel “TopicMaps” des Fujitsu Browsers Hybrick entnommen sind[32].

Die auf der folgenden Seite abgebildete Grafik[33] illustriert, wie ein **Browser** auf das Anklicken eines multi-direktionalen extended Links reagieren könnte. Auf der linken Seite ist das Ausgangsdokument zu sehen, in welchem das Hotword „Known for Bernoulli‘s Principle“ mit der Maus angeklickt wurde. Daraufhin wurde ein Popup-Menü geöffnet, das die Verzweigung zu den drei Link-Ressourcen (von denen zwei in den anderen Dateien liegen, und eine in der Datei selbst) erlaubt. Im Popup-Menü sind die Rollen der Links zu sehen. Durch Klick auf eine der drei Rollen-Bezeichner (Fluid Mechanics, B’s Principle Reference, Bernoulli’s Principle) wird auf die entsprechende Ziel-Ressource, die im Code mittels XPointern spezifiziert wurde, verzweigt. Der danach abgedruckte **Source Code** zeigt die vom betreffenden Extended Link ausgehenden Verzweigungen, die per XPointer-Locator definiert sind.

[32] Die Beispiel-Dateien finden sich im Installationsverzeichnis unter .\TopicMaps.

[33] Original Screenshots des Browsers Hybrick.

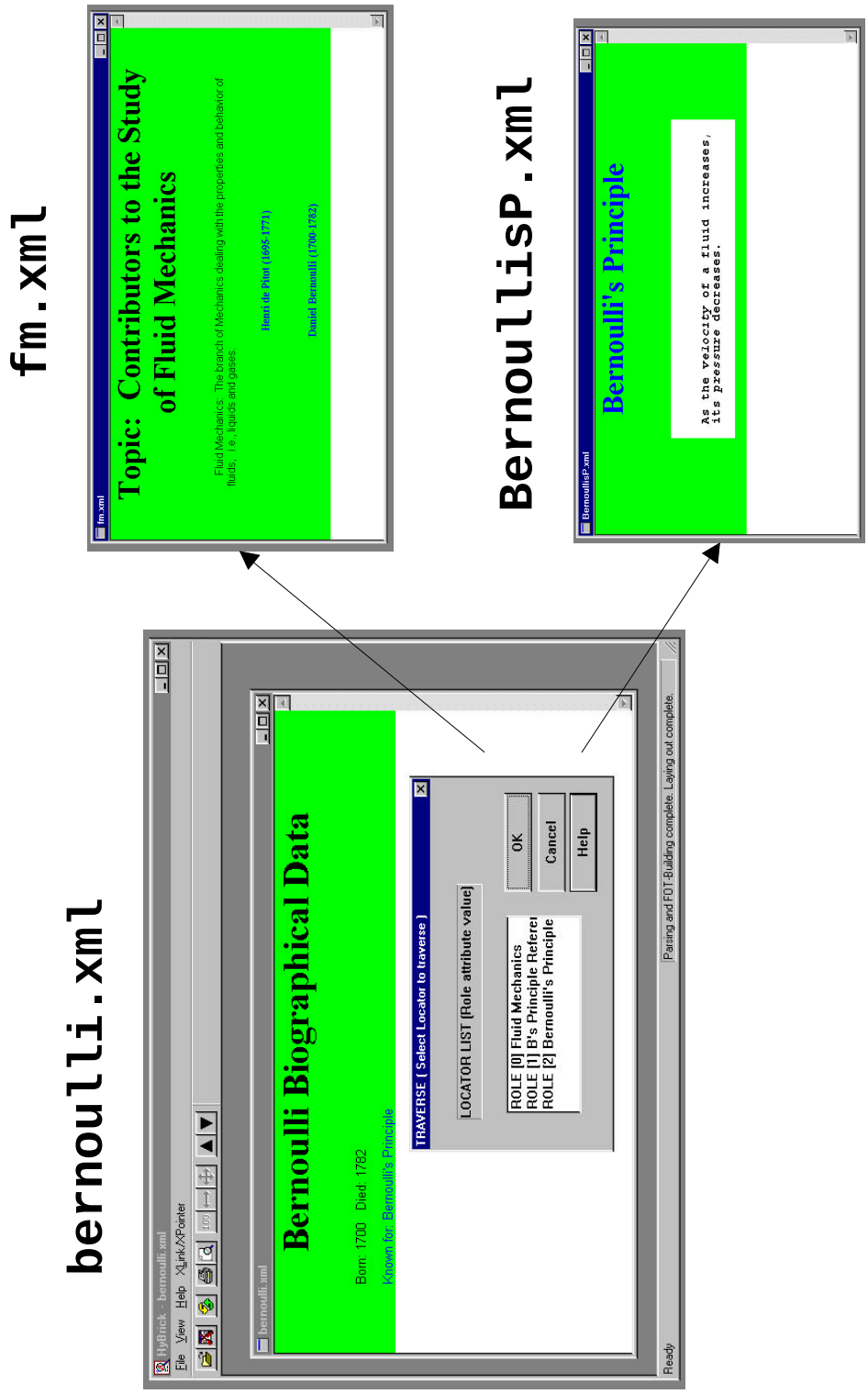


Abbildung 2: Xlinks - Darstellung der Dateien im Browser. Quelle: eigen

Abbildung 3: Xlinks - Darstellung des XML-Codes. Quelle: eigen

Erläuterung zum Source-Code des out-of-line Beispiels:

1) Hier wird eine extended Link Group (xml:link=”group”) definiert. Durch dieses Konstrukt wird in den Dokumenten hubdoc.xml und fm.xml nach weiteren (out-of-line) Extended Links gesucht, die das Dokument bernoulli.xml betreffen. Der Wert steps=”2” bewirkt, daß die Dokumente der Extended Link-Group selbst und wiederum deren Extended Link-Groups berücksichtigt werden. Wollte man diese “rekursive” Berücksichtigung von Dokumenten noch fortsetzen, so wäre der Wert zu erhöhen. (**Hinweis!** Die verwendete Notation entstammt dem Xlink Working Draft vom 3.3.1998).

2) Referenzen auf Dokumente, die über die Extended Link Group (s. oben Punkt 1) referenziert werden.

3) (out-of-line) Extended Link. Es verweisen Links auf drei Ressourcen außerhalb des Extended Links.

4) Locator-Links, die mittels Xpointern definiert sind (**Hinweis!** Notation gemäß dem XPointer Working Draft vom 3. März 1998).

3.2 Logische Beziehungen von Elementen: IDs und IDREFs

Ein wichtiges Konzept zur Herstellung von Beziehungen zwischen Elementen in XML-Dokumenten stellen die Ausprägungen „ID“, „IDREF“ und „IDREFS“ des Attribut-Typs „Token“ dar (XML-Sprachspezifikation [W3C 1998e, Kap. 3.3.1]).

Wird in einer Dokumenttyp-Deklaration ein Attribut mit dem Wert „ID“ deklariert, so hat das zur Folge, daß Elemente welche dieses Attribut verwenden, eindeutige[34] Werte für dieses Attribut verwenden müssen. Die resultierenden Identifikations- Schlüssel können von Attributen, die als „IDREF“ bzw. „IDREFS“ deklariert sind, referenziert werden, wodurch ein logischer *Bezug* zwischen Elementen hergestellt werden kann.

[34] = eindeutig *innerhalb des Dokumentes*

Durch die Verwendung von „IDREF“-Attributen (ohne „S“) können *genau zwei* Elemente logisch verknüpft werden. Das IDREF-Attributes des ersten Elementes erhält hierzu den Wert des ID-Attributes des zweiten Elementes. Analog ist unter Verwendung eines IDREF*S*-Attributes die Verknüpfung von *mehr als zwei* Elementen möglich.

Obwohl dieses Konstrukt im Gegensatz zu XLinks keine Traversierungsfunktionalität bietet, wird es in dem XLink-WD [W3C 1999h] als „linking relationship“ bezeichnet.

Folgendes Beispiel hierzu wurde einem XML-Dokument zur Repräsentation von KHS Forumsbeiträgen entnommen:

```
<Forum idXml="x12345x" >
    <Name>Test-Forum</Name>
    <Text>Willkommen im Testforum.</Text>
</Forum>
<Heading idXml="x000x">
    <Name>Test-Rubrik </Name>
    <Text>Text zur Test-Rubrik</Text>
</Heading>
<HierarchyLink From="x12345x" To="x000x"/>
<User idXml="x709x">
    <FirstName>Uwe</FirstName>
    <Name>Speck</Name>
    <EMail>specku@fmi.uni-konstanz.de</EMail>
    <City>Konstanz</City>
</User>
<ForumDocument idXml="x1x">
    <Name>Erst hell dann dunkel</Name>
    <SubTitle>Sonnenfinsternis am 11.8.1999</SubTitle>
    <Text>Sie kommt bestimmt!</Text>
</ForumDocument>
<HierarchyLink From="x000x" To="x1x"/>
<AuthorDocuLink From="x709x" To="x1x">
    <AboDepth>2</AboDepth>
</AuthorDocuLink>
```

Die eingesetzten Attribute mit dem Namen **idXml** wurden dabei mittels “ID” und die Attribute From und To durch “IDREF” definiert. Ein Auszug aus der Dokumenttyp-Definition illustriert dies :

```
<!ATTLIST User
 idXml ID #REQUIRED>
```

beziehungsweise

```
<!ELEMENT HierachyLink EMPTY >
<!ATTLIST HierarchyLink
       From IDREF #REQUIRED
       To IDREF #REQUIRED >
```

Mit Elementen vom Typ HierachyLink und AuthorDocuLink werden also Beziehungen zwischen (zwei) Elementen hergestellt. Es ergibt sich folgender logischer Zusammenhang zwischen den Elementen des Beispiels:

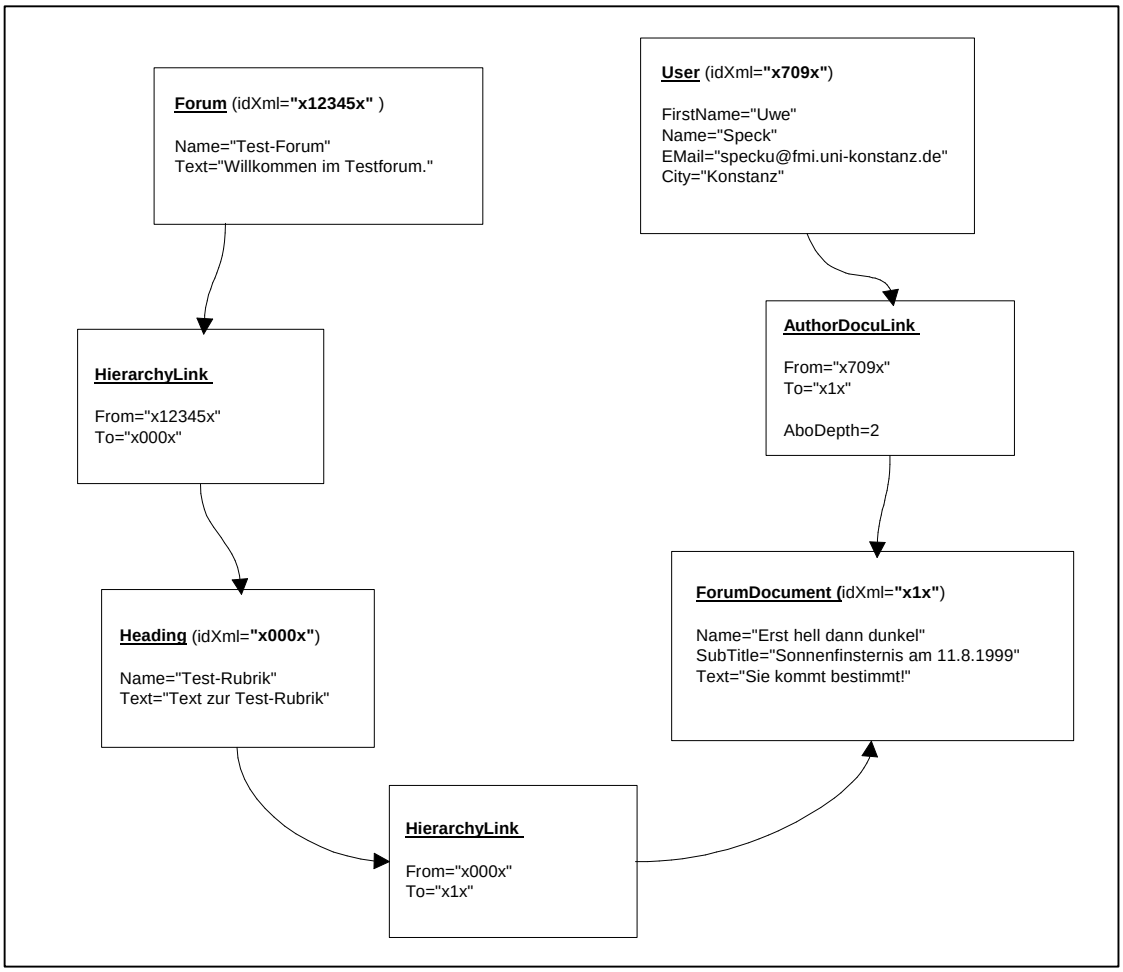


Abbildung 4: Logische Beziehungen zwischen Elementen mittels ID/IDREF. Quelle: eigen

3.2.1 Integritätsbedingungen

Die **Integritätsbedingungen**, die von XML-Prozessoren bei Anwendung der Attribut-Deklarationen der Art ID bzw. IDREF/S in der Regel geprüft werden, sind:

- Eindeutigkeit von Werten von ID-Attributen (“Primärschlüssel”)
- nur ein ID-Attribut pro Element erlaubt
- Existenz von Attribut-Werten, auf die von IDREF und IDREFS Attributen verwiesen wird (“referentielle Integrität”)

Somit ist mittels dieser Konzepte eine konsistente Verwaltung von Elementen, respektive von durch Elemente codierten Entitäten möglich. Allerdings sind diese Integritätsbedingungen gegenüber denen, die man beispielsweise von relationalen DBMS her kennt, *rudimentär* (vgl. [W3C 1998e, Kap. 3.3.1]).

Einschränkungen der Integritätsbedingungen

- Im Gegensatz zu den Primärschlüsseln von RDBMS, bei denen der Gültigkeitsbereich beschränkt ist (auf die jeweilige *Tabelle*), ist der **Gültigkeitsbereich** von IDs *global* und nicht - wie man erwarten würde - auf einzelne Elementtypen beschränkt. Somit wird die Integritäts-Bedingung der Eindeutigkeit von IDs stets verletzt, wenn zwei Elemente die gleiche ID besitzen, – unabhängig von den zugehörigen Elementtypen.
- Auch die Verwendung von IDREF/S deklarierten Attributen hat erhebliche Einschränkungen referentielle Integrität gegenüber der welche von Tabellen-Relationen in relationalen DBMS gewährleistet wird. Da hier keine Beschränkung auf Elementtypen vorliegt, ist eine *typisierte* referentielle Integrität nicht umzusetzen.

Es sei darauf hingewiesen, daß, – wie bereits in Kapitel 2.1.1.1 erwähnt - mittels XML-Schema-Sprachen Integritätsbedingungen zur Verfügung gestellt werden, die sowohl typisierte, als auch (mittels Kardinalitätsangaben) eingeschränkte Relationen zwischen Elementen erlauben

4 DAS Konstanzer Hypertext System

Nachfolgende Erläuterungen beziehen sich vor allem auf [Odenthal, Aßfalg, Handschuh 1998; Kuhlen 1996].

4.1 Zweckbestimmung

Das offene Konstanzer Hypertextsystem, das im Rahmen des DFG-Forschungsprojektes WITH im Jahre 1992 an der UNI-Konstanz entstand, dient der Verwaltung hochgradig vernetzter Dokumentenmengen, wie sie bei Lexika, Campus-Informationssystemen, Diskussionsforen, CSCW-Systemen, usw. anfallen. Das Haupteinsatzgebiet des KHS ist seit Jahren der Bereich **elektronischer Kommunikationsforen**.

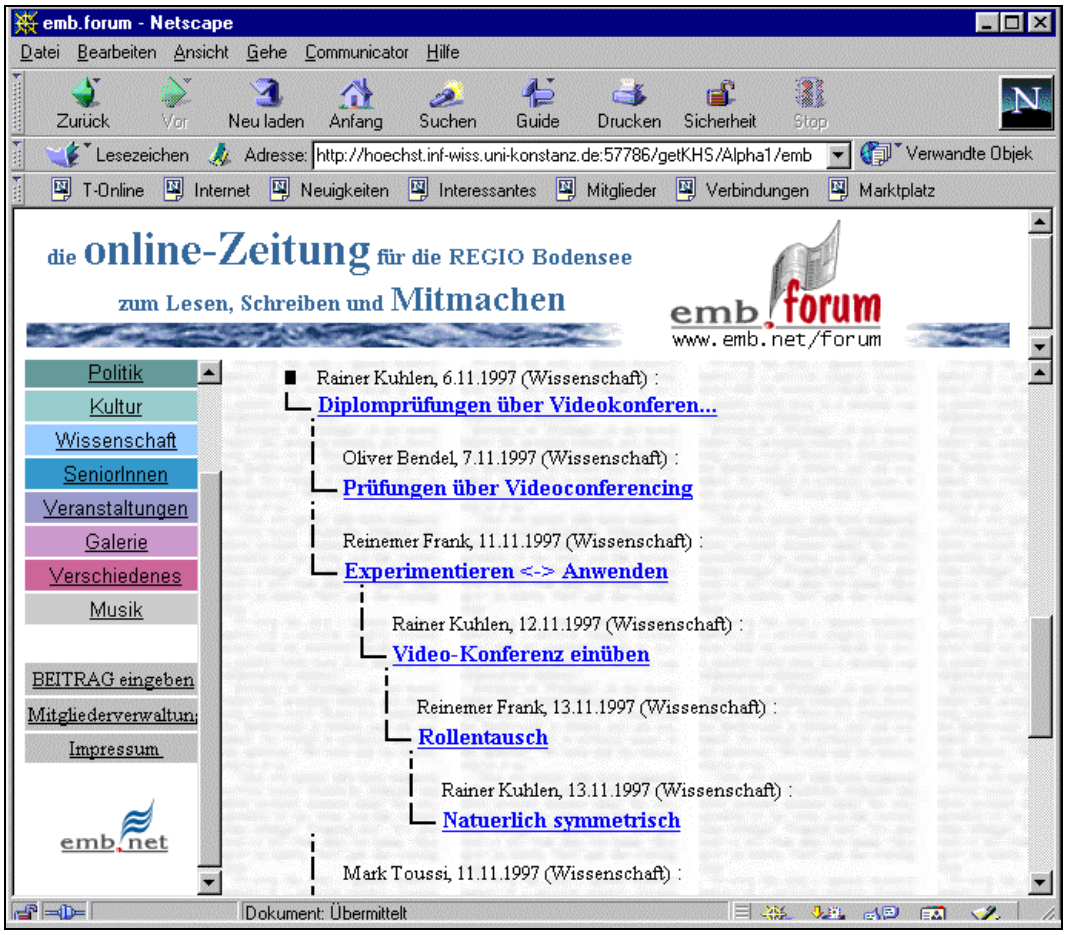


Abbildung 5: Diskussionsverlauf in einem Forum des Konstanzer Hypertext Systems. Quelle: eigen

Während das System in dieser Betriebsart dabei scheinbar wie eine gewöhnliche Internet-Newsgroup[35] wirkt, weist es in seiner Eigenschaft als offenes Hypertextsystem (vgl. [Kuhlen 1996, S. 432ff.]) wesentliche *konzeptionelle* Unterschiede zu jenen auf.

Gegenüber der festen Baumstruktur ("Diskussionsverlauf der Beiträge") liegt im Falle des KHS eine flexible Hypertext-Struktur vor, flexibel in bezug auf die verwalteten Objekte/ Verknüpfungen und in bezug auf die Organisationsformen der Information. Beim KHS ist eine Vielzahl von möglichen Benutzer-Kategorien (u. a. demnächst auch sog. Experten) samt einer umfangreichen Konfiguration von (Lese-/Schreib-)Zugriffsrechten möglich. Ferner ist der Ausgabe-Frontend des KHS html-basiert, weshalb Forums-Beiträge auch HTML-Code enthalten können, der bei der Visualisierung entsprechend formatiert wird und dabei z. B. auch *multimediale Dokumente* referenzieren kann. Mittels Preview-, History und Hypertext-Inhaltsverzeichnis stellt das KHS auch *Metainformation* zur Verfügung. Ebenso sind *Abstimmungen* innerhalb des Teilnehmer-Kreises und die Kennzeichnung von Beiträgen durch charakteristische Icons möglich ([Speck, Toussi 1998]). Zuletzt sei die Tatsache erwähnt, daß das KHS externe Informationssressourcen wie Email (Benachrichtigen von Benutzern/ Entgegennahme von Artikeln von Benutzern) oder externe Online-Systeme einzubinden vermag.

4.2 Systemeigenschaften, Technik und SW-Architektur

Das KHS besitzt folgende System-Eigenschaften:

- Die informationellen Einheiten (Knoten) des KHS-Hypertextes werden mittels typisierter Links semantisch kontrolliert vernetzt.
- Die Präsentation ist von der Struktur, beziehungsweise internen Repräsentation der informationellen Einheiten strikt getrennt.
- Zur Wartung der Inhalte dient ein Redaktionssystem.

[35] auch: “NetNews”-Groups genannt

Technik und Software-Architektur

Das KHS wurde 1998 von der Programmiersprache Smalltalk nach Java portiert und wird seither unter dem Projekt-Namen „KHS/J“ weiter entwickelt. Die eingesetzte Entwicklungsumgebung ist das Produkt „Visual Age 2.0“ von IBM (vgl. [IBM 1998]), das auf dem Java Development Kit (JDK) 2.0 basiert. Es setzt nun auf einem Servlet-basierten WWW-Server auf. Während die Vorgängerversion des KHS auf das objektorientierte DBMS Gemstone aufsetzte, ist die Anbindung des KHS/J an verschiedene DBMS (unter anderem über JDBC) noch in Arbeit.

Im Zusammenhang mit der Portierung wurde ein Re-Design vorgenommen, bei dem das von Smalltalk stammende Struktur Pattern *Model-View-Controller* (MVC)[36] noch stringenter angewandt wurde. Sämtliche KHS-Klassen sind so einer der MVC-Komponenten zugeordnet.

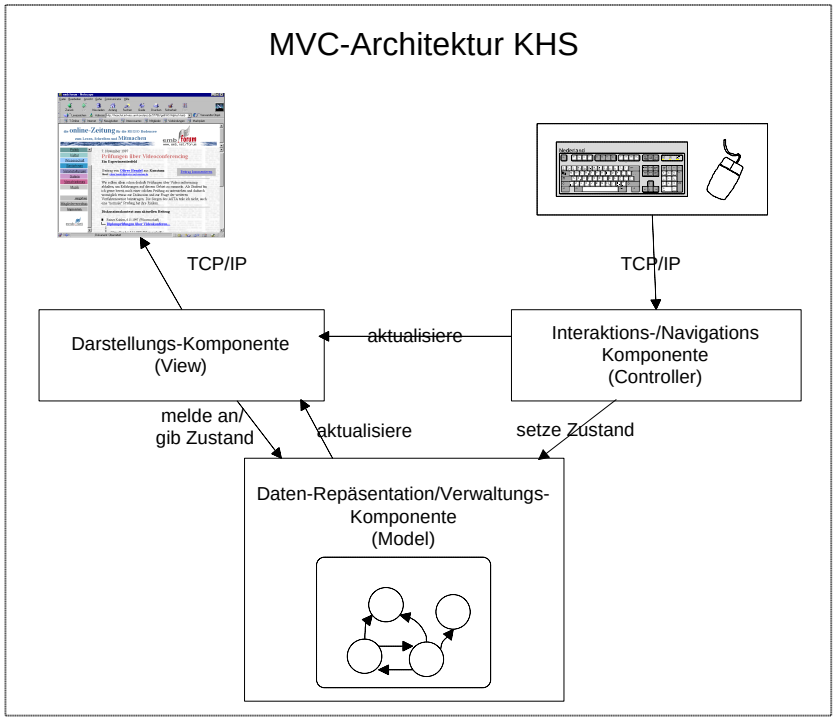


Abbildung 6: MVC-Architektur KHS, angelehnt an [Banner, Weitzel 1999][37]. Quelle: eigen

[36] [Gamma et al. 1996]

[37] Über die Darstellung der Funktionsweise der MVC gibt es unterschiedliche Ansichten. [ObjectArts 1999] sieht beispielsweise eine ausgeprägte Kommunikation zwischen Controller und View vor. Da die Sichtweise von [Bannert,

Erläuterung

Das Objekt "Model" ist das sogenannte Anwendungsobjekt. Der Controller verbindet die Benutzerschnittstelle mit dem Model. Er nimmt Benutzer-Eingaben entgegen und legt fest, wie auf diese reagiert wird. Das View-Objekt visualisiert die aus dem Model-Objekt angeforderten Daten. Folgende Design-Patterns sind in dieser Architektur enthalten (vgl. [Gamma et al. 1996]):

- Model-View: Beobachtermuster
- View-Controller: Strategiemuster
- Schachtelung von Views (falls mehrere vorhanden): Kompositionsmuster

4.2.1 Klassendiagramm

Sämtliche informationellen Einheiten des KHS-Hypertextes (Forumsbeiträge, Rubriken, Autoren …) werden als **Knoten des Hypertextes** aufgefaßt. Aus Modellierungs-Sicht spiegelt sich diese Tatsache darin, daß sämtliche Klassen, deren Instanzen informationelle Einheiten repräsentieren, Tochterklassen der Klasse *Node* sind. Analog werden alle Spezialisierungen von Links als Tochterklassen der Klasse *Link* implementiert (vgl. [Toussi 1999, Kap. 5.1 „KHS Hypertextmodell"]). Beide Kategorien, sowohl Links als auch Knoten, sind direkte Tochterklassen der Klasse *HypertextObject*. Neben Nodes und Links besitzen sogenannte Folder eine wichtige Rolle: Sie dienen der logischen Zusammenfassung einzelner Node bzw. Link Objekte. Aus Gründen der konsequenten Umsetzung des Hypertext-Paradigmas wird im KHS größtenteils auf Aggregationen komplexer Objekte[38] verzichtet.

Weitzel 1999] übereinstimmend in mehreren Literaturquellen zu finden ist, vgl. [Meyer 1998, S. 130; Buschmann et al. 1999, S. 129], wurde diese Darstellung gewählt.

[38] komplex, im Gegensatz zu simplen Objekttypen, wie Strings, Integer, boolean …

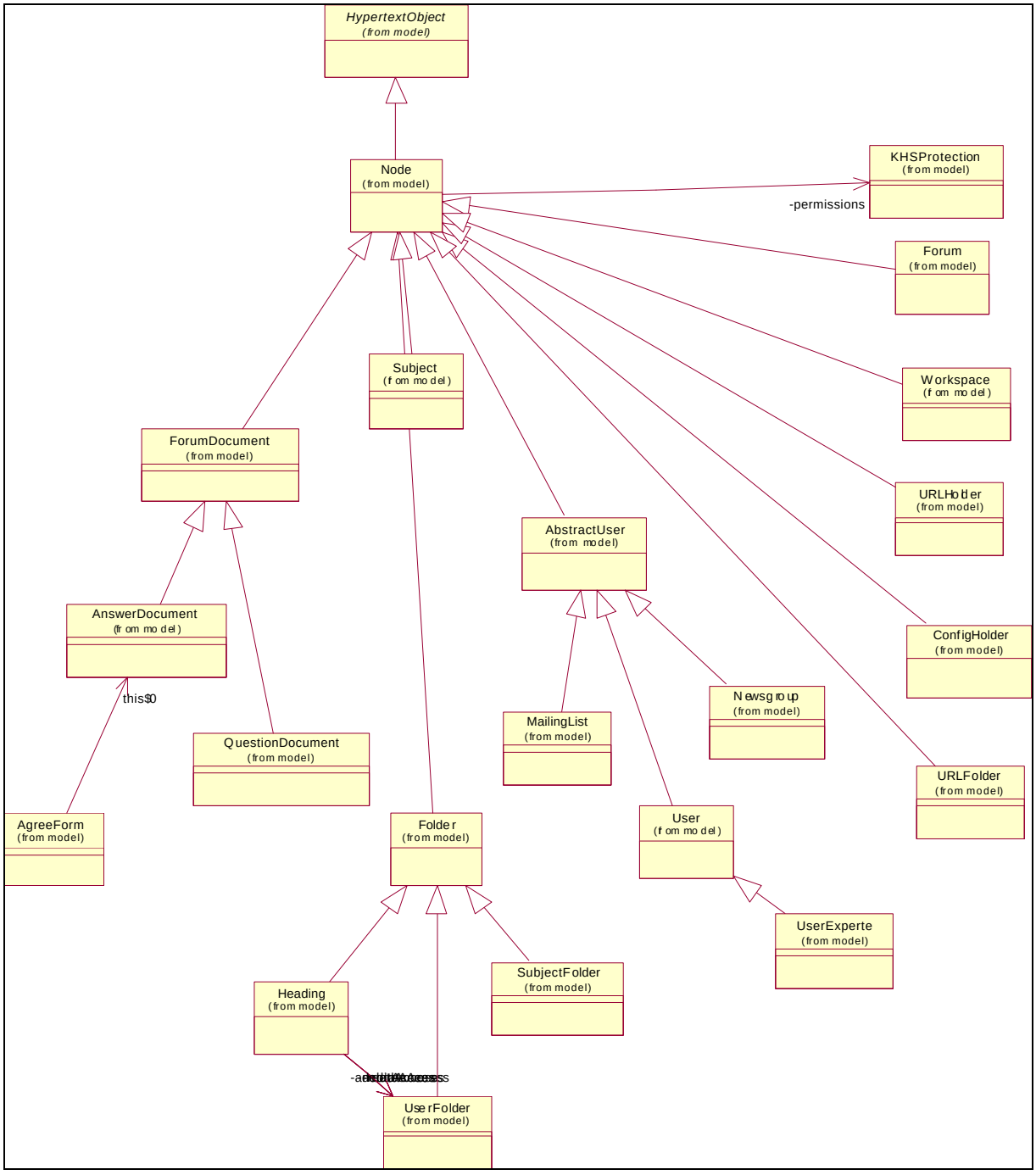


Abbildung 7: Auszug aus dem Klassendiagramm des KHS. Quelle: eigen

Künftige Entwicklungen

Wie bereits angesprochen, ist derzeit noch keine Datenbank-Anbindung des KHS/J implementiert. Für die Zukunft werden jedoch Anbindungen an DBMS (ORACLE 7), ebenso wie die Ausgabe in verschiedenen Druckformaten (LATEX, RTF, usw.) realisiert werden. Ferner gehört die Integration von XML-codierten Daten, wie durch die Themenstellung der vorliegenden Arbeit gezeigt, ebenfalls zu den Zielen der Entwicklung.

5 XML-Prozessoren - Standard APIs

Zur Verwendung von XML-Dokumenten in einem anwendungsspezifischen Kontext (Darstellung in einem Browser, Ablegen in einer Datenbank …) ist die formale Analyse von XML-Dokumenten notwendig. Typische Verarbeitungsschritte sind dabei die lexikalische Analyse, die syntaktische Analyse und die Prüfung auf Gültigkeit und Wohlgeformtheit[39]. Derartige Aufgaben, die beispielsweise auch zum Funktionsspektrum von Programmiersprachen-Compiler gehören[40], werden im Bereich der XML-Technologie von sogenannten XML-*Prozessoren*, oft auch als "XML-Parser" bezeichnet, durchgeführt.

Per offizieller Definition arbeitet ein XML-Prozessor mit einem zweiten Modul, der sogenannten *Applikation* zusammen [W3C 1998e, Kap. 1]. Daneben führt das W3C in [W3C 1998e] folgende Eigenschaften von XML-Prozessoren aus:

- Verarbeitung von Unicode in UTF-8 und UTF-16 Codierung[41] (Kap. 2.2),
- Ausgabe von (Parse-)Fehlern/definierte Reaktion auf Parse-Fehler (bspw. kein Fortsetzen der Verarbeitung bei sog. kritischen Fehlern (fatal errors)) (Kap. 2.2),
- Prüfung auf Wohlgeformtheit (Kap. 5.1),
- Prüfung auf Gültigkeit (optional) (Kap. 5.1).

Neben proprietären *Application Programming Interfaces* (API) zum Aufbau von XML-Prozessoren existieren derzeit zwei dominierende Standards auf diesem Gebiet:
die ereignis-gesteuerte *Simple API for XML* (SAX) und die baum-basierte API *Document Object Model* (DOM). Diese beiden, von Programmiersprachen und (HW/SW)-Plattformen unabhängigen, Standards werden in den nachfolgenden Kapiteln beschrieben.

[39] Die Erklärung dieser Begriffe findet sich im Anhang „Parser Technologie: Fachtermini".

[40] vgl. [Aho et al. 1990, S. 12 ff.]

[41] vgl. [UNICODE 1998] – zum Unterschied zwischen Zeichen-*Satz* und Zeichen-*Codierung* siehe [Maruyama et al. 1999, S. 24 „Note: Do not confuse character sets with character encodings"]

5.1 Simple API for XML (SAX)

5.1.1 Zielrichtung, Geschichte, Spezifikation, Verbreitung

Zielrichtung

Zielrichtung von SAX ist in erster Linie die Entwicklung von XML-Prozessoren, die zur Erfüllung ihrer Aufgabe mit einem Ausschnitt des Dokumentes auskommen. Zu diesen Aufgaben gehören vor allem *Konvertierungen*[42], die Ausgabe von *Statistiken* oder die Durchführung von *Suchvorgängen* in Dokumenten.

Geschichte

Die Schnittstelle " SAX - Simple API for XML" entstand im Rahmen eines Projektes einiger Teilnehmer der XML-DEV Mailing-List unter Koordinierung von David Megginson[43]. Das Projekt startete im Dezember 1997; das erste Release (1.0) wurde am 11. Mai 1998 herausgegeben. Die Idee der Entwickler war es, eine *einfache* Entwicklung von XML-Prozessoren zu erlauben.

Spezifikation

Im Gegensatz zum in Kapitel 5.2 erläuterten DOM-Standard ist SAX keine Spezifikation des W3C, sondern hat sich wegen seiner hohen Verbreitung (s. u.) als "de facto Standard" etabliert.

Verbreitung

Viele erhältliche Prozessor-API unterstützen per se den SAX Standard, bzw. es existieren *Treiber* für einige Produkte, die von Hause aus keinen SAX-Support bieten. Hinweise zu SAX-kompatiblen Produkten finden sich unter URL: http://www.megginson.com/SAX/applications.html. Die große Anzahl von SAX-kompatiblen Prozessoren legt nahe, SAX als einen *de facto Standard* im Bereich ereignis-basierter APIs zu bezeichnen.

---

[42] Z B. von XML nach HTML oder von einer XML-DTD in eine andere, die jedoch eine (fast) identische Schachtelungs-Struktur wie die ursprüngliche aufweisen muß.

[43] URL: http://www.megginson.com/SAX/history.html

5.1.2 Funktionsweise, Zugriff auf den Ableitungsbaum

Funktionsweise

SAX ist *ereignis-basiert*, das heißt: Während des Parsing Vorganges werden sogenannte Dokument-Ereignisse ausgelöst. Dies ist z. B. beim Erkennen von Tokens (wie z. B. Elementtypen, Elementinhalten, Attributen …) oder beim Auftreten von Parsing-Fehlern und Warnungen der Fall. Aufgefangen werden die Ereignisse mit Ereignis-Prozeduren, auch „call backs" oder „event-handler" genannt.Typische Ereignisse sind:

- Start, Ende eines Elementes/Dokumentes
- Behebbarer Fehler, Nicht-behebbarer Fehler, Warnung
- Auflösung einer externen Entität

Funktionsweise schematisch

Eine schematische Darstellung eines *nicht*-validierenden Parsing-Vorganges mittels der SAX-API findet sich in [Laurent 1999].

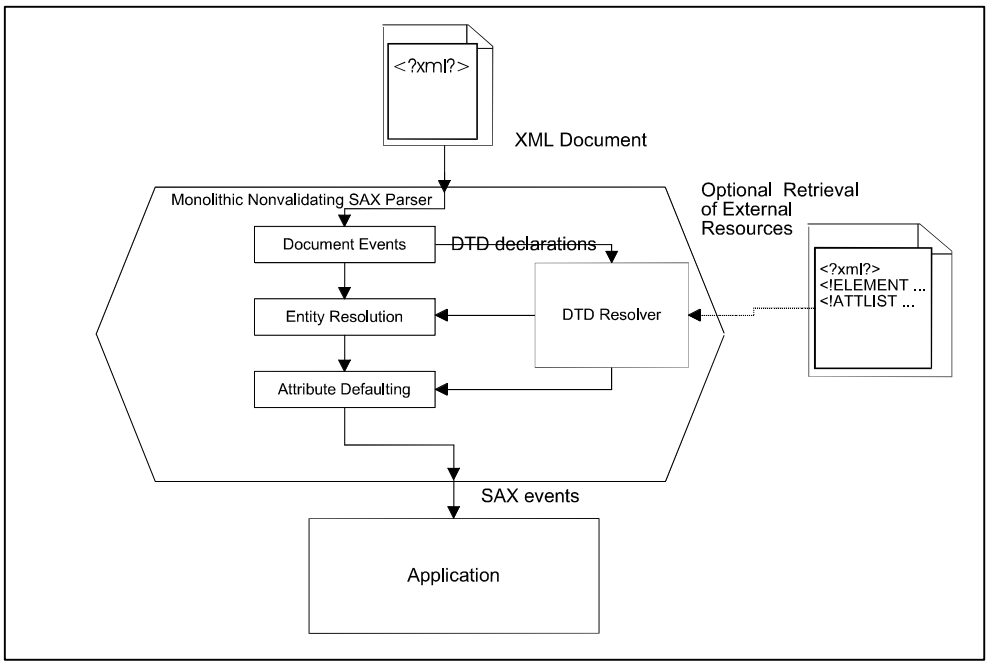


Abbildung 8: Nicht-validierender Parser auf Basis der SAX-API. Quelle [Laurent 1999]

Erklärung

Bei der Verarbeitung des Dokumentes werden die besagten Dokument-Ereignisse (Document Events) ausgelöst. DTD-Deklarationen (DTD Declarations) werden an den DTD-Resolver weitergereicht. Über die in der DTD enthaltenen Entitäts-Deklarationen werden die Entitäten (analog ß ü bei HTML) aufgelöst (Entity Resolution). Die Elemente des XML-Dokumentes erhalten aus der DTD die Standardwerte ihrer Attribute (Attribute Defaulting). Gegebenenfalls werden externe Dokumente (external Resources) wie z. B. eine externe DTD-Datei oder sogenannte externe Untermengen[44] herangezogen. Eine Validierung, also die Prüfung auf Gültigkeit, findet im gezeigten Typus *nicht* statt.
Die Parsing-Information wird in Form der ausgelösten Dokument-Ereignisse an das zweite Programm-Modul der (vollständigen) XML-Anwendung, die Applikation (Application), weitergereicht.

Zugriff auf den Ableitungsbaum

Der Zugriff auf die Struktur des Ableitungsbaumes nach Verarbeitung des Dokumentes, wie er beispielsweise bei den später erläuterten DOM-basierten Parsern möglich ist, wird von SAX systembedingt *nicht* unterstützt. Laurent und Cerami drücken dies folgendermaßen aus:

> "Event driven parsers are essentially stateless. Once a parsing event has been triggered, that event and its data are lost, unless your application explicitely saves it. [Laurent, Cerami 1999, S. 346 ff.]

David Megginson, der Initiator der SAX-API, spricht in diesem Zusammenhang von einem durch die ausgelösten Ereignisse „serialisierten" Ableitungsbaum[45]. Um mit Hilfe von SAX einen komfortablen Zugriff auf den Ableitungsbaum zu erhalten, kann der Entwickler zum einen auf freie Produkte wie z. B. das "Docuverse DOM SDK"[46] zurückgreifen, das mittels eines SAX-Parsers eine dom-kompatible Verwaltung des Ableitungsbaumes erzeugt. Zum anderen ist dem Entwickler natürlich auch eine selbst

---

([44] engl. external subsets)

[45] in einer Mail an den Autor vom 29. Juni 1999

[46] (vormals: SAXDOM) URL: http://www.docuverse.com/domsdk/index.html

entwickelte Verwaltung des Ableitungsbaums freigestellt. Dabei wird er zur Transformation des "serialisierten" Ableitungsbaumes in eine hierarchische Struktur in der Regel einen *Kellerautomaten*[47] einsetzen. Die konkrete Umsetzung eines solchen Verfahrens findet sich beispielsweise bei [Laurent, Cerami 1999, S 346 ff. "Building a SAX Tree Utility"]. SAX Parser sind übrigens im Allgemeinen nicht-validierend, das heißt, sie unterstützen normalerweise keine Prüfung auf die Gültigkeit von XML-Dokumenten.

5.1.3 Beispiel einer Verarbeitung, weitere Programmbeispiele

Als Test-Programm wurde ein SAX-basierter Prozessor geschrieben, der auf essentielle Parsing-Ereignisse (Start-/End-Element, Start-/End-Document, error, warning, usw.) reagiert. In den entsprechenden Ereignis-Prozeduren werden die Ereignis-Namen sowie die übergebenen Parameter (ElementName, Fehler-Code, Attribut-Namen und -Werte ...) ausgegeben. Verarbeitet man auf diese Art und Weise das aus dem Kapitel 3.2 bekannte XML-Dokument, das einen Auszug aus einem KHS-Forum repräsentiert, so erhält man folgendes Ergebnis.

```
Starting parse process ...
"Start Document" event ...

   "Start Element" event: ElementName ="HypertextData"
      "Start Element" event: ElementName ="Forum"
         Attribute Index[0] : Name="idForum"  Type="ID"  Value="x12345x"

         "Start Element" event: ElementName ="Name"
            "Characters" event: Test-Forum
         "End Element" event: ElementName ="Name"
         "Start Element" event: ElementName ="Text"
            "Characters" event: Willkommen im Testforum.
         "End Element" event: ElementName ="Text"
      "End Element" event: ElementName ="Forum"

      "Start Element" event: ElementName ="Heading"
         Attribute Index[0] : Name="idHeading"  Type="ID"  Value="x000x"

         "Start Element" event: ElementName ="Name"
            "Characters" event: Rubrik aus XML-File
         "End Element" event: ElementName ="Name"

         "Start Element" event: ElementName ="Text"
```

[47] engl. stack machine

```
            "Characters" event: Text zur Rubrik aus XML-File
         "End Element" event: ElementName ="Text"
      "End Element" event: ElementName ="Heading"

      "Start Element" event: ElementName ="HierarchyLink"
         Attribute Index[0] : Name="idrefFrom"  Type="IDREF"  Value="x12345x"
         Attribute Index[1] : Name="idrefTo"  Type="IDREF"  Value="x000x"
      "End Element" event: ElementName ="HierarchyLink"
   ...

   "End Element" event: ElementName ="HypertextData"
"End Document" event
```

Anmerkung

Zeilen die mit **"...." event:** eingeleitet werden, stehen für ein ausgelöstes Ereignis. Zwischen den doppelten Hochkommata steht der Name des Ereignisses. Zeilen, die auf eine “Ereignis-Zeile” folgen und nicht in dieser Form eingeleitet werden, geben Parameter, die zu diesem Ereignis gehören, wieder. Beispiel:

```
"Start Element" event: ElementName ="Forum"
         Attribute  Index[0]  :  Name="idForum"    Type="ID"    Value="x12345x"
```

Erklärung: Ein “Start-Element” Ereignis wurde ausgelöst. Der Name des Elementes ist Forum. Das erste Attribut hat den Namen “idForum”, ist vom Typ “ID” und hat den Wert "x12345x".

Weitere Programmbeispiele

Als weitere Programmbeispiele für den Einsatz von SAX-Parsern findet sich ein HTML-Konverter bei [Middendorf 1999] sowie James Coopers “Humble XMLstats program” (in Perl) unter http://www.xml.com/xml/pub/98/09/xml-perl.html.

5.1.4 Schnittstellen-Beschreibung und -Hierarchie

Schnittstellen-Beschreibung

SAX bietet insgesamt sieben Schnittstellen, davon vier, die Event-Handler für unterschiedliche Kategorien von Ereignissen bieten. Weitere drei Schnittstellen stellen Hilfsmittel zur Verfügung.

a) Schnittstellen zum Auffangen von Dokument-Ereignissen

Interface-Name	Ereignis-Kategorie	Beispiel für Ereignisse
`DocumentHandler`	Allgemeine Dokument-Ereignisse	Start/Ende eines Elementes/des Dokumentes, Zeichenfolge aufgetreten, …
`DTDHandler`	DTD-Ereignisse	Deklaration einer nicht zu parsenden Entität (unparsed entity), …
`EntityResolver`	Ereignisse zur Auflösung von Entitäten	Auflösung einer externen Entität
`ErrorHandler`	Fehler/Warnung- Ereignisse	Auftreten eines behebbaren Fehlers (error)/eines nicht behebbaren Fehlers (fatal error)[48]/einer Warnung (warning)

Abbildung 9: SAX-API: Schnittstellen zum Auffangen von Dokument-Ereignissen[49]

b) Hilfs-Schnittstellen

Interface-Name	Aufgabe
`Parser`	Bereitstellung der Parse-Funktionalität (Instanziierung des Parsers und Verarbeitung des Dokumentes)
`AttributeList`	Zugriff auf der Attribute eines Elementes
`Locator`	Zugriff auf die Positionen innerhalb XML-Dokumenten (Zeilennr, Spaltennr …) auf die sich die ausgelösten Ereignisse beziehen

Abbildung 10: SAX API: Hilfs-Schnittstellen

Die Schnittstellen –Hierarchie wird von der Abbildung auf der folgende Seite illustriert.

[48] die Begriffe error/ fatal error sind präzise in [W3C 1998e] in Abschnitt 1.2 definiert

[49] ,(extrahiert aus [IBM 1999a])

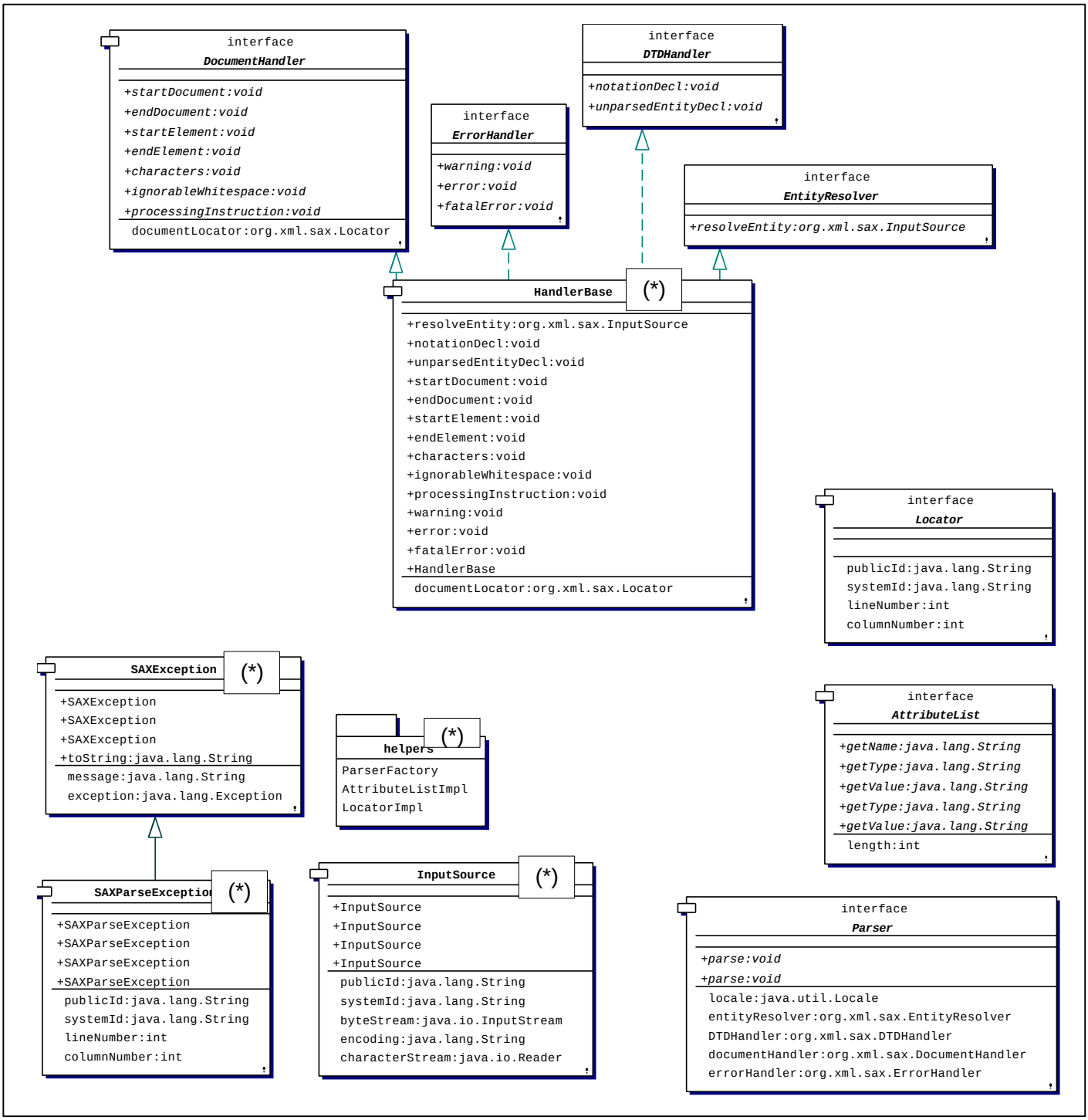


Abbildung 11: SAX-Schnittstellen Hierarchie[50]

Anmerkung: Die mit (*) gekennzeichneten Klassen/Pakete sind *zusätzliche* Bestandteile der SAX-Implementierung des IBM-Prozessors XML4J.

[50] (extrahiert aus [IBM 1999a])

5.2 Document Object Model (DOM)

5.2.1 Zielrichtung, Geschichte, Spezifikation, Verbreitung

Zielrichtung

Die Grundidee des *Document Object Model* (DOM) ist es, die Baum-Struktur von XML-Dokumenten mittels einer *standardisierten* Schnittstelle zugänglich zu machen. Dies ist zum einen im Falle der Analyse von XML-Dokumenten durch XML-Prozessoren, jedoch auch bei der Scriptprogrammierung[51] in WEB-Browsern von Bedeutung. Im Falle eines XML-Prozessors entspricht die Baumstruktur der Dokument-Bestandteile einer *Repräsentation des Ableitungsbaumes*.

Laut W3C-Spezifikation [W3C 1998d, Kap. „What the DOM is“] beschreibt das DOM:

- „the interfaces and objects used to represent and manipulate a document,
- the semantics of these interfaces and objects - including both behavior and attributes,
- the relationships and collaborations among these interfaces and objects“.

Obwohl die Bezeichnung “Model” auf den ersten Blick vermuten läßt, es handle sich dabei um ein Objektmodell analog dem Component Object Model-Standard der Firma Microsoft, ist es in Wirklichkeit eine API, also (Programmier-) *Schnittstelle*. Eine Tatsache, die von der W3C DOM Recommendation folgendermaßen beschrieben wird [W3C 1998d, Kap. „What the DOM is not“]:

> „The Document Object Model, despite its name, is not a competitor to the Component Object Model (COM). COM, like CORBA, is a language independent way to specify interfaces and objects; the DOM is a set of interfaces and objects designed for managing HTML and XML documents. The DOM may be implemented using language-independent systems like COM or CORBA; it may also be implemented using language-specific bindings like the Java or ECMAScript bindings specified in this document“.

[51] mit JavaScript, VBScript

In welcher Form die Objekte hinter der "API-Fassade" miteinander kommunizieren, oder ob die interne Datenstruktur überhaupt in Form von Objekten implementiert ist[52], ist nicht festgelegt. Dies bleibt den Herstellern der DOM-kompatibler Produkte überlassen.

Geschichte

Das ursprüngliche Einsatzgebiet des DOM war das Gebiet (client side) Script-Programmierung in WEB-Browsern analog des von Dynamic HTML (DHTML) bekannten Prinzips. Durch die Verwendung von DHTML können Scriptsprachen auf die Elemente einer HTML-Seite definiert zugreifen und deren Inhalt manipulieren (umbenennen, verschieben, löschen, erweitern) oder aufbereiten (z. B. durch Erzeugung eines Diagrammes).

Die Problematik bei DHTML liegt darin, daß die entsprechenden Scriptsprachen wie JavaScript und VB-Script auf *proprietäre* Schnittstellen von WEB-Browsern zugreifen müssen [Hueskes 1997; Koch 1999, S. 269, 299]. Deshalb muß ein und dieselbe Script-Funktionalität in *mehreren* Codierungen für die unterschiedlichen Browser-Typen verfaßt werden. Ein WEB-Server muß also prüfen, bevor er ein Script an den Client sendet, welcher Browser client-seitig vorliegt, um die passende Script-*Codierung* auszuwählen. Hier wird mit dem DOM-Ansatz Abhilfe geschaffen. Das DOM bietet eine *einheitliche,* vom W3C genormte Schnittstelle für Scriptprogrammierung von *HTML-Dokumenten*.

Neben HTML Dokumenten werden auch *XML Dokumente* im DOM berücksichtigt. Dabei gliedert sich das DOM in zwei Teile[W3C 1998d] : *DOM Core*, das Funktionalität sowohl für XML - als auch HTML-Dokumente zur Verfügung stellt und *DOM HTML*, das spezifische Funktionalität für den Zugriff auf HTML Dokumente bietet.

Spezifiziert wird die Schnittstelle DOM mittels der Interface Definition Language (IDL) der Object Management Group (OMG), wie sie in der CORBA 2.2 Spezifikation festgelegt ist. Schnittstellen-Spezifikationen, die auf spezielle Anwendungssprachen wie

[52] Im Falle von nicht-objektorientierten Programmiersprachen wie C wird man die DOM-"Objekte" in gewöhnlichen Structs, die über Zeiger verknüpft sind, ablegen.

z. B. Java, ECMA Script angepaßt sind, sogenannte „language bindings“, werden ebenfalls angeboten.

Ergänzend sei bemerkt, daß - laut Norm - eine DOM-Anwendung auch dann als DOM-kompatibel gilt, wenn sie Zusatzfunktionalität, also Schnittstellen-Implementierungen aufweist, die über das DOM hinausgehen[53]. Das DOM selbst stellt somit den kleinsten gemeinsamen Nenner aller DOM basierten Anwendungen dar.

Die Recommendation des W3C legt derzeit den *level 1* des DOM fest. Zukünftige Versionen des DOM sollen laut dieser Spezifikation folgende Zusatzfunktionalitäten bieten:

- Ein Struktur-Modell für die internen und externen Untermengen,
- Validierung in bezug auf ein Schema,
- Steuerung der Formatierung von Dokumenten mittels Formatvorlagen (“style sheets”),
- Zugriffs Kontrolle,
- Thread-Sicherheit.

Quelle: übersetzt aus [W3C 1998d, Kap. “Limitations of Level 1“]

Spezifikation

Im Gegensatz zum SAX Standard handelt es sich beim DOM nicht etwa um einen “de facto Standard” sondern um eine vom W3C verabschiedete Recommendation [W3C 1998d].

Verbreitung

Parser-Produkte die das Document Object Model unterstützen, sind aufgrund ihrer Komplexität seltener als Produkte, die die SAX-API unterstützen. Dem Autor sind aufgrund seiner Recherchen lediglich die folgenden drei Produkte bekannt geworden:

- IBM - XML4Java (http://www.alphaworks.ibm.com/tech/xml)
- Silfide Technology - SXP (http://www.loria.fr/projets/XSilfide/EN/sxp/)

[53] [W3C 1998d, Kap.“What the Document Object Model is not – DOM Interfaces and DOM Implementations“]

- Javasoft Java Project X – (http://developer.javasoft.com/developer/earlyAccess/xml/index.html[54])

Es muß jedoch darauf hingewiesen werden, daß - wie auf Seite 43 beschrieben - mittels des Docuverse DOM SDK eine DOM-API in sax-kompatible Parser integriert werden kann.

5.2.2 Funktionsweise, Zugriff auf den Ableitungsbaum

Die Funktionsweise eines entsprechenden, DOM-basierten Prozessors (hier die validierende Version) ist in [Laurent 1999] illustriert: Nachfolgende Seite zeigt das entsprechende Schaubild.

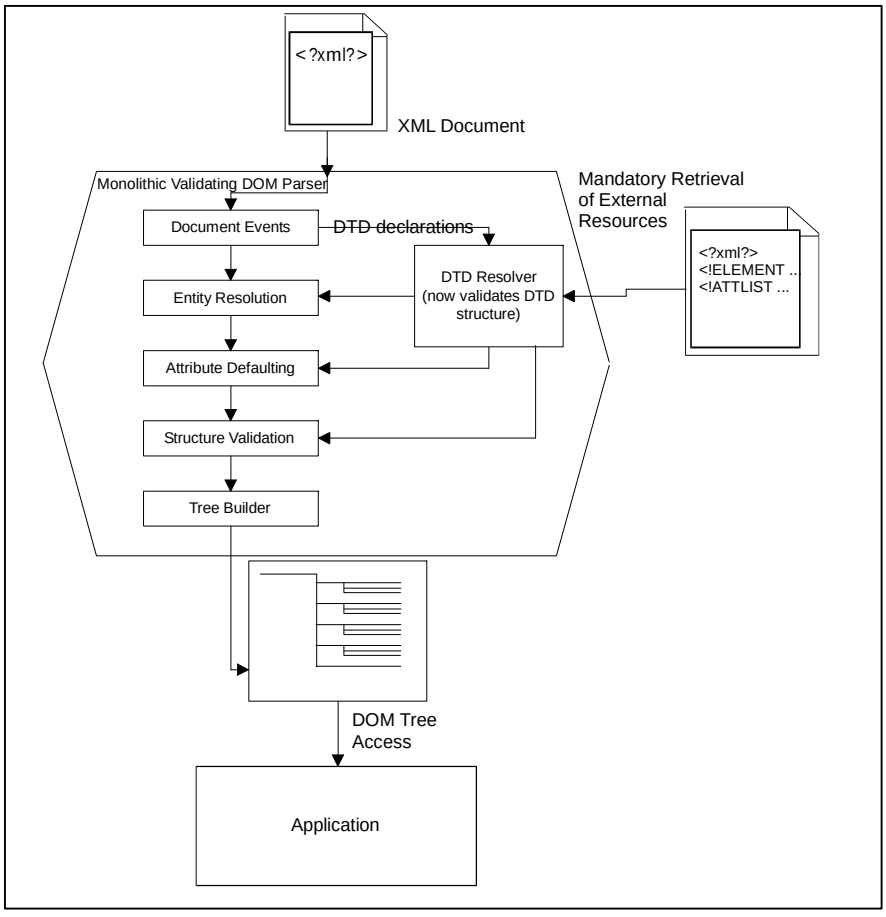


Abbildung 12: Validierender Parser auf Basis der DOM-API. Quelle [Laurent 1999]

[54] Zugriff nur für registrierte Benutzer. (Stand URL-Angaben: 23.8.1999)

Erklärung

Die ersten Stufen der Verarbeitung (Auslösen der Dokument-Ereignisse, Entitäten-Ausflösung …) werden analog des auf Seite 42 beschrieben Beispiel des nicht-validierenden SAX-Prozessors ausgeführt. Hinzu kommt hier die Validierung des Dokumentes anhand der DTD (Validating DTD Structure). Desweiteren wird aus den ausgelösten Ereignissen der Ableitungsbaum erzeugt (Tree Builder). Als Schnittstelle zwischen Prozessor und Applikation dienen die vom W3C festgelegten Schnittstellen-definitionen für den Ableitungsbaum-Zugriff.

Zugriff auf den Ableitungsbaum

Das DOM spezifiziert die Repräsentation des Ableitungsbaumes durch eine Vielzahl von Schnittstellen (insgesamt 18), die meistens einen Bestandteil des XML-Dokumentes (Element, Attribut, Kommentar, Entität ...) darstellen. Die Dokument-Bestandteile (=Knoten des Ableitungsbaumes) sind eine Spezialisierung der zentralen Schnittstelle „Node“. Zusätzliche Schnittstellen wie z. B. die Schnittstelle „Nodelist“ (Liste zur Aufnahme von Knoten), dienen im allgemeinen der *Verwaltung* von Dokument-Bestandteilen.

Über entsprechende Kapselungsmethoden wird auf die Eigenschaften der Bestandteile („Objekt-Attribute“) wie z. B. Typ, Name, zugegriffen. Für den Zugriff auf „verwandte“ Objekte[55] sind ebenfalls Methoden vorhanden.

Dabei können die Daten des Ableitungsbaumes nicht nur gelesen, sondern auch *modifiziert* oder *gelöscht* werden. Eine weitere interessante Funktionalität, welche in Verbindung mit dem DOM möglich ist, stellt die *Integration von Anwendungslogik* in den Ableitungsbaum dar. Hierzu werden intelligente Objekte, sogenannte XML-Beans, in den Ableitungsbaum eingebettet[56] (vgl. [Middendorf 1999, S. 48]).

[55] (Eltern-Objekte = hierarchisch übergeordnet, Kind-Objekte = hierarchisch untergeordnet, Geschwister-Objekte = hierarchisch gleichgestellt)

[56] salopp gesprochen: „eingehängt“

5.2.3 Beispiel einer Verarbeitung, weitere Programmbeispiele

Verarbeitet man das im Kapitel 3.2 vorgestellte XML-Dokument, so erhält man folgende Repräsentation in Form von DOM-Objekten[57] der geparseden XML Elemente und deren Attribute.

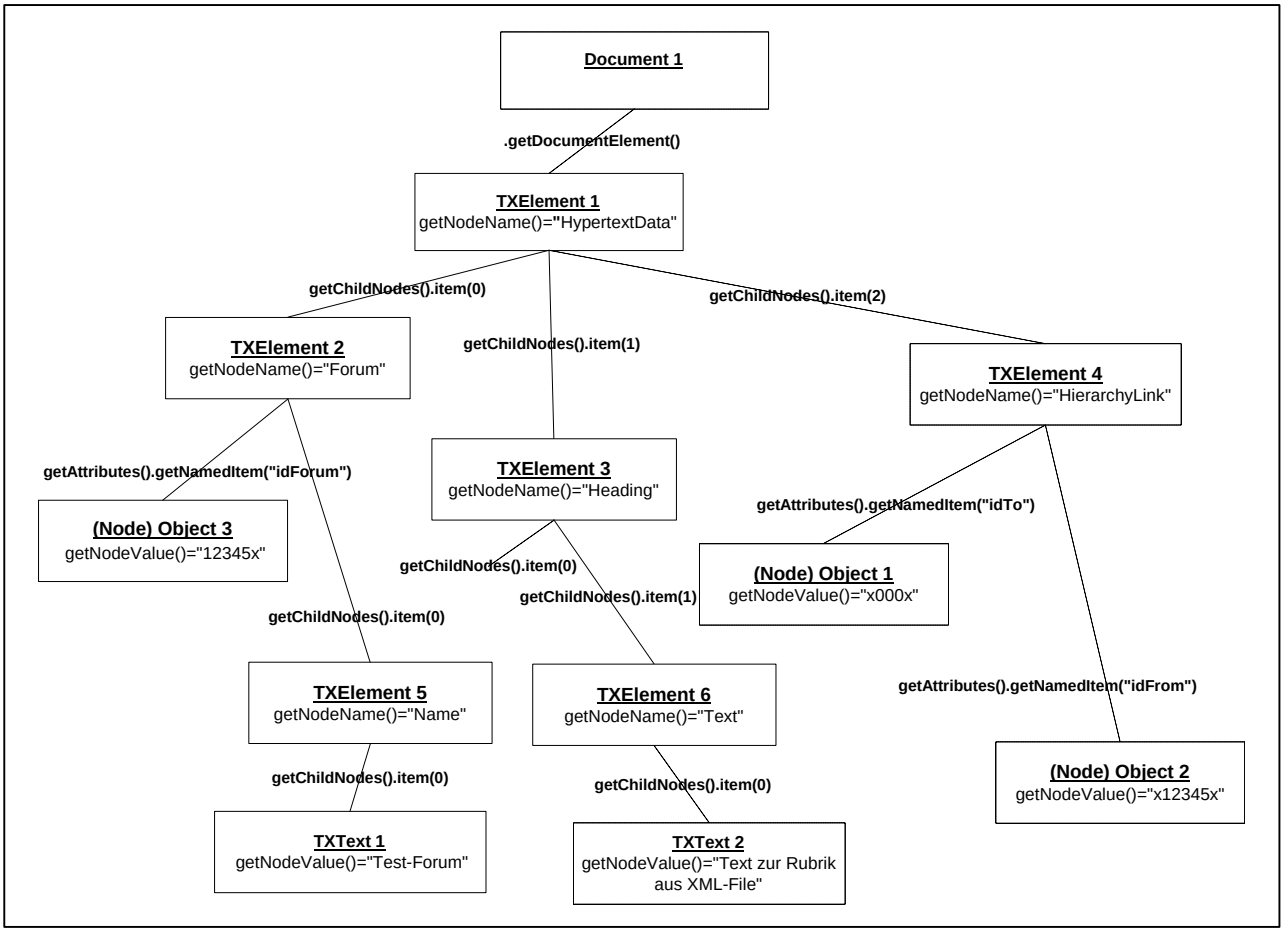


Abbildung 13: Objekte in einem DOM-Baum (transf. Ausschnitt eines XML-Dokumentes). Quelle: eigen

[57] Wie bereits eingangs erwähnt, wird die DOM-Datenstruktur intern nicht zwingend in Form von Objekten repräsentiert. Während dies im Falle von OO-Programmiersprachen in der Regel tatsächlich der Fall sein dürfte, verhält es sich im Falle klassischer Programmiersprachen wie z. B. „C“ so, daß die API nach außen den „Eindruck erweckt”, als lägen intern hierarchisch angeordnete Objekte vor.

Legende

Es wurde die UML-Notation zur Darstellung der Instanzen verwendet. Die Überschriften in den Instanzen entsprechen den (proprietären) Klassennamen des IBM Xml4J, die fast gleich lautende DOM-Schnittstellen implementieren[58]. Type-Casts wurden in Klammer gesetzt. Die Beschriftungen der Kanten entsprechen den DOM-API Aufrufen, die als Ergebnis die untergeordnete Instanz liefern. Zu den Instanzen gehörige Attribute werden über die zugehörigen Kapselungsmethoden (getXY()) spezifiziert.

Weitere Programmbeispiele

finden sich bei [Maruyama et. al. 1999] und in der Dokumentation des Parsers XML4J(ava) ([IBM 1999a]). Scriptprogrammierungs-Beispiele unter zuhilfenahme des DOM findet man unter Microsofts „XML Tutorial Lesson 4: Using the XML Object Model“[59].

5.2.4 Schnittstellendefinition und Hierarchie

Im folgenden werden sämtliche (=18) Schnittstellen des DOM tabellarisch vorgestellt. In der rechten Spalte wird Bezug auf die DOM-Implementierung des Prozessors XML4J von IBM genommen. Hier sind Klassen aufgeführt, welche die entsprechende DOM-Schnittstelle implementieren.

Tabelle 1: Schnittstellen des DOM Quelle: übersetzt aus [Maruyama et al. 1999, S. 46]

Schnittstellen Name	Beschreibung	Implementierungsklasse in XML for Java
`Node`	Der primäre Daten-Typ. Repräsentiert einen einzelnen Knoten im Dokument-Baum	`Child` oder `Parent`
`Document`	Das gesamte XML Dokument	`TXDocument`
`Element`	Ein Element und alle enthaltenen Knoten	`TXElement`
`Attr`	Ein Attribut in einem Element-Objekt	`TXAttribute`

[58] (Bsp. Klasse TXElement implementiert das DOM-Interface Element)

[59] (http://msdn.microsoft.com/xml/tutorial/use_om.asp, Stand 16.8.1999)

Fortsetzung von Tabelle 1: Schnittstellen des DOM

`ProcessingInstruction`	Eine [XML-]Prozessor-Anweisung	`TXPI`
`CDATASection`	CDATA Bereich[60]	`TXCDATASection`
`DocumentFragment`	Ein „Leichtgewicht" Dokument – wird benutzt, um mehrere Unter-Bäume (engl. subtrees) oder Teile von Dokumenten zu repräsentieren	`TXDocumentFragment`
`Entity`	Eine Entität, geparsed oder ungeparsed, in einem `DocumentType` Objekt	`EntityDecl.EntityImpl`[61]
`EntityReference`	Eine Entitäten-Referenz, wie sie innerhalb eines Dokument-Baumes auftritt	`GeneralReference`
`DocumentType`	Eine DTD, die eine Liste von Entitäten enthält	`DTD`
`Notation`	Eine Notation[62], die in einer DTD deklariert wird (eine Notation deklariert über ihren Namen das Format einer ungeparseden Entität)	`TYNotation.NotationImpl`
`CharacterData`	Ein Eltern-Interface von `Text` und anderer, welche Operationen wie Einfügen einer Zeichenfolge oder das Löschen einer Zeichenfolge benötigen	`TXCharacterData`
`Comment`	Ein Kommentar	`TXComment`
`Text`	Text	`TXText`
`DOMException`	Eine Exception, die ausgelöst wird, wenn keine weitere Verarbeitung mehr möglich ist; normale Fehler werden über Rückgabe-Werte gemeldet. Beachte: Dies ist eine abstrakte Klasse.	`TXDOMException`
`DOMImplementation`	Ein Platzhalter für Methoden die unabhängig von einer spezifischen DOM- Implementierung sind.	Nicht als eigene Klasse implementiert![63]
`NodeList`	Eine geordnete Sammlung (engl. ordered Collection) von Knoten; Elemente in der NodeList können über einen integralen Index (Start bei 0) angesprochen werden	Nicht als eigene Klasse implementiert!
`NamedNodeMap`	Eine Sammlung (engl. Collection) von Knoten, die über ihren Namen angesprochen werden können.	Nicht als eigene Klasse implementiert!

[60] [geschützter Bereich; vgl. [W3C 1998e, Kap. 2.7]]

[61] [Schreibweise `EntityDecl.EntityImpl` bedeutet: `EntityImpl` ist eine innerhalb von `EntityDecl` definierte Klasse („inner Class")]

[62] [vgl. [W3C 1998e, Kap. 4.7]]

[63] [Nach einer Plausibilitätsprüfung wurde die Interpretation „Nicht als eigene Klasse implementiert!" als Äquivalent des „NA" im englischen Original angesehen].

Als Beispiel für eine echte DOM-*Schnittstellendefinition* wird nachfolgend die Definition der Schnittstelle „Node“ angeführt. Diese *zentrale* Schnittstelle der DOM-API ist das Eltern-Interface sämtlicher Schnittstellen, welche Knoten des Objekt-Baumes (d. h. Elemente, Elementinhalte, Attribute usw.) repräsentieren. Die abgedruckte Spezifikation ist in der *Java*-IDL abgefaßt, stellt also ein *Java Language Binding* dar.

```
interface Node { // NodeType
const unsigned short ELEMENT_NODE = 1;
const unsigned short ATTRIBUTE_NODE = 2;
const unsigned short TEXT_NODE = 3;
const unsigned short CDATA_SECTION_NODE = 4;
const unsigned short ENTITY_REFERENCE_NODE = 5;
const unsigned short ENTITY_NODE = 6;
const unsigned short PROCESSING_INSTRUCTION_NODE = 7;
const unsigned short COMMENT_NODE = 8;
const unsigned short DOCUMENT_TYPE_NODE = 10;
const unsigned short DOCUMENT_FRAGMENT_NODE = 11;
const unsigned short NOTATION_NODE = 12;
readonly attribute DOMString nodeName;
attribute DOMString nodeValue;
// raises(DOMException) on setting
// raises(DOMException) on retrieval
readonly attribute unsigned short nodeType;
readonly attribute Node parentNode;
readonly attribute NodeList childNodes;
readonly attribute Node firstChild;
readonly attribute Node lastChild;
readonly attribute Node previousSibling;
readonly attribute Node nextSibling;
readonly attribute NamedNodeMap attributes;
readonly attribute Document ownerDocument;
Node insertBefore(in Node newChild,
in Node refChild)
raises(DOMException);
Node replaceChild(in Node newChild,
in Node oldChild)
raises(DOMException);
Node removeChild(in Node oldChild)
raises(DOMException);
Node appendChild(in Node newChild)
raises(DOMException);
boolean hasChildNodes();
Node cloneNode(in boolean deep);
};
```

(Auszug aus [W3C 1998d, Appendix D:Java Language Binding])

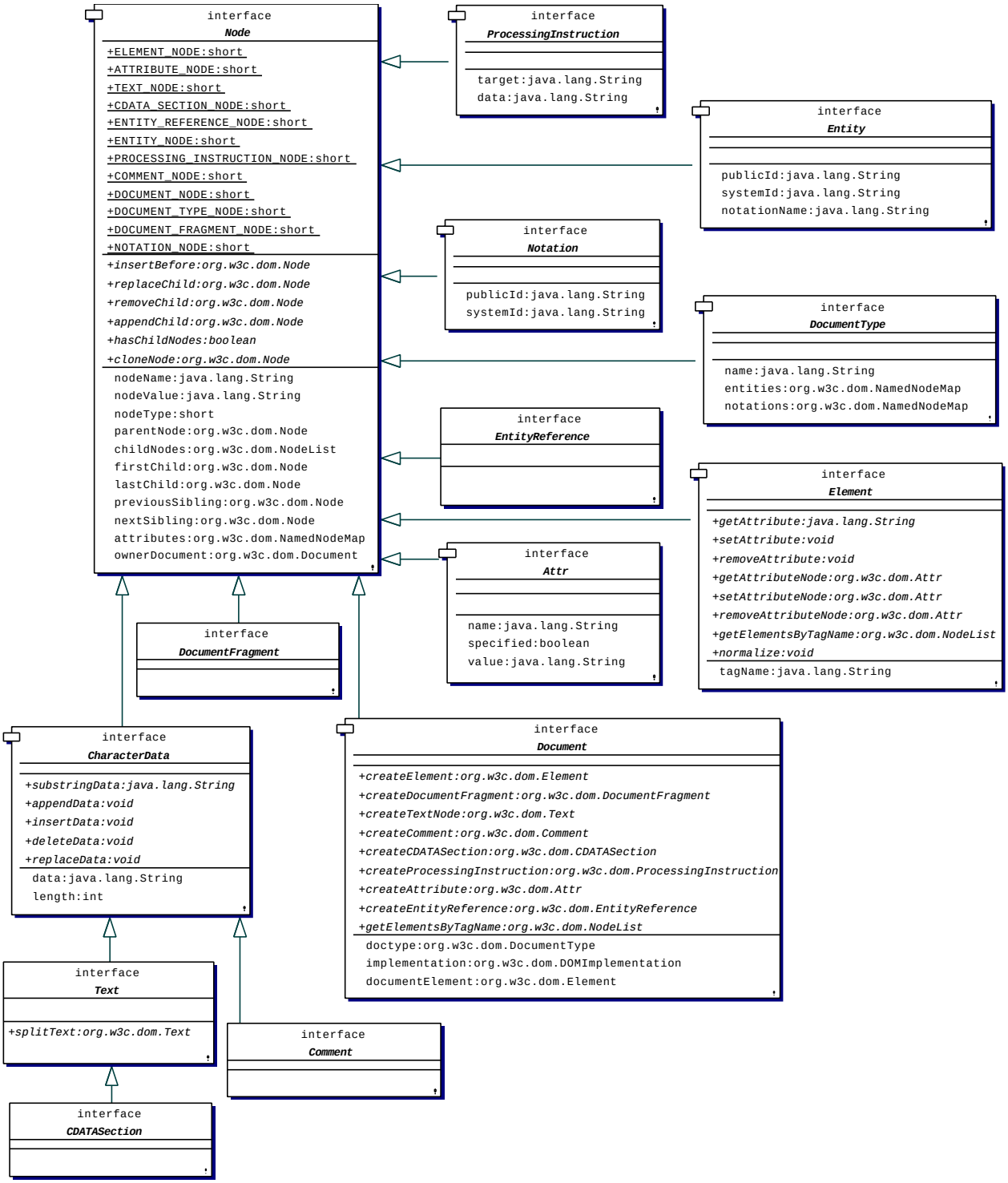


Abbildung 14: Schnittstellenhierarchie des DOM , Teil I Quelle: extrahiert aus [IBM 1999a]

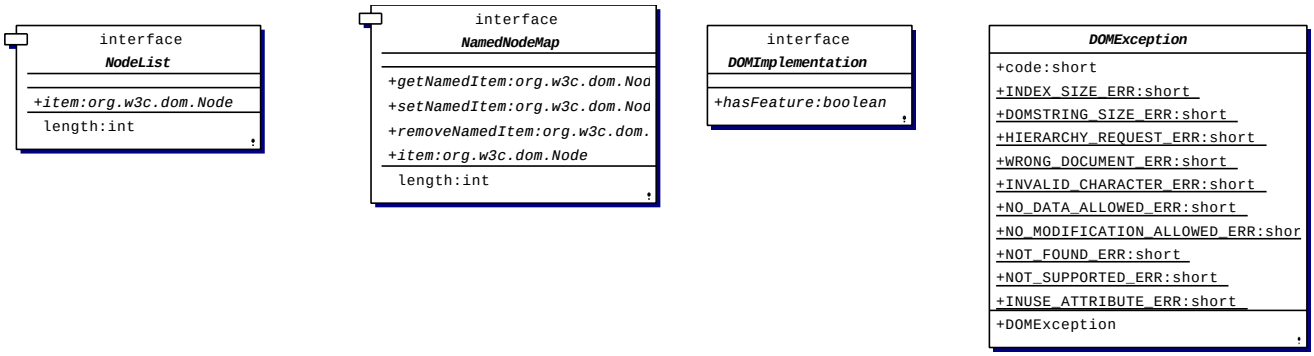


Abbildung 15: Schnittstellenhierarchie des DOM , Teil II extrahiert aus [IBM 1999a]

5.3 Bewertung der beiden API-Standards

5.3.1 Grundlegende Bewertung

Der Hauptvorteil von SAX besteht im schonenden **Umgang mit den Ressourcen** des Computersystems. Da der Fokus des Parsers lediglich die aktuell zu bearbeitende Zeichenfolge berücksichtigt, müssen vorangegangene Parsing-Ergebnisse eines Dokumentes nicht im Speicher gehalten werden, abgesehen von kleinen Ausnahmen, z. B. der Strukturanalyse im Falle eines Validerungsvorganges. Dadurch es leicht möglich, Dokumente zu verarbeiten, die wesentlich größer sind als der Hauptspeicher des Computers.
Demgegenüber erweist sich das DOM als wahrer Ressourcenfresser. Hier wird der Ableitungsbaum *zusätzlich* zum bearbeitenden Dokument vollständig im Hauptspeicher (respektive swap space) des Computers gehalten. Entsprechend kann man etwa von der doppelten Dokumentgröße als Hauptspeicherbedarf ausgehen.

Als äußerst mächtig erweist sich das DOM hingegen beim **Zugriff auf den Ableitungsbaum**. Dieser steht – über die im DOM Standard festgelegten Schnittstellen – einer entsprechenden Weiterverarbeitung zur Verfügung. Mittels des DOM Elemente ist nicht nur ein Lesezugriff auf den Ableitungsbaumes möglich, sondern auch die Manipulation desselben. Ein weiteres Feature ist die *Einbettung von Anwendungslogik* durch XML-Beans in den Ableitungsbaum.

In dieser Hinsicht ist die SAX-API spartanisch, der Parsebaum wird während des Parsing in Form von Dokumentereignissen seriell ausgegeben. Ein Zugriff auf die Baumstruktur nach Beendigung des Parse-Vorganges wird nicht geboten.

In Bezug auf den **Einarbeitungsaufwand** besitzt das SAX gegenüber dem DOM leichte Vorteile; dies ist alleine durch den vorhanden Umfang gegeben (7 Schnittstellen gegenüber 18).

Der **Grad der Standardisierung** ist ein weiterer Vorteil des DOM. Dieses ist vom W3C durch eine Recommendation festgelegt – SAX hingegen ist ein „de facto Standard“. Es ist somit wahrscheinlich, daß DOM-kompatible Werkzeuge noch einige Jahre gepflegt werden, während sich sax-basierte Tools zur „Eintagsfliege” entpuppen könnten.

5.3.2 Bewertung im Kontext des vorliegenden Projektes

Aufgrund der personellen Situation (Teamgröße < 10 Mitarbeiter/innen) im Rahmen des KHS/J Projektes ist es zwingend, den Implementierungsaufwand so gering wie möglich zu halten und besonders Tendenzen “das Rad neu zu erfinden” sind zu vermeiden. Anpassungen sind somit Neu-Entwicklungen vorzuziehen.
Für die komplexe Integration in das KHS ist es effizient, auf den Ableitungsbaum auch nach Beendigung des Parse-Vorganges zugreifen zu können. Da dom-basierte Parser per se diesen Zugriff auf den Ableitungsbaum bieten (auch schreibend!), sind sie gegenüber den ereignis-basierten Parsern eindeutig im Vorteil. Die weiteren Vorzüge des DOM wie z. B. die Möglichkeit, Anwendungslogik in Form von XML-Beans einzubetten sowie die Standardiserung durch das W3C, sind für Weiterentwicklungen des KHS ebenfalls von Bedeutung.
Deshalb fiel die *Entscheidung zugunsten eines DOM-basierten Parsers*. Der einzige schwerwiegende Nachteil des DOMs, bestehend in dem hohen Ressourcenbedarf des verarbeitenden Computersystems, kann gegebenenfalls durch die Beschaffung von zusätzlichem Hauptspeicher bei verhältnismäßig geringem Aufwand ausgeglichen werden[64].

[64] (Preis für 256 Megabyte SD-RAM im Juni 1999: ca. DM 200.-)

5.4 Produkt-Evaluation

Wie im vorangegangenen Kapitel im Falle der Grundsatzentscheidung zugunsten eines DOM- oder SAX-unterstützenden Parsers, standen auch bei den anderen Aspekten der Produktauswahl die *Effzienzvorteile bei der Entwicklung* im Vordergrund.

Beispielsweise fiel die Wahl der Programmiersprache auf **Java**, um die Integration in das KHS/ J(ava) möglichst einfach zu gestalten. Diese Entscheidung wird übrigens auch durch die aktuellen Parser-Entwicklungen gerechtfertigt: Wie man im Kapitel 5.6 „Exkurs: XML und Java“ erfahren kann, erweist sich auch in diesem Zusammenhang die Portierung des KHS in die Programmiersprache Java als positiver Faktor.

Darüber hinaus sollte der gesuchte Parser einen möglichst großen Funktionsumfang (Validierung, etc.) bieten und lizenzfrei erhältlich sein. Eine Zusammenfassung der Kriterien zeigt folgende Aufstellung.

5.4.1 Kriterien zur Bewertung

1. Funktionsumfang

- Unterstützung des DOM-Standard,
- Validierend,
- Unterstützung der SAX-API (Kann-Kriterium),
- DTD Zugriff (Kann-Kriterium).

2. Integrationsfähigkeit in das KHS/J

- Java-Bibliothek (Java Source oder JAR-Archiv)
- leichte Integration in Entwicklungsumgebung “IBM Visual Age 2.0”, daher geforderte JDK-Version höchstens 1.16
- Source Code verfügbar (Kann-Kriterium)

3. Kosten/Lizenzrechtliches

- kostenlos verfügbar
- kostenloser Vertrieb (lizenzfrei)

4. Produkt-Lebenszyklus

- Weiterführung des Produktes durch renommierten Hersteller oder erfolgreiche, nicht-kommerzielle Institution (OSF, GNU …)

Man kann gegen den aufgestellten Anforderungskatalog anführen, daß **keine Performance-Kriterien** angesetzt wurden. Dies liegt darin begründet, daß für die Import-Vorgänge keinerlei Echtzeit-Anforderungen bestehen und diese gegebenenfalls im Rahmen einer Stapel-Verabeitung über Nacht stattfinden können. Dennoch wurde im Hinblick auf die Performance recherchiert. Nach [Cooper 1999] entscheidet offensichtlich mehr die *Programmierprache* als die Implementation, resp. der Algorithmus über die Ausführungsgeschwindigkeit von XML-Prozessoren[65]. Folgende relative Performance-Raten wurden bei einer gegebenen Testapplikation gemessen.

Kombination	Produkte	Typischer Faktor [66]
Identische Programmiersprache (Java) Unterschiedliche Implementierung:	Java-XP / Java Xml4J	1,42
Identische Implementierung (Expat von J. Clark) Unterschiedliche Programmiersprache:	Java-XP/ C-Expat C	12,6

Da die Grundsatzfrage aus Gründen der optimalen Integration in das KHS-Gesamtsystem bereits zugunsten der Programmiersprache Java entschieden wurde, muß man sich also mit einer Performance- Einbuße von Faktor 12,6 (gegenüber C) bei einer “typischen” XML-Datei abfinden. Dagegen kann das Gewicht der Implementierung mit einem typischen Faktor von 1,42 vernächlässigt werden.

[65] An dieser Stelle muß angemerkt werden, daß die Genauigkeit der quantitativen Aussagen des Vergleiches mit Vorsicht zu genießen sind, jedoch die grobe Tendenz eindeutig ausfällt.

[66] gemessen bei gegebener Dateigröße (1,3 MB, Markup-Dichte 33%)

5.4.2 Vergleich konkreter Produkte

Bei [Middendorf 1999] findet man folgende Gegenüberstellung bezüglich des Funktionsumfanges von XML-Parsern, die in Java implementiert sind[67]:

XML-Parser					
Parser	**Version**	**Hersteller**	**JDK**	**DOM-Imlementierung**	**DTD-Zugriff**
XP	0.4	James Clark	1.1	-	X
XML4J	1.1.4	IBM	1.1	Level 1	X
XML-Parser	Ea2	JavaSoft	1.1	Level 1	-
msxml	1.8	Microsoft	1.1	-	-
lark	1.0	Tim Bray (Textality)	1.0	-	-

Abbildung 16: Gegenüberstellung von (Java-)XML-Parsern. Quelle [Middendorf 1999]

Da laut Anforderungskatalog eine DOM-Implementierung gefordert wird, kamen also lediglich die Parser von IBM und JavaSoft in die engere Wahl.

5.5 IBM: XML for Java (XML4J)

Die Entscheidung fiel zugunsten des Parsers *XML for Java* (XML4J) der Firma IBM (URL: http://www.alphaworks.ibm.com/tech/xml), da die problemlose Integration in die Entwicklungsumgebung Visual Age des gleichen Herstellers bei diesem Produkt zu

[67] Zur Enscheidungsfindung wurde zu Beginn der vorliegenden Arbeit obige Übersicht herangezogen. Jedoch wurde vom Autor eine weitere **umfangreichere Gegenüberstellung** gegen Ende der vorliegenden Ausarbeitung in [Maruyama et. al 1999], S. 316 entdeckt.

erwarten war. In der Überprüfung zeigte sich, daß dieser Parser *nicht nur sämtliche* geforderten Muß-Kriterien erfüllte, *sondern auch sämtliche* Kann-Kriterien.

Anmerkungen zu den Kriterien-Sparten:

1. Funktionsumfang

- neben den APIs SAX und DOM besitzt XML4J die proprietäre Schnittstelle ElementHandler[68],
- der Parser ist sowohl in der DOM-Betriebsart *als auch* in der Betriebsart "SAX" validierend[69],
- eine proprietäre Schnittstelle für den Zugriff auf die Bestandteile der DTD eines XML Dokumentes ist vorhanden,
- XPointer werden unterstützt,
- über die von der XML-Sprachspezifikation geforderten Zeichen-Codierungen[70] ([W3C 1998e, Kap. 4.3.2]) UTF-8 und UTF-16 hinaus unterstützt XML4J viele Codierungen, darunter Europäische, Japanische, Chinesische und Koreanische – insgesamt rund 30 ([Maruyama et al. 1999, S. 39]),
- die *Ausgabe* von XML-Dokumenten über einen generierten DOM-Baum (umgekehrter Betrieb!) wird mittels einer proprietären Schnittstelle ermöglicht,
- zum Parser ist der *XML-Editor* "Xeena"[71] erhältlich,
- die Module von XML4j können leicht in *Java-Beans* überführt werden ([Maruyama et al. 1999, S. 281 ff.],
- die Verarbeitung einer XML-Schema Sprache wird von XML4J im Gegensatz z. B. zu Microsofts msxml[72] derzeit noch **nicht** unterstützt[73].

---

[68] `ElementHandler` wird in [Maruyama et. al., S. 60ff.] erläutert

[69] im Falle der Betriebsart SAX erst seit der Version 2.0.11

[70] engl. character-encodings

[71] der unter URL: http://www.alphaworks.ibm.com/tech/xml zu finden ist

[72] http://msdn.microsoft.com/downloads/tools/xmlparser/xmlparser.asp, Stand. 16.8.1999

[73] Kurz vor Ende der vorliegenden Arbeit annoncierte IBM eine Version des XML4J mit W3C-Schema Unterstützung. Zitat aus den WEB-Seiten von IBM: „XML4J Early Access release!!! **Preliminary support for W3C XML Schema Language** and access to the DTD via the DOM." (http://www.alphaworks.ibm.com/tech/xml4j, Stand: 16.8.1999)

2. Integrationsfähigkeit in das KHS/J

XML4J ist sowohl im Source-Code als auch als JAR-Archiv erhältlich. Eine Anleitung zur Integration in Visual Age wird mitgeliefert, ebenso verschiedene Code-Beispiele wie der sogennannte „Tree-Viewer", der XML-Dokumente in Baum-Struktur und normaler Text-Editor Darstellung visualisieren kann.

3. Kosten/Lizenzrechtliches

Die *kostenlose Nutzung* und *lizenzgebührfreie Weitergabe* von Produkten (auch kommerziellen!) ist im Falle einer Registrierung möglich.

4. Produkt-Lebenszyklus

Es kann unterstellt werden, daß der renommierte Hersteller IBM, der bereits an der Erfindung von SGML beteiligt war, das Produkt noch lange Zeit professionell pflegen wird. Diese Vermutung wird auch durch jüngste Weiterentwicklungen des XML4J untermauert, wie z. B. die Unterstützung von XML-Schema.

Wie bereits in der tabellarischen Gegenüberstellung auf Seite 62 gezeigt, stellt XML for Java keinen Einzelfall für die Kombination von XML mit einer Implementierung in JAVA dar. Im folgenden Kapitel wird die Frage aufgegriffen, warum die Wahl von Anbietern xml-verarbeitender Produkte, wie Prozessoren oder Editoren auf die Programmiersprache Java fällt.

5.6 Exkurs: XML und Java

> "Seit der Standardisierung der Extensible Markup Language (XML) ist eine Vielzahl von Softwarepaketen zur Verarbeitung von XML-Dokumenten entstanden. Neben C[++] und Java werden hierbei Script-Sprachen wie Perl und Python eingesetzt. Unter den Herstellern von Java-Software für XML finden sich so prominente Namen wie Java-Soft (deren Paket allerdings noch im ´Early Access´ Stadium ist), Microsoft und IBM. Allein diese Tatsache zeigt schon, daß es sich bei dem Gespann XML/Java um ein Thema von strategischer Bedeutung handelt" [Middendorf 1999].

Daß sich gerade Java zunehmend als Partner von XML darstellt, ist vielleicht ein (reiner) Marketing-Erfolg der Firma Sun, wie beispielsweise von Behme vermutet[74], es gibt für die Kombination aus XML und Java jedoch auch rationale Gründe. Diese liegen vor allem im *WWW als Haupteinsatzgebiet* von XML begründet – im diesem Bereich hebt sich Java deutlich von seinen Konkurrenten (C++, Perl, Python, Visual Basic[75]) ab. Zusammengefaßt kann man Java in bezug auf die "XML-Qualifikation" wie folgt bewerten [vgl. auch Middendorf 1999; Behme, Mintert 1998, S. 221ff ; Fuchs 1999; Maruyama et. al. 1999, S. 25ff.].

Stärken

Web-Technologien: Applets, Servlets, JSP

Diese Technologien stellen einen Hauptvorteil von Java gegenüber anderen Programmiersprachen dar. Mit Hilfe von *Applets* kann ein XML-Dokument client-seitig verarbeitet werden[76]. Mittels der Server-Technologien *Servlets* und *JSP* können XML-Seiten komfortabel von WEB-Servern zur Verfügung gestellt werden. Keine andere Programmiersprache kann im Bereich auf die Leistungsfähigkeit im Kontext des WEB vergleichbares aufweisen.

---

[74] "Es gibt Anzeichen dafür, daß die Strategie von Sun aufgeht, Java als *die* Sprache für XML zu propagieren und damit den Stand von Java weiter zu stärken." [Behme 1998, S. 221]

[75] um die verbreitetsten Programmier-/Scriptsprachen zu nennen, die derzeit im Bereich XML-Verarbeitung eingesetzt werden

[76] Umgekehrt kann XML von Applets als Daten-Austauschformat eingesetzt werden.

SW-Engineering/ Code Management: Bean-Technologie/ Java Packages

Java Beans stellen ein wichtiges Werkzeug zur komponenten-basierten SW-Entwicklung dar. Mittels dieser Technologie kann OO-Programm-Funktionalität perfekt gekapselt, leicht konfiguriert und adaptiert werden. Der *Package-Mechanismus* zur Distribution von Source Code oder Klassenbibliotheken ist leicht zu handhaben. Ein Makefile, wie z. B. in C/C++ zur Compilierung des Source-Codes üblich, kann entfallen. Auf diese Art und Weise ist das Teilen von Sourcen und Bibliotheken leicht möglich.

Laufzeit-Verhalten: Dynamic Class Loading, Multithreading, Virtual-Machine-Konzept

Dynamic Class loading erlaubt das dynamische Laden von Programm-Modulen zur Laufzeit. Weil sich - im Gegensatz zu monolithischen Systemen - lediglich die momentan benötigten Klassen[77] im Hauptspeicher des Computers befinden müssen, werden einerseits Ressourcen geschont, andererseits müssen nicht alle benötigten Klassen lokal vorhanden sein – sie können auch über das Netzwerk geladen werden. Hier kommt also die Eignung von Java für verteilte Systeme zum tragen. Mittels *Multithreading* kann die parallele Verarbeitung komplexer Dokumente geleistet werden. Durch – für alle gängigen Rechnerplattformen verfügbare – *Virtual Machines* können Java Programme plattformunabhängig ausgeführt werden. Diese Funktionalitäten werden von anderen Programmiersprachen überhaupt nicht, oder nur bei Verwendung von Zusatz-Bibliotheken unterstützt.

Syntaktisches: Ähnlichkeit mit der Standard –Browser Scriptsprache Javascript

Die Ähnlichkeit von Java mit Javascript, der Standard-Browser-Scriptsprache, werden SW-Entwickler, die XML im WEB Kontext einsetzen möchten[78], zu schätzen wissen. Hier besitzt Java quasi „ergonomische“ Vorteile.

Plattformunabhängige Datenbank-Technologie JDBC

[77] die im engen Sinne Programm-Module darstellen.

[78] (was nicht die Ausnahme, sondern die Regel sein dürfte)

Die java-eigene Datenbank-Technologie „JDBC“ ermöglicht Java Programmen mit unterschiedlichen *Datenbank-Managementsystemen* (DBMS) zu interagieren. Diese nehmen im Bereich Dokumenten-Management von Web-Servern an Bedeutung zu. Die technologische Umsetzung von JDBC funktioniert wie folgt:
JDBC gestattet es Java Programmen, Standard-SQL Befehle abzusetzen. Die Programme wenden sich für den Zugriff auf die Datenquellen an den sogenannten „Treiber-Manager“, in welchen proprietäre JDBC-Treiber für den Zugriff auf DBMS „eingeklinkt“ sind. Dabei werden JDBC-Treiber von DBMS-Anbietern für ihre Produkte zur Verfügung gestellt. Durch die Plattformunabhängigkeit von Java werden auch die DB-Applikationen plattformunabhängig.

Sonstiges: Unicode-Zeichensatz

Der – von XML verwendete - UNICODE-Zeichensatz wird von Java, im Gegensatz zu fast allen anderen Programmiersprachen, *standardmäßig* unterstützt.

Schwächen

Performance

Im Gegensatz zu (Native-Code-)Compilern, die reinen Maschinencode erzeugen, arbeitet Java mit dem sogenannten Byte-Code. Dieser muß während der Ausführung von der Virtual Machine in den Maschinencode der jeweiligen lokalen (HW/SW-)Plattform übersetzt werden. Hierbeit geht (Ausführungs-)Geschwindigkeit verloren. Die geringe Performance von Java Anwendungen im Vergleich z. B. zu Anwendungen die mit C++ entwickelt wurden[79], hängt in erster Linie mit diesem Prinzip zusammen. Diese Schwäche teilt Java mit Sriptssprachen (Perl, Python). Diese Sprachen, die ebenfalls auf Basis eines Interpreters arbeiten, legen sogar ein schlechteres Performance-Verhalten an den Tag.

Ausnutzung (hardware-)plattformspezifischer Vorzüge

Auch die Ausnutzung plattformspezifischer Vorzüge (wie z. B. spezielle Hardware-beschleuniger zur Grafikdarstellung) ist sub-optimal da der von Java erzeugte Bytecode lediglich einem gemeinsamen Nenner vieler Rechnerplattformen ansprechen kann.

[79] vgl. Anhang 9.2

Anmerkungen zu den Nachteilen

Beide Nachteile können jedoch – falls erforderlich - durch den Einsatz von JIT (Just-In-Time) Compilern bzw. gewöhnlicher „Native-Code“ Compiler weitgehend behoben werden.

JIT-Compiler sind derzeit u. a. von der Firma Microsoft (Produktname: Visual J++) erhältlich. Marktreife Java-Native-Code Compiler werden bereits von Asymetrix und Siemens angeboten. Es ist davon auszugehen, daß Native-Code Compiler für Java die Leistungsfähigkeit von C++-Compliern erlangen werden, da Java grammatikalisch eine enge Verwandtschaft mit C++ aufweist.

Fazit

Die Stärken von Java überwiegen in der Bilanz eindeutig. Insbesondere im Hinblick auf den Anwendungsbereich des WWW liegen deutliche Vorteile gegenüber den Konkurrenten vor. Der einzige, schwerwiegende Nachteil gegenüber den Haupt-Konkurrentinnen C/C++, die Performance kann, wie beschrieben, durch den Einsatz von (JIT-)Compilern eliminiert werden. Ein letztes Argument für die Prädestinierung der Kombination aus XML und Java, vielleicht eher philosophischer Natur, soll an dieser Stelle nicht vorenthalten werden. “…both languages [XML and Java] are simplifications of powerful beasts [SGML resp. C++, ☺] whose complexity had gotten out of control” [Fuchs 1999]. Somit stellt Java in Verbindung mit XML tatsächlich eine *Schlüsseltechnologie* dar.

6 REALISIERUNG ImportFilter XML4KHS

6.1 Zielsetzung des Prototypen

Ziel des Importfilters, **Arbeitstitel XMl4KHS** war es, vernetzte **Daten eines KHS-Testforums** (Benutzer, Beiträge, Rubriken ...), welche in XML codiert sind, in die Datenbasis des KHS zu integrieren[80].

6.2 Grundprinzip

Das Grundprinzip des Filters besteht in der Transformation einzelner XML-Entitäten in KHS-Objekte. Als Entität wird dabei analog dem Entity Relationship Modell, das im Bereich Danbank-Modellierung eingesetzt, wird folgendes verstanden: „Eine Entität (entity) ist ein individuelles und identifizierbares Exemplar von Dingen, Personen oder Begriffen der realen oder der Vorstellungswelt und wird durch Eigenschaften beschrieben“ [Balzert 1996, S. 138] . Als *XML*-Entität bezeichnet der Autor Entitäten, die in XML codiert sind. Die folgende Abbildung zeigt beispielhaft die Überführung einer XML-Entität (hier: eines Forumsartikels) in ein KHS Objekt.

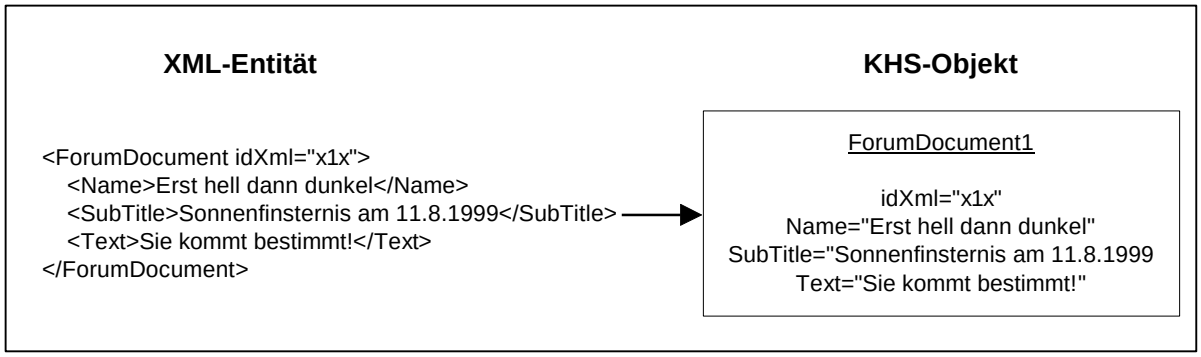


Abbildung 17 Überführung einer XML-Entität in ein KHS-Objekt. Quelle: eigen

Der Forumsartikel besitzt die Eigenschaften Name, SubTitle (Untertitel) und Text, die durch XML-*Elemente* („Tags“) mit dem gleichem Namen codiert sind und ferner die

[80] Das ursprüngliche Ziel der Integration von lexikografischen Daten wurde aus Gründen des Mehraufwandes zur Erstellung einer **Klassenhierarchie** für spezifische, lexikografische Daten fallen gelassen .

Eigenschaft „idXml“ (eine Daten-ID zur Verwaltung) welche als XML-*Attribut* codiert ist. Ein XML-Element mit dem Namen „ForumDocument“ umschließt die Inhalte.

Bei dieser Transformation liegt laut [Fuchs 1999, letzter Abschnitt] ein sogenanntes one-to-one Mapping vor, das heißt: Jede XML-Entität wird auf ein Objekt im KHS abgebildet und jeder XML-Element/Attribut-Inhalt wird in eine Member-Variable des KHS-Objektes transformiert. Das korrespondierende Objekt muß beim Importvorgang erzeugt werden, wobei die zugehörige Klasse (User, ForumDocument, Heading …) bereits im KHS vorhanden sein muß.

Beziehungen zwischen Objekten, wie sie beispielsweise von Hypertext-Links ausgehen, werden von der beschriebenen Transformation *nicht zwingend* berücksichtigt. Hierzu ist eine semantische Analyse der zu transformierenden Entitäten in bezug auf Link-Eigenschaften notwendig, die beim vorliegenden Prototypen implementiert wurde.

6.3 Rahmenbedingungen zur Implementierung/Anforderungen

Zunächst wurde daran gedacht, den Import-Filter statisch zu implementieren. Die erste Filterversion konnte lediglich Entitäten des Typs „Mitarbeiter“ in KHS-Objekte der Klasse „User“ übertragen. Dabei wurde der Source-Code „hart-codiert“, das heißt, die Namen der XML-Elemente und -Attribute wurden als String-Konstanten festgelegt. Analog wurde der Zugriff auf die KHS-Klasse (User) und deren Member-Variable, respektive Kapselungs-Methoden[81] (get/set), fixiert.

Diese „zementierte“ Codierung wurde von Anja Odenthal, einem Mitglied der KHS Projektgruppe der FG Informationswissenschaft, wegen der im Rahmen der Weiterentwicklung zu erwartenden Änderungen an der KHS-Klassenstruktur, als nicht praktikabel bezeichnet. Stattdessen schlug Anja Odenthal eine „**generische“ Implementierung** des Filters vor, die sich selbständig an die Inhalte von XML-Dokumenten anpassen sollte. Konkret sollte diese in der Lage sein, XML-Entitäten eines

[81] Als „Kapselungs-Methoden werden im weiteren solche Methoden verstanden, die der Kapselung von Member-Variablen, also dem Setzen und Lesen derselben dienen.

beliebigen XML-Dokumentes zu erkennen, automatisch nach korrespondierenden KHS-Klassen zu suchen und entsprechende Objekte zu erzeugen. Selbstverständlich müßten die Entitäten dabei in einer vorgegebenen *Grund-Struktur* verfaßt sein.

Als weitere gewünschte Anforderung kam folgender Aspekt hinzu: Die Implementierung der **KHS-Objekte durfte nicht „angetastet" werden**, d.h.: die Methoden bzw. Member-variable durften nicht verändert werden/ auch keine hinzugefügt werden. Dies hatte zur Folge, daß der Filter die Inhalte der instanziierten Objekte *nur* mittels deren Public-(Kaspelungs-) Methoden oder Public-Member-Variable[82] setzen durfte. Eine der Implementations-Varianten, die Implementierung des Filters in der obersten KHS-Klasse HypertextObject, schied somit aus.

6.4 Schematische Darstellung

Ein schematische Darstellung des Filtervorganges gibt folgende Abbildung wieder.

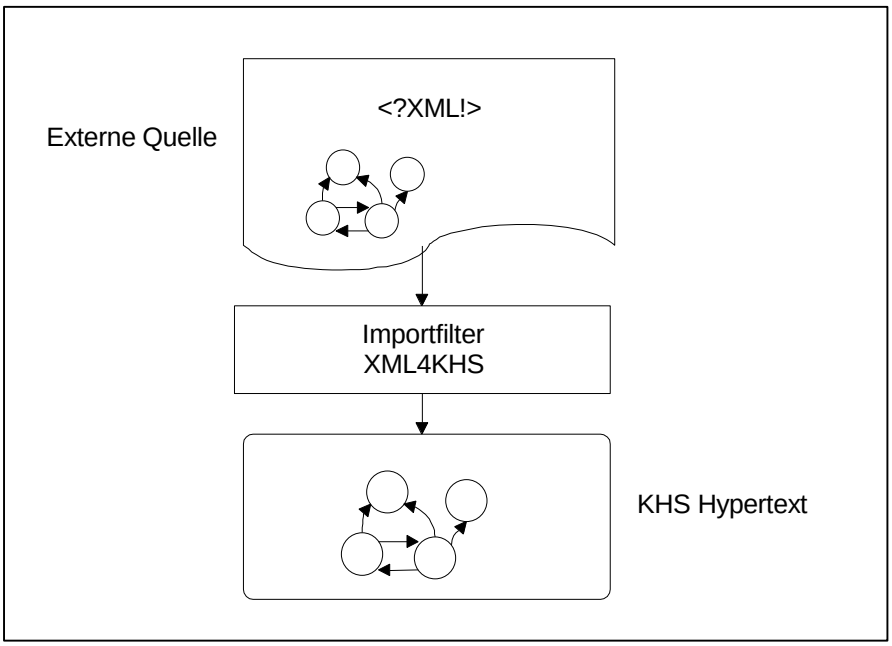


Abbildung 18 Schematische Darstellung des Importfilters. Quelle: eigen

Diese Architektur mußte schon bald ergänzt werden. Grund: Es stellte sich heraus, daß für die Arbeit des Filters zusätzliche Meta-Information erforderlich ist, die in einer eigenen Datei abgelegt wurde. Diese Datei, die im Rahmen des Projektes als „Filter-Definition-

[82] Member-Variable mit *Public*-Zugriff werden im KHS aus Design-Gründen in der Regel vermieden.

Datei“ bezeichnet wird, ist im nachfolgenden Kapitel näher beschrieben. Es ergibt sich für die schematische Darstellung des Filters folgende, erweiterte Anordnung:

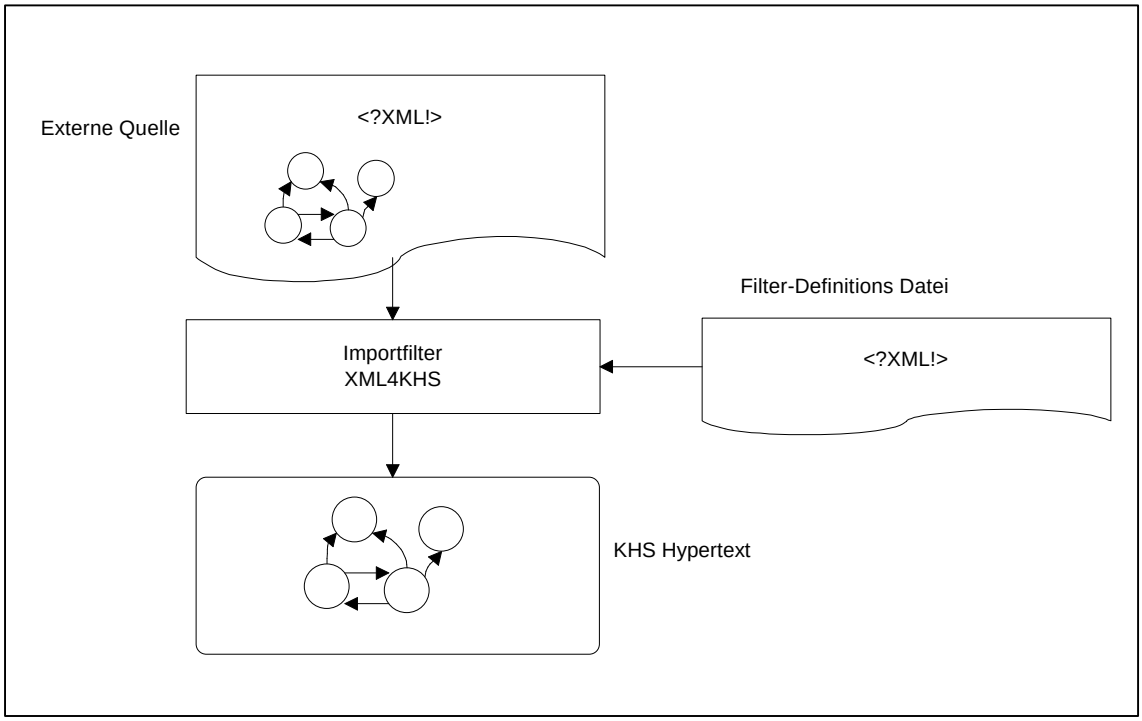


Abbildung 19 Schematische Darstellung des Filters XML4KHS mit Filter-Definitions-Datei. Quelle: eigen

Diese grobe Darstellung wurde im Rahmen des Entwicklungsprozesses zu folgender Detail-Darstellung verfeinert[83]:

[83] Aus Gründen der Übersichtlichkeit wurden nur solche Verzweigungen dargestellt, die für den Ablauf wesentlich sind.

Schematische Darstellung

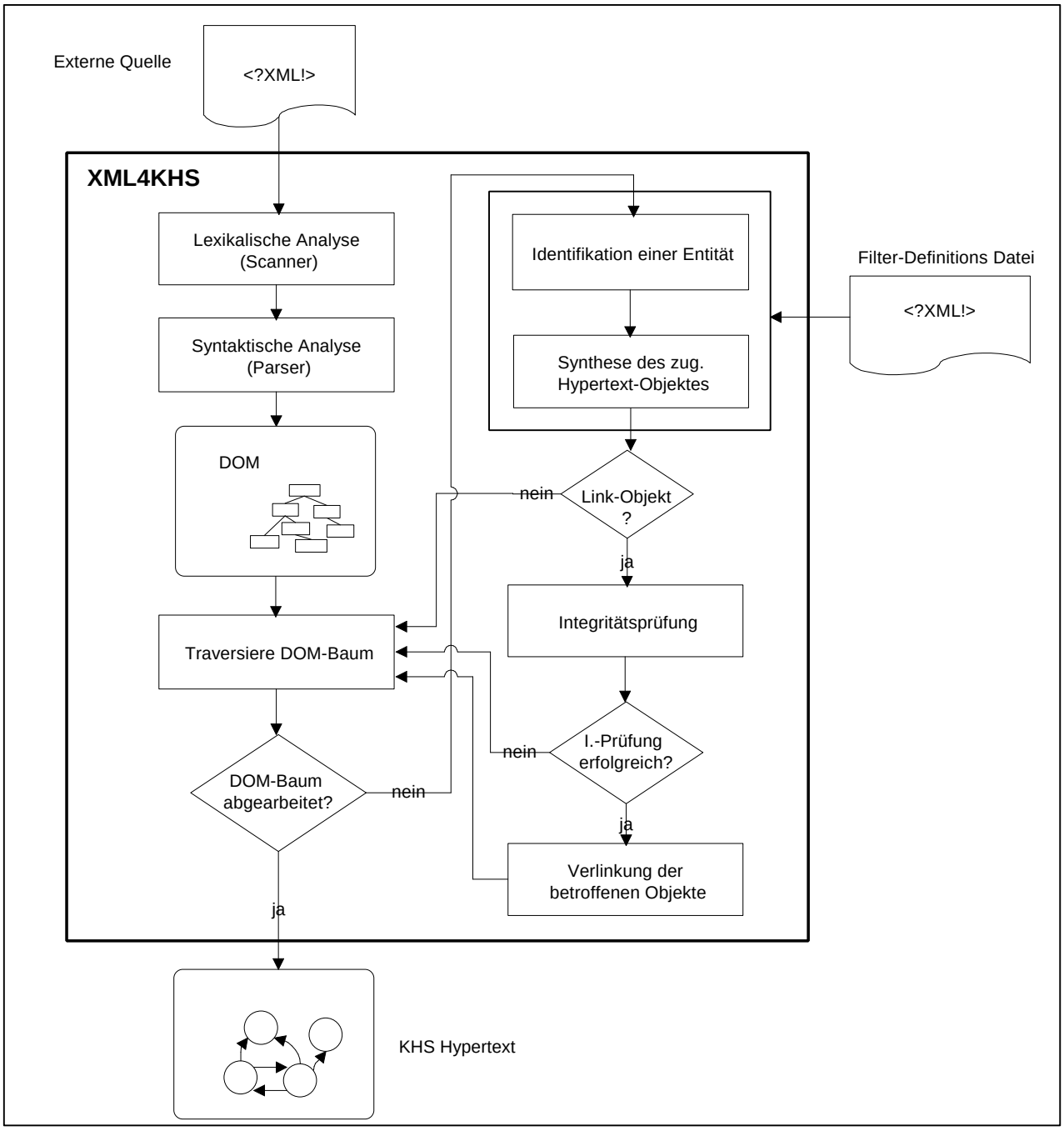


Abbildung 20 Schematische Darstellung des Filters XML4KHS (detailliert). Quelle: eigen

Erklärung

Die XML-Datei wird mittels eines Parsers zunächst lexikalisch, dann syntaktisch analysiert. Diese Verarbeitungsschritte münden in die Überführung der Bestandteile des

XML-Dokumentes in die Form eines DOM-Baumes. Dieser Baum wird rekursiv traversiert, das heißt: die Objekte des DOM-Baumes (vgl. Kapitel 5.2) werden durchlaufen. Dabei werden XML-Entitäten identifiziert. Nach der Identifikation einer Entität wird das entsprechende KHS-Objekt erzeugt, also ein Hypertext-Knoten oder Link. Dabei werden auch die Member-Variablen des Objektes über die im zugehörigen DOM-Teilbaum enthaltenen Daten gesetzt.
Die Identifikation der Entitäten und die Synthese des korrespondierenden KHS-Objektes wird über die Daten der Filter-Definitions-Datei gesteuert. Im Falle eines Links wird dabei zusätzlich die Verlinkung der betroffenen KHS Objekte vorgenommen[84]. Im Rahmen der Integritätsprüfung wird geprüft, ob der generierte Hypertext in sich konsistent ist. Die Verarbeitung endet, wenn alle Entitäten des DOM-Baumes berücksichtigt wurden.

6.5 Generisches Prinzip: Filter-Definitions-Datei

Die Filter-Definitions-Datei enthält (Meta-)Information über die Zuordnung von XML-Entitäten zu den korrespondierenden KHS-Klassen, respektive die Zuordnung von deren XML-Element- und Attribut-Namen zu den Member-Variablen. Als Dateiformat für die Codierung dieser Information wurde pragmatischerweise XML gewählt[85]. Die Erzeugung dieser Datei orientiert sich an den Klassen, die im KHS bereits vorhanden sind.

Da die Datei, soweit möglich, **automatisch** erstellt wird, setzt das ihr zugrunde liegende Konzept den geforderten **generischen Teil der Implementierung** um.

Anmerkung
Es wird darauf hingewiesen, daß sich die Inhalte der Filter-Definitions-Datei auf einer *Meta-Ebene* gegenüber den Inhalten einer zu importierenden XML-Datei befinden - analog dem Prinzip, das auch bei XML-DTDs oder treffender, bei XML-Schemata angewandt wird: *Werte* von XML-Attributen in der *Filter-Definitions*-Datei spezifizieren

[84] Die Verlinkung wird durch die Einrichtung von Rerefenzen der betroffenen KHS-Objekte auf das Link-Objekt realisiert.

[85] Dieser Gedanke wurde (im Rahmen einer länger geführten Diskussion per Email) erstmals von Rolf Aßfalg geäußert.

Attribut- und Element-*Typen* in einer XML-*Quell*-Datei. Analog werden *Namen* von KHS-Klassen und Klassen-Methoden durch Attribut-*Werte* repräsentiert.

Folgender Auszug aus einer konkreten Datei illustriert den typischen Aufbau einer Filter-Definition:

```
<?xml version="1.0"?>
<ClassSpecifications>

  <!-- XML4KHS Filter-Definition File -->
  <?FilterVersion 1.0?>
  <?CreationDate Mon Aug 23 13:20:11 CEST 1999?>
  <!--+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +-->
  <Class name="ForumDocument" type="de.iwk.khs.model.ForumDocument">
(1)
    <Hierarchy>
      <Ancestor level="1" type="de.iwk.khs.model.Node"/>
      <Ancestor level="2" type="de.iwk.khs.model.HypertextObject"/>
(2)
      <Ancestor level="3" type="java.lang.Object"/>
    </Hierarchy>
    <Content>                                                          (3)
      <ContentDefinition name="SubTitle" isElement="yes">
        <setMethod name="setSubTitle" type="java.lang.String"/>        (4)
        <getMethod name="getSubTitle" type="java.lang.String"/>
      </ContentDefinition>
      <ContentDefinition name="Text" isElement="yes">
        <setMethod name="setText" type="java.lang.String"/>
        <getMethod name="getText" type="java.lang.String"/>
      </ContentDefinition>
    </Content>
  </Class>
    ...
<!--+ + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +-->
  <Class name="HypertextObject" type="de.iwk.khs.model.HypertextObject">
    <Hierarchy>
      <Ancestor level="1" type="java.lang.Object"/>
    </Hierarchy>
    <Content>
      <ContentDefinition name="CreationDate" isElement="yes">
        <setMethod name="setCreationDate" type="java.util.Date"/>
        <getMethod name="getCreationDate" type="java.util.Date"/>
      </ContentDefinition>
      <ContentDefinition name="IdXml" isElement="no">
```

```
        <setMethod name="setIdXml" type="java.lang.String"/>
(5)
        <getMethod name="getIdXml" type="java.lang.String"/>
      </ContentDefinition>
      ...
    </Content>
  </Class>
    ...
  </ClassSpecifications>
```

Eine zu dieser Konvention gehörende XML-Entität sieht folgendermaßen aus:

```
<ForumDocument idXml="x1x">
 <Name>Erst hell dann dunkel</Name>
 <SubTitle>Sonnenfinsternis am 11.8.1999</SubTitle>
 <Text>Sie kommt bestimmt!</Text>
</ForumDocument>
```

Erläuterungen zum Beispiel der Filter-Definitions-Datei

Abschnitt (1): XML-Entitäten, die mittels des Element-*Namens* („Tag“) "ForumDocument" zusammengefaßt sind, werden zu Instanzen des KHS-*Typs* "de.iwk.khs.model.ForumDocu ment".

```
<Class name="ForumDocument" type="de.iwk.khs.model.ForumDocument">
```

Abschnitt (2): Der zu beschreibende KHS-Typ (s. Abschnitt (1)) besitzt die Vorfahren "de.iwk.khs. model.Node", "de.iwk.khs.model.HypertextObject" und "java.lang.Object". Diese Information wird benötigt, um auch in den Vorfahren-Klassen nach Merkmalen der Klasse zu suchen. Diese Merkmale sind die Member-Variable bzw. deren Kapselungs-Methoden, die von den Vorfahren geerbt werden.

```
<Hierarchy>
      <Ancestor level="1" type="de.iwk.khs.model.Node"/>
      <Ancestor level="2" type="de.iwk.khs.model.HypertextObject"/> (2)
      <Ancestor level="3" type="java.lang.Object"/>
</Hierarchy>
```

Abschnitt (3): Das Element „Content“ umrahmt die *Inhalte* der Entitäten.

```
<Content> (3)
```

Abschnitt (4): Eine „ContentDefinition“ beschreibt eine Eigenschaft (=„Inhalt“) der Entität[86]. Im obigen Code-Ausschnitt wird die Eigenschaft mittels des XML-*Elementes* „SubTitle“ codiert. Die Tatsache, daß hier ein Xml-*Element* - im Gegensatz zu einem XML-*Attribut* - zur Codierung der Eigenschaft verwendet wird, wird durch die Zuweisung „isElement="yes"“ festgelegt Die zur Entitäts-Eigenschaft „SubTitle“ gehörige Member-Variable des KHS-Objektes wird mittels der Kapselungs-Methode "setSubTitle" gesetzt und mittels der Kapselungs-Methode "setSubTitle" gelesen. Der Name der Member-Variablen selbst ist nicht angegeben.

Anmerkung: Es wurde festgelegt, daß die Inhalts-Definitionen sich auf Member-Variable bzw. Kapselungs-Methoden beschränken, die in der Klasse selbst deklariert sind. Vererbte Eigenschaften treten als Inhalts-Definitionen der Vorfahren-Klassen auf. Dieses Verfahren wurde eingeführt, um Redundanz zu vermeiden, die bei manuellen Nachbesserungen an einer Filter-Definitions-Datei zu Fehlern führen könnte.

```
<ContentDefinition name="SubTitle" isElement="yes">
        <setMethod name="setSubTitle" type="java.lang.String"/> (4)
        <getMethod name="getSubTitle" type="java.lang.String"/>
</ContentDefinition>
```

Abschnitt (5): Die Spezifikation der Klassen „HypertextObject“ und „Node“ liefern aufgrund der Vorfahren-Beziehung[87] zur Klasse „ForumDocument“ ebenfalls relevante Informationen über die Entitäten, die ein Forum-Dokument repräsentieren. Ein typischer Effekt wird anhand des Abschnitts (5) der Beschreibung der Klasse „HypertextObject“ illustriert. Die Entitäts-Eigenschaft „idXml“ - eine Daten-ID, die der Verwaltung dient - wird an die Entitäten vom Typ „ForumDocument“ vererbt. Die Codierung der Eigenschaft wird mittels eines XML-Attributes geleistet (isElement="no").

```
<ContentDefinition name="IdXml" isElement="no">
        <setMethod name="setIdXml" type="java.lang.String"/>
        <getMethod name="getIdXml" type="java.lang.String"/>
```

[86] Beim Schreiben der Dokumentation, also nach Abschluß der Implementierung, fiel dem Autor auf, daß der englische Ausdruck „**Property**Definition“ eventuell die bessere Alternative zu „ContentDefinition“ gewesen wäre. Diese Bezeichnung bringt die Verbindung zu einer Entitäts-Eigenschaft seiner Meinung nach besser zum Ausdruck.

[87] Alternativ könnte statt einer „Vorfahren-Beziehung“ auch von einer „Generalisierungs-Beziehung“ gesprochen werden.

```
</ContentDefinition>
```

Erzeugung

Die Filter-Definitions-Datei wird von dem kommandozeilen-orientierten Standalone-Programm des XML4J „CreateFilterDefinitionFile“ zunächst automatisch erstellt.

Die Generierung richtet sich, wie bereits erwähnt, nach den Eigenschaften der zu beschreibenden KHS-Klassen. Diese Eigenschaften bestehen in den Klassen-Vorfahren und Kapselungs-Methoden.

Implementierungsseitig wurde das Problem der Auffindung von Vorfahren und Kapselungs-Methoden der Klassen mit dem Java-Standard Package java.lang.reflect realisiert (vgl. [Naughton, Schildt 1999, S. 826 ff.; SUN 1998]). In diesem Paket sind primär die Klassen Method, Constructor, Field und Modifier von Bedeutung. Diese gestatten den Zugriff auf Methoden, Konstruktoren, Member-Variablen und auf die zugehörigen Zugriffsrechte[88]. Die „Reflect“-Klassen arbeiten mit der Klasse java.lang.Class zusammen, die unter anderem die Instanziierung von Objekten jeder beliebigen Klasse[89] erlaubt.

Zur Ausgabe in Form einer XML-Datei wurden proprietäre Methoden der Klasse TXDocument[90] des Parsers XML4J verwendet (vgl. [Maruyama et al. S. 69 ff.]). Hierzu wird zunächst ein DOM-Baum, welcher die Filter-Definitions-Datei 1:1 repräsentiert erzeugt. Danach erfolgt die Ausgabe dieses Baumes in Form einer XML-Datei, sozusagen „serialisiert“.

Das standalone-Programm createFilterDefFile wird per Kommandozeile ausgeführt. Als Kommandozeilen-Argumente werden die zu spezifizierenden Klassen selbst (i. o.

[88] (private, public, protected, …)

[89] Zur Instaziierung von Objekten muß die zugehörige Klasse bereits vorhanden sein.

[90] Die Klasse TXDocument ist übrigens nach dem Erzeugungsmuster „Fabrik“ (engl. „factory“) aufgebaut (vgl. [Maruyama et al. 1999, S. 70 ff.]).

Beispiel: die Klasse "de.iwk.khs.model.ForumDocument") explizit angegeben[91]. Sodann erzeugt das Programm eine Datei des oben erläuterten Typs.

6.5.1.1 Umfang der automatischen Generierung

a) Vorfahren einer Klasse

Die Ermittlung der Vorfahren der Klasse erfolgt vollautomatisch.

b) Kapselungsmethoden

Das Auffinden von Kapselungsmethoden der Member-Variablen ist vollautomatisch kaum zu bewältigen. Der Grund hierfür ist, daß es schwer herauszufinden ist, welche Methoden einer Klasse Kapselungs-Methoden sind. Nach verschiedenen Überlegungen, die der Autor in Verbindung mit den Mitgliedern der KHS-Projektteam Rolf Aßfalg und Michael Metzler anstellte wurden folgende vier Kriterien ermittelt, die Kapselungs-Methoden kennzeichnen:

1. **Präfix:** Der Name der Methode zum Setzen der Instanzvariablen besitzt den Präfix „set“, die Methode zum Lesen der Instanzvariablen, den Präfix „get“ (KHS-Programmierkonvention).

2. **Methoden-Paare:** Zu einer gefundenen Methode mit dem Namen der Form „getXY“ existiert eine Methode des Namens „setXY“ (KHS-Programmierkonvention).

3. **Datentypen:** Der Rückgabetyp der Get-Methode ist gleich dem Parametertyp der Set-Methode

4. **Zugriff:** Beide Methoden sind „public“ deklariert.

[91] Ein Verfahren, nach welchem automatisch einige oder sämtliche Namen, der im KHS implementierten Klassen gefunden werden, konnte vom Autor nicht in Erfahrung gebracht werden.

Dieses Verfahren besitzt Grenzen, die vermutlich meistens durch Verletzung der Namenskonventionen der o. g. Punkte 1 und 2 erreicht werden.

Beispielsweise verstoßen die Methoden-Namen „*Setze*Titel“ - „*Lese*Titel“ gegen Kriterium 1 und die Methoden-Namen set*Heading*Name/get*Rubrik*Name verstoßen gegen Kriterium 2.

Diese Kapselungsmethoden würden vom FilterDefWriter nicht automatisch erkannt und müßten deshalb manuell eingetragen werden (s. folgendes Kapitel).

c) Entscheidung XML-Element oder -Attribut

Als Name des zu einer Member-Variablen korrespondierenden XML-Elementes/Attributes wird der Rumpf des Namens der Kapselungsmethode (zum *Lesen* der Variable) festgelegt. Als Rumpf des Namens wird dabei der Teil des Methoden-Namens verstanden, der auf den Präfix „get“ folgt.

Beispiel:

Methoden-Name	XML-Element-Name
getHeadingText	HeadingText
getName	Name

Grundsätzlich wird angenommen, daß die Member-Variable als XML-Element codiert wird, weshalb in der zugehörigen Inhalts-Definition das Attribut isElement defaultmäßig den Wert “yes“ erhält. Soll die Variable als XML-Attribut codiert werden, so muß dies manuell korrigiert werden (s. folgendes Kapitel).

6.5.1.2 Manuelle Anpassung

Der zweite Schritt bei der Erzeugung der Filter-Definitions-Datei ist die manuelle *Anpassung* der automatisch generierten Datei. Zwei Punkte müssen dabei beachtet werden:

a) Falls eine Member-Variable nicht mittels eines XML-Elementes gekennzeichnet werden soll, muß isElement="yes" in isElement="no" umgewandelt werden.
b) Sind bei der beschriebenen automatischen Erzeugung Membervariable nicht berücksichtigt worden, so müssen diese von Hand nachgetragen werden.

Die Nachbesserung wird gemäß dem, anhand des Beispiels auf Seite 75 ff. gesehenen, Format abgelegt – hier in einer Notation mit Platzhaltern dargestellt:

```
<ContentDefinition name=<XML-Element/Attribut-Name> isElement=<XML-Element
„yes“|„no“>>
        <setMethod name=<Name Set Methode> type=<Parametertyp Set Methode>/>
        <getMethod name=<Name Set Methode> type=<Rückgabetyp Get Methode>/>
</ContentDefinition>
```

6.5.2 Schema-Konzept

Diese Datei ist ähnlich wie ein XML-Schema aufgebaut. Nachdem der IBM-Parser „XML4J“ zum Ende der vorliegenden Arbeit in einer Version mit Unterstützung von XML-Schema- Sprachen (hier: W3C Schema) erschien, wird empfohlen, das Format der Filter-Definitions- Datei zu einem späteren Zeitpunkt an diese Syntax anzupassen. Damit wäre auch die *Validierung* zu importierender XML-Dateien in der von den Schemata bekannten, mächtigen Art und Weise, möglich

6.6 Klassendiagramm/ Beschreibung der Klassen

Folgende Abbildung zeigt die Klassen/Schnittstellenhierarchie des XML4J in UML-Notation. Es werden lediglich die wichtigsten Bestandteile dargestellt – verzichtet wird beispielsweise auf die Darstellung der Klassen für Standalone-Anwendungen

(CreateFilterDef, TestFilterDefWriter/Reader) und die Assoziationsbeziehungen zwischen Klassen und Schnittstellen, die lediglich Konstantendefinitionen enthalten.

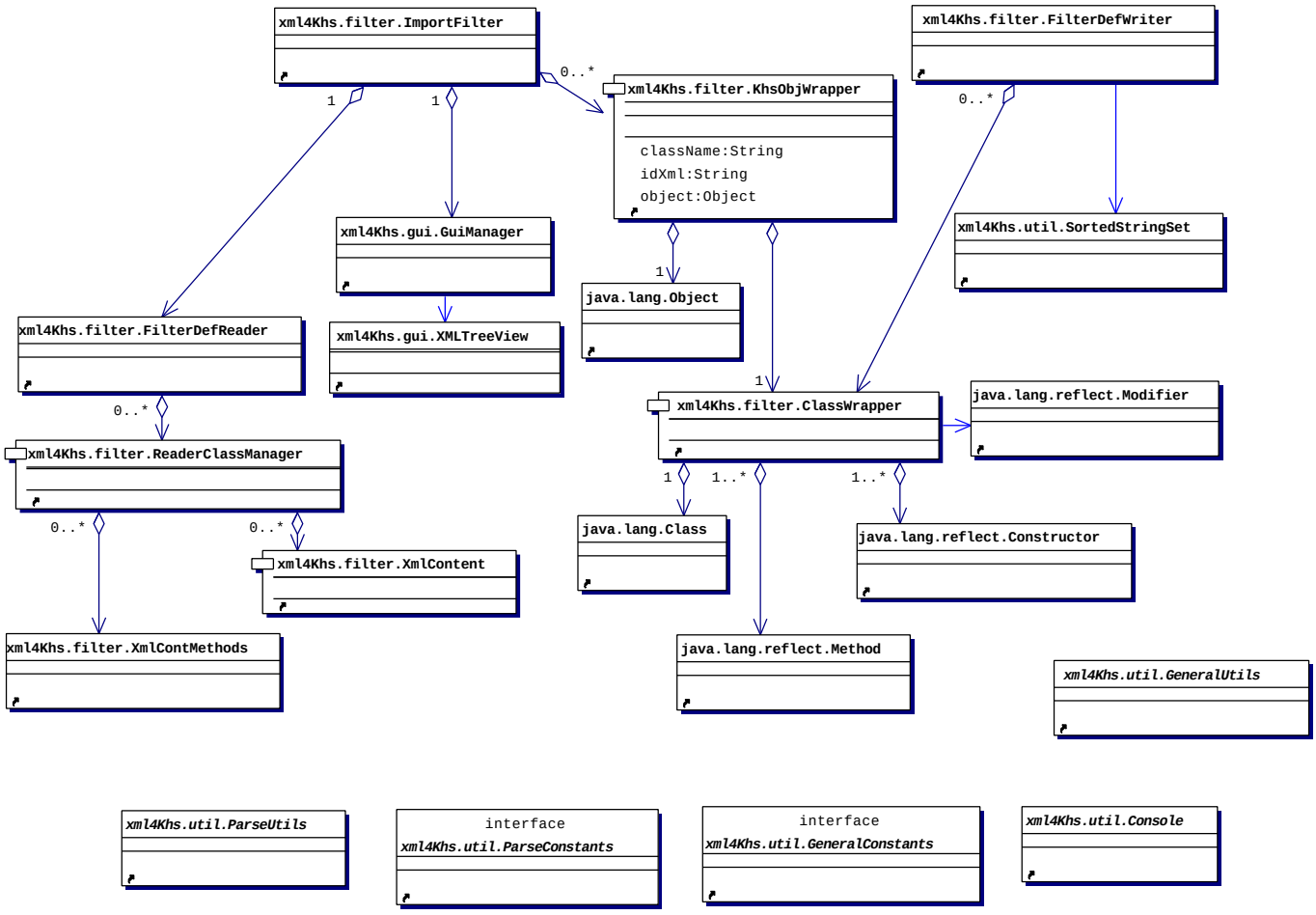


Abbildung 21 Klassendiagramm des Importfilter. Quelle: eigen

Es werden nur die wichtigsten Klassen/Schnittstellen erläutert. Weitere Informationen können dem Anhang „API-Dokumentation“ oder dem Source-Code entnommen werden, der auf der mitgelieferten CD zu finden ist.

6.6.1 Paket: xml4Khs.filter

ClassWrapper

Beschreibung

Diese Klasse wird von der Klasse FilterDefWriter benötigt, die die Generierung der Filter-Definitions-Datei vornimmt. Sie erweitert die Java-Standard-Klasse java.lang.Class um diverse Methoden (unter anderem zur Erzeugung eines DOM-Baumes, der die Eigenschaften der Klasse enthält) und ist dabei nach dem Strukturmuster „Adapter“ vgl. [Gamma et al. 1996, S. 151], das auch „Wrapper“ (dt. etwa „Umwickler“) genannt wird, aufgebaut.

Implementierung

Das Wrapping-Prinzip wird nicht über Vererbung, sondern über Aggregation hergestellt. Dies ist dadurch bedingt, daß die Java Klasse „Class“ mit dem Modifier „final“ deklariert ist und keine Tochterklassen erlaubt.

Zentrale Methoden

getAncestorNames, createDomTreePart

FilterDefReader

Beschreibung

Diese Klasse stellt dem Importfilter (Klasse ImportFilter) die in der Filter-Definitions-Datei enthaltenen Informationen zur Verfügung.

Implementierung

Es werden mehrere Instanzen der Klasse ReaderClassManager aggregiert, welche die Information (get/setMethoden, Vorfahren, …) über *genau eine* der – in der Filter-Definitions-Datei spezifizierten - Klassen enthalten.

Zentrale Methoden

elementToClass, getXmlContMethods, readXmlFile, traverseDomBranch (private)

FilterDefWriter

Beschreibung

Diese Klasse erstellt eine Filter-Definitions-Datei.

Implementierung

Es werden Instanzen der Klasse ClassWrapper aggregiert, die jeweils eine zu spezifizierende Klasse oder eine Vorfahren-Klasse umwickeln. Diese Instanzen sind in der Lage, DOM-Teil-Bäume zu erzeugen. Diese Teilbäume werden vom FilterDefWriter zu einem vollständigen DOM-Baum zusammengeführt, der die Informationen über alle zu spezifizierenden Klassen und deren Vorfahren enthält. Dieser Baum wird dann als XML-Datei ausgegeben.

Zentrale Methoden

createXmlFile

ImportFilter

Beschreibung

Zentrale Klasse zum Import von XML-Dokumenten. Wahlweise kann eine grafische Benutzungs-Oberfläche mit Preview-Funktion gestartet werden oder bei Angabe der Quell-Datei und der zugehörigen Filter-Definitions-Datei auf diese verzichtet werden.

Implementierung

Zunächst wird aus der XML-Datei ein DOM-Baum generiert. Dieser wird sodann rekursiv abgearbeitet. Anhand der Informationen aus der Filter-Definitions-Datei werden Entitäten in der Quell-Datei identifiziert. Die zugehörigen KHS-Objekte werden mittels aggregierter Instanzen der Klasse KhsObjectWrapper erzeugt. Es wird unterschieden zwischen Objekten, die *reservierten* Klassen, *Link*-Klassen oder *Standard*-Klassen („weder reserviert noch Link“) entstammen. Je nach Typ werden unterschiedliche Methoden aufgerufen (processReservedClass, processLinkClass, processStandardClass). Ein identifizierter Link hat, neben der Erzeugung des Link-Objektes, zur Folge, daß die betroffenen Objekte (Quelle/Ziel des Links) Referenzen auf das Link-Objekt erhalten.

Zentrale Methoden

ImportFilter (Konstruktor), extractForumFromXml, startGui, traverseDomBranch (private)

KhsObjWrapper

Beschreibung

Hilfsklasse der Klasse ImportFilter. Sie umwickelt KHS-Objekte. Da sie gleichzeitig auch KHS Objekte intern erzeugen kann (je nach verwendetem Konstruktor) hat sie auch Erzeugungs-Funktion[92]. Die Instanzvariablen eines umwickelten Objektes können aus dem Inhalt eines DOM-Teilbaumes, der aus dem XML-Quelldokument extrahiert wurde, gesetzt werden.

Implementierung

Das KHS-Objekt wird durch Aggregation umwickelt („Object Wrapper/Adapter“). Verschiedene Konstruktoren ermöglichen das Wrappen sowohl eines existierenden Objektes als auch eines anhand von Klassennamen oder ClassWrapper-Instanz zu erzeugenden Objektes. Zum Erzeugen von Objekten wird die Java-Klasse java.lang.Class eingesetzt, zum Setzen der MemberVariable das Java-Standard Paket java.lang.reflect.

Zentrale Methoden

KhsObjWrapper (Konstruktor in drei Varianten), setMemberVars

ReaderClassManager

Beschreibung

Instanzen dieser Klasse arbeiten der Instanz der Klasse FilterDefintionReader zu. Sie verwalten sämtliche Informationen einer KHS-Klasse, die in der Filter-Definitions-Datei spezifiziert sind (korrespondierende XML-Elemente, get/setMethoden …) .

[92] Wegen der Option Objekte erzeugen zu können, wäre statt „Wrapper“ alternativ die Bezeichnung „Builder“ möglich.

Implementierung

Die spezifischen Infomationen der KHS-Klasse werden in verschiedenen Hashtables, die teils benutzerdefinierte Datentypen (XmlContMethods, XmlContent) enthalten, abgelegt.

Zentrale Methoden

addDeclMethods, getXmlContMethods

6.6.2 Paket xml4Khs.gui

GuiManager

Beschreibung

Diese Klasse stellt die Methoden zum Aufruf der grafischen Oberfläche zur Verfügung. Dazu gehört eine Methode zur Anzeige eines „Datei-Öffnen“ Dialoges und eine Methode zum Starten der Benutzungsoberfläche (mit eingebauter Vorschau).

Implementierung

Der Datei-Öffnen-Dialog wird von der Java-Standard-Klasse java.awt.FileDialog zur Verfügung gestellt. Die Benutzungsoberfläche entspricht in leicht modifizierter Form der Klasse „TreeView“, die mit dem XML4J von IBM mitgeliefert wird.

Zentrale Methoden

openXmlFileDialog, openXmlViewer

TreeView

Beschreibung

Diese Klasse stellt die Benutzungsoberfläche mit eingebauter Vorschau zur Verfügung. Sie wurde von dem Parser Produkt Xml4J von IBM in leicht modifizierter Form übernommen.

Implementierung

Die Baum-Darstellung der Datei wird mittels der Java-SWING (1.1) Klasse Tree javax.swing.JTree erzeugt. Die interne Darstellung der Baumstruktur erfolgt in DOM-kompatibler Form.

Zentrale Methoden
XMLTreeView (Konstruktor)

6.6.3 Paket xml4Khs.util

In diesem Paket werden verschiedene Schnittstellen (ParseConstants, GeneralConstants) definiert, die sämtliche Konstanten des Importfilters enthalten. Ferner sind Utitilty-Klassen mit statischen Methoden, z. B. zur Konvertierung von Vektoren in String-Arrays enthalten.

6.6.4 Paket xml4Khs.standalone

Hier befinden sich Test-Programme und das besagte Programm „CreateFilterDef“ zur Erzeugung der Filter-Definitions-Datei.

6.7 Known limitations/ Erweiterungen

Die Filter-Implementation besitzt verschiedene Einschränkungen („Known limitations“). Vom Autor werden die folgenden Erweiterungen als sinnvoll erachtet:

1. **Unterstützung komplexer bzw. mehrdimensionaler[93] Datentypen** für Member-Variable. Derzeit werden lediglich simple eindimensionale Datentypen wie String, int, boolean … verarbeitet.
2. **Unterscheidung zwischen aggregierten und referenzierten Inhalten** von Member-Variablen. Im Moment werden nur Aggregationen akzeptiert, es sei denn, es wurde

[93] (Array, Vector, sonstige Collections/Sets)

explizit eine gesonderte Verarbeitung eingerichtet, wie dies beispielsweise bei KHS-Links der Fall ist.

3. **Validierung der Quell-Dokumente**: Hierzu bietet es sich an, die Syntax der Filter-Defintions-Datei an XML-Schemata anzupassen. Bei Verwendung der Schema-Sprache „W3 Schema“ [W3C 1999d; W3C 1999d], die vermutlich demnächst in einer standardisierten Version verabschiedet wird, kann die jüngst erschienene Version des IBM Parsers zur Validierung eingesetzt werden. Institutionen, die mit dem KHS-Projektteam kooperieren, sollte gegebenenfalls die Verwendung von XML-Schemata als Alternative zu DTDs empfohlen werden.
4. Erweiterung der **Integritätsprüfung** des Filters. Die derzeitige Integritätsprüfung beschränkt sich darauf, daß im Falle einer Verlinkung geprüft wird, ob die zu verlinkenden KHS-Objekte vorhanden sind
5. **Betriebsmodi**: Momentan ist ausschließlich das *Hinzufügen* von Hypertext-Objekten möglich. Es sollten auch die Betriebsarten Ändern, Überschreiben und Löschen implementiert werden – was auch mit einer entsprechenden Erweiterung der Interaktionsmöglichkeiten der Benutzungsoberfläche einher ginge.
6. **Export**: Der Export von Daten ist ebenfalls ein mögliches Ziel der Erweiterung.

6.8 Beschreibung der Testdaten

Validierung

Während in der ersten nicht-generischen Version des Importfilters eine Validierungsmöglichkeit mittels einer DTD zur Verfügung gestellt wurde, wurde bei der generischen Version darauf verzichtet. Der Grund hierfür ist, daß aufgrund des flexiblen Verhaltens des Filters beim Erstellen neuer Klassen *eine DTD ebenfalls neu generiert* werden müßte. Diese Problematik konnte durch eine Implementierung mittels eines neuen Programm-Moduls des XML4KHS aus Zeitgründen nicht mehr bewältigt werden.
Es gab jedoch auch einen guten Grund, diese Option zum gegenwärtigen Zeitpunkt nicht zu implementieren: Die Tatsache, daß anstatt DTDs besser XML-Schemata, aufgrund deren genannten Vorteile, zum Zwecke der Validierung, eingesetzt werden sollten. Da keines dieser Konzepte vom W3C als definitiver Standard verabschiedet wurde, ist eine Implementierung zum gegenwärtigen Zeitpunkt nicht sinnvoll.

Verknüpfung von Entitäten / Link-Typen

Aus Gründen (bislang) fehlender Notwendigkeit wurden die XML-Entitäten nicht mittels XLinks/Xpointern verknüpft. Die logische Verknüpfung von Entitäten läßt sich ausreichend mit der Verwendung von ID/IDREF-Attributen beschreiben. Ein Vorteil dieser Konstruktion ist zum einen die bereits vorliegende Standardisierung durch die XML 1.0 Sprachspezifikation, zum anderen die Möglichkeit der Prüfung durch den Parser, ob zu einer Link-„Quelle“ auch das Link-„Ziel“ vorhanden ist.

Zeichencodierung

Als Zeichencodierung für die XML-Files wurde die UNICODE UTF-8 Codierung gewählt.

Beispiel

Der Auszug einer „typischen“ Testdatei kann in Kapitel 3.2 nachgelesen werden – ebenfalls befindet sich ein Beispiel auf der beigefügten CD im Verzeichnis „\TestData“.

7 Oberfläche/Bedienung XML4KHS

Sobald der Import-Filter aufgerufen wird, startet die Benutzer-Oberfläche.

Es erscheint folgendes Bild:

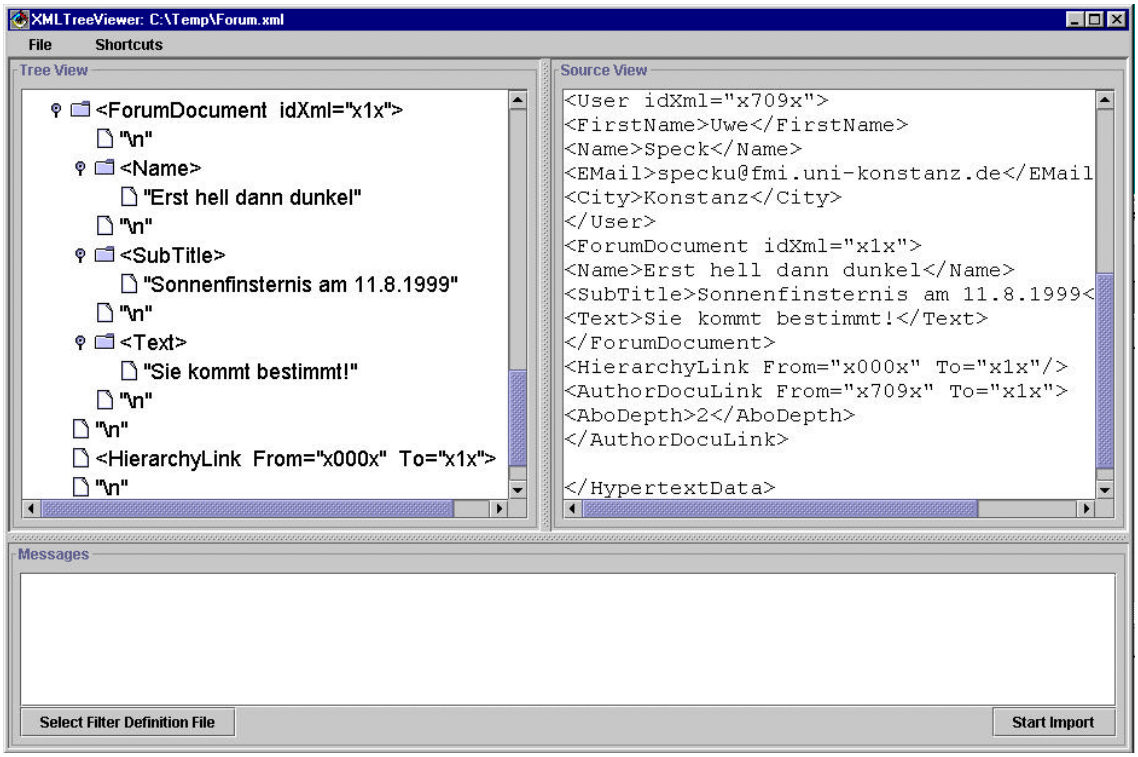


Abbildung 22 Benutzungsoberfläche des Importfilters. Quelle: eigen

Im linken Teilfenster wird eine Baum-Darstellung, im rechten Fenster eine Editor-ähnliche Ansicht der XML-Datei erzeugt. Im unteren Fensterbereich „Messages“ werden Verstöße gegen die Wohlgeformtheit angezeigt. Sollte das XML-Dokument eine DTD enthalten, so werden auch Validierungsfehler ausgegeben.

- Über den Menüpunkt File⇒Open kann eine Datei ausgewählt werden (Standard-Einstellung: C:\Temp\Forum.xml).

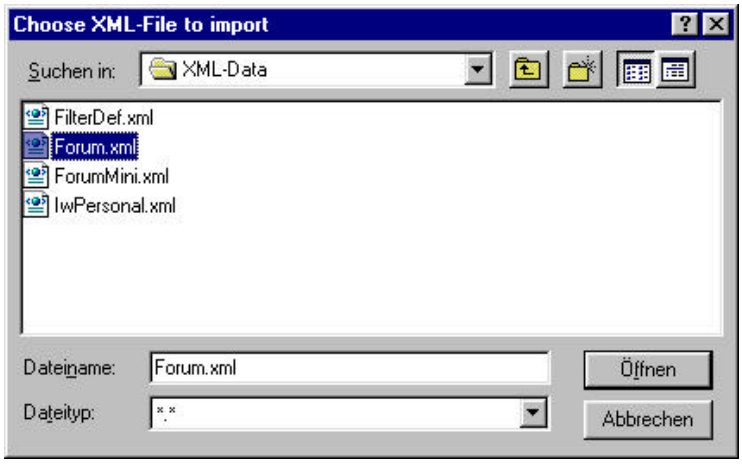


Abbildung 23 „XML-Datei-Öffnen“-Dialog. Quelle: eigen

- Durch Klick auf den Button „Select Filter Definition File“ erscheint analog ein Dialog zur Auswahl der Filter-Definitions-Datei.
- Der Klick auf den Button „Start import“ löst den Import-Vorgang aus.

Der Importvorgang kann im Prototyp des KHS wie folgt begutachtet werden:

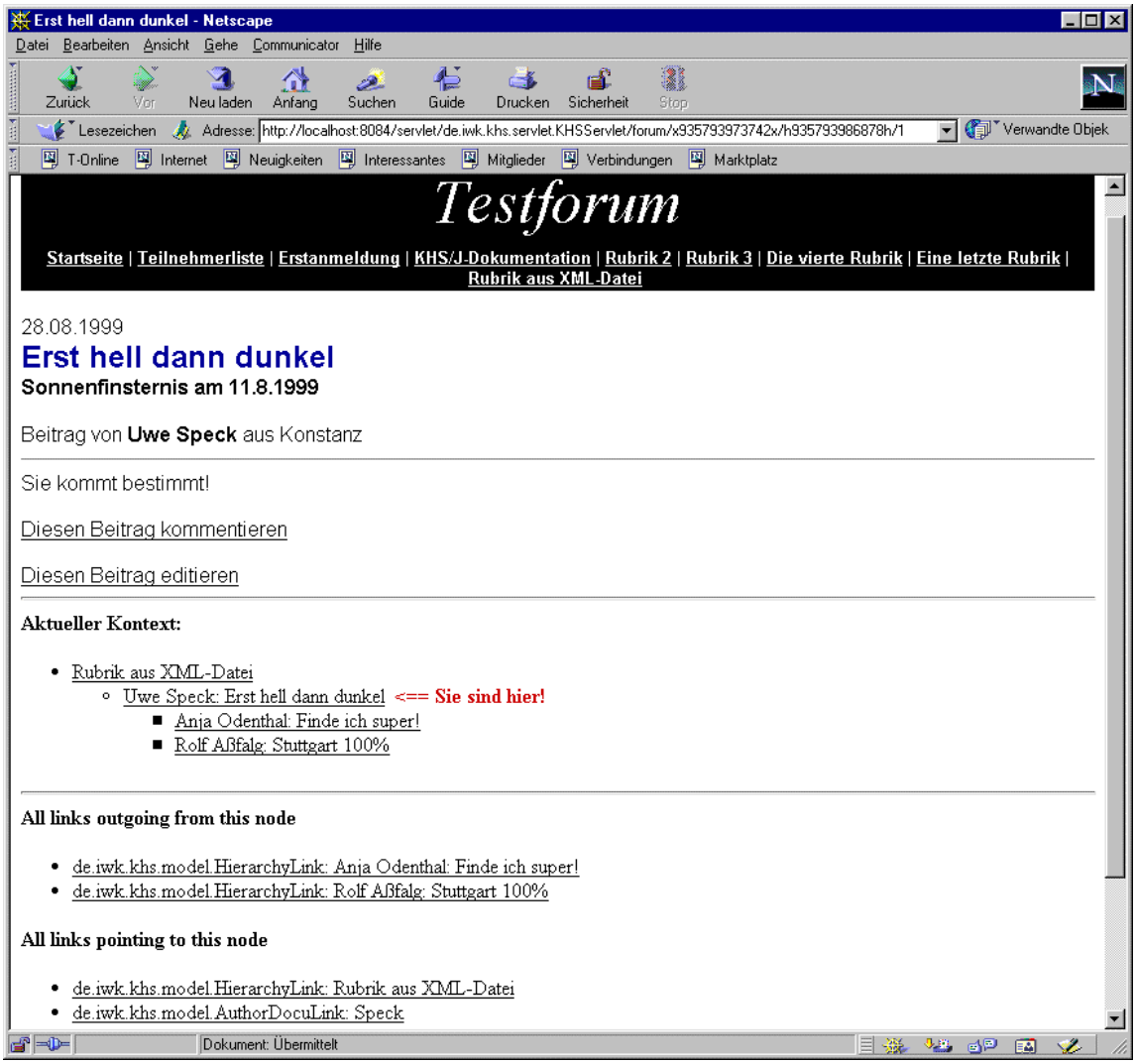


Abbildung 24 Screenshot: Importiertes Forums-Dokument. Quelle: eigen

Hier ist ein Forums-Artikel in Groß-Ansicht dargestellt. Im oberen Bereich („Testforum“) des Fensters sind die einzelnen Rubriken zu sehen, unter anderem die aus XML-importierte Rubrik „Rubrik aus XML-Datei“. Darunter ist der Forumsartikel zu erkennen. Unter der Überschrift „Aktueller Kontext“ ist der Diskussionsverlauf zu erkennen – darunter die Links die vom aktuellen Artikel ausgehen[95]. Eine analog aufgebaute Darstellung der importierten *Rubrik* findet sich auf folgender Seite.

[95] Die oben erkennbare Darstellung der Links ist nur im Prototyp des KHS enthalten und wird in der endgültigen Version entfernt.

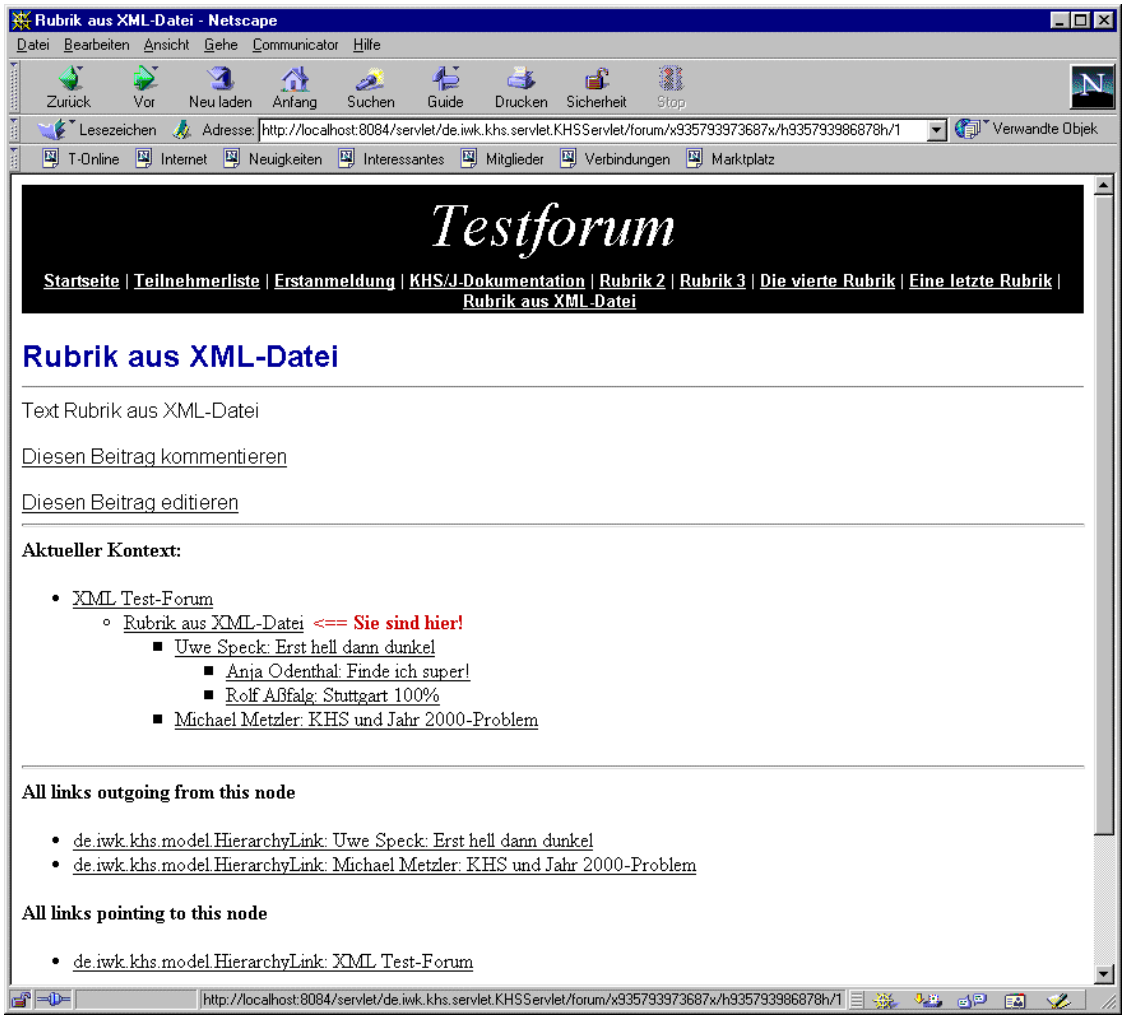


Abbildung 25 Screenshot: Importierte Rubrik. Quelle: eigen

8 Zusammenfassung und Vision

Zusammenfassung

Die zur Verarbeitung von XML-Dokumenten notwendigen Grundlagen wurden erarbeitet und mündeten in der erfolgreichen Implementierung eines Filters mit grafischer Benutzungsoberfläche, welcher mit Testdaten eines **Forums des KHS** erprobt wurde. Hervorzuheben ist dabei das sogenannte „generische Prinzip“ der Implementierung, das eine relativ leichte Anpassung des Filters an neue Klassen des KHS ermöglicht.

Auf dem Wege zur Implementierung wurde zunächst auf die Eigenschaften der *Extensible Markup Language* (XML), die ein „web-optimiertes SGML“ darstellt, eingegangen. Die Kernkomponenten (Inhalt/Struktur, Stil, Link) wurden mit unterschiedlichem Detaillierungsgrad - je nach Nähe zum geplanten Integrationsvorhaben – dargestellt:
Dabei stellte sich zunächst heraus, daß alternativ zu den in der XML-Sprachspezifikation 1.0 festgelegten Dokument-Typ Definitionen, auf deren Basis der Inhalt und die Struktur von Dokumenten definiert werden können, sogenannte XML-Schema-Sprachen zur Definition von XML-Applikationen eingesetzt werden können. Diese Sprachen, die sich jedoch noch auf dem Wege der Standardiserung befinden, sind aufgrund ihrer erheblichen Erweiterungen oft sogar vorzuziehen.
Desweiteren wurde im Kapitel „Hypertext und XML“ nachgewiesen, daß die Erweiterungen zu den Link-Möglichkeiten von HTML, dem derzeitigen WEB-Standard, nicht nur marginal, sondern essentiell sind (out-of-line Links, XPointer) – konkrete Fallbeispiele hierzu wurden recherchiert und aufbereitet. Ein ähnliche Überlegenheit von XML gegenüber HTML wurde in Bezug auf die Möglichkeiten der XML-Stilsprachen (XSL, DSSSL) anhand der integrierten Transformations- und Abfrage-Sprachen festgestellt.

Die bereits im Kapitel „Motivation“ aufgestellte These, daß XML eine zunehmende Bedeutung im online Bereich einnehmen wird, konnte auch durch die Tatsache der wachsenden Anzahl an peripheren XML-Komponenten belegt werden.

Im Kapitel 5 „XML-Prozessoren - Standard APIs“ erfolgte zunächst die Erörterung der vom W3C festgelegten Anforderungen an einen „XML-Prozessor“, einem Programm-Modul, das zur Analyse von XML-Dokumenten eingesetzt wird und deshalb oft auch als „XML-Parser“ bezeichnet wird.
Sodann wurden die beiden bedeutenden Standard-APIs, welche zur Herstellung von XML-Prozessoren eingesetzt werden, detailliert vor- und gegenüber gestellt. Während die baum-basierte DOM-API einen komfortablen Zugriff auf die Bestandteile eines XML-Dokumentes bietet, bietet die ereignis-gesteuerte SAX-API die Möglichkeit, ressourcen-schonende Prozessoren zu implementieren, die jedoch ein stark eingeschränktes Anwendungsgebiet besitzen. Die Ergebnisse der Gegenüberstellung führten zur Grundsatzentscheidung, für den Import der externen Daten in das KHS eine DOM-basierte API einzusetzen.
Der folgende Vergleich am Markt erhältlicher DOM-APIs hatte die Auswahl des Parsers „XML4J“ von IBM zur Folge, der lizenzfrei verwendet und vertrieben werden darf. Das Kapitel über XML-Parser wurde durch die Untersuchung der häufig anzutreffenden Symbiose von XML und der Programmiersprache Java, die auch beim zugrunde liegenden Projekt KHS/J eingesetzt wird, abgerundet. Hierbei stellte sich heraus, daß es sich bei dieser Kombination tatsächlich um eine *Schlüsseltechnologie* handelt.

Den letzten Teil stellte die Dokumentation der Realisierung des Importfilters, Arbeitstitel „XML4KHS“, dar:

Der XML4KHS stellt ein generisches Filterprinzip zur Verfügung, das sich an Klassen des KHS (halb-)automatisch anpaßt. Das heißt: Sind zu den Entitäten eines XML-Dokumentes, das eine gewisse Grundstruktur aufweisen muß, korrespondierende Klassen im KHS vorhanden, so ermöglicht der Filter einen Import der in „XML-Entitäten“ über ein sogenanntes „one-to-one-mapping“. Dabei wird auch die Semantik von Hypertext-Links berücksichtigt. Die notwendige Meta-Information für die Filterspezifikation, bestehend aus der Zuordnung der Entitätstyp-Eigenschaften zu den Eigenschaften der zugehörigen KHS-Klassen, wird in einer weitgehend *automatisch* generierten Datei, der sogenannten „Filter-Definitions-Datei“, festgehalten. Eine manuelle Anpassung dieser Datei ist nur in geringem Umfang erforderlich.

Mit Hilfe dieses Prototypen war es bereits möglich, wie oben erwähnt, die Daten eines KHS-Testforums zu importieren. Technologisch weist XML4KHS folgende Eigenschaften auf

Die ca. 1500 Code-Zeilen (locs)[96] umfassende Implementierung erfolgte in Java. Der Zugriff auf den Ableitungsbaum erfolgt ausschließlich über Methoden, die vom DOM spezifiziert sind. Hierdurch ist, falls erforderlich, die problemlose Substitution des verwendeten IBM-Parser-Produktes „XML4J“ durch ein anderes, DOM-kompatibles Werkzeug möglich. Die schema-ähnliche Filter-Definitions-Datei, welche in XML-codiert ist, wird mittels proprietärer Schnittstellen-Methoden des XML4J erzeugt.

Ausblick

Die Implementierung des „Xml4KHs“ kann aufgrund ihres modularen Aufbaus als Framework für künftige Arbeiten eingesetzt werden. Wie bereits im Kapitel „Known limitations/ Erweiterungen“ angeführt bieten sich verschiedene Erweiterungen an. In Stichworten ausgedrückt:

- **Member Variable:** Unterstützung komplexer bzw. mehrdimensionaler[97] Datentypen/ Unterscheidung zwischen aggregierten und referenzierten Inhalten..
- **Validierung der Quell-Dokumente**
- Erweiterung der **Integritätsprüfung**.
- **Betriebsmodi**: Ändern, Überschreiben und Löschen
- **Export** von Daten.

Vision

Obwohl der derzeit implementierte Filter-Ansatz, der mittels der oben erläuterten Maßnahmen beliebig erweitert werden kann, aufgrund des generischen Prinzips flexibel auf

[96] Die Teile der verwendeten Parser und Java-Standardbibliotheken sind beim Umfang von 1500 Zeilen selbstverständlich ausgenommen.

[97] (Array, Vector, sonstige Collections/Sets)

neue Klassenstrukturen des KHS reagieren kann, wurde dem Autor im Laufe des Entwicklungsprozesses bewußt, daß Erweiterungen auf *Seiten des KHS* ebenfalls sinnvoll sein könnten; vorrangig um die Flexibilität in Bezug des Systems auf die Integration von xml-codierten Hypertexten zu erhöhen.

Dieses Problem sei anhand einer komplexen XML-Entität illustriert:

Beispielsweise wird ein *linguistischer Lexikonartikel*, innerhalb seines Erklärungs-Textes im allgemeinen eine *Vielzahl* von *geschachtelten* XML-Elementen aufweisen, die auch mit Attributen versehen sein können. Mittels der momentan im KHS implementierten Struktur, die sich im Bereich elektronischer Kommunikationsforen bewährt hat, würde der Artikel-Text in der Member-Variable eines Objektes der Klasse „Lexikonartikel“ abgelegt. Die Zuordnung einer geschachtelten Folge von XML-Elementen zu einer *einzigen* Member-Variable legt zunächst zwei Repräsentationen nahe, die jedoch problembehaftet sind:

- Wird die Xml-Codierung der Artikel-Textes als *String* in der Variable abgelegt, so ist zwar kein Informationsverlust vorhanden, jedoch wird dabei beispielsweise die Semantik von Links, welche innerhalb des Artikel-Textes bei dieser internen Darstellung nicht berücksichtigt. Obwohl die Semantik bei der Bildschirm-Ausgabe des Artikels mittels eines XML-Frontends („View“) *decodiert* werden kann, ist diese Lösung aufgrund der Divergenz der internen Repräsentationen von Model und View nicht befriedigend.

- Wird zur Aufnahme ein statisches Objekt verwendet, so stößt dies aufgrund der mannigfaltigen Varianten von Reihenfolge und Schachtelung, welche die Elemente/ Attribute innerhalb des Artikel-Textes annehmen können, auf Grenzen.

Alleine eine *dynamische* Objektstruktur schafft hier einen Ausweg. Eine derartige Struktur sollte jedoch nach Ansicht des Autors nicht auf der Ebene einzelner Member-Variablen-Inhalte sondern auf der Ebene der Gesamt-Struktur der KHS-Objekte eingesetzt werden.

Derartige Strukturen, die speziell der Aufnahme von SGML-/XML-Daten dienen, werden im Artikel von [Aberer, Böhm, Hüser 1993] beschrieben. Das Grundprinzip ist dabei die

Überführung der Elemente/Attribute und deren Inhalte in eine *baum-artige* Struktur von Objekten. *Links* werden in diesem Ansatz über die Implementierung der entsprechenden Anwendungslogik in den Instanzen der Link-Bestandteile des Dokumentbaumes implementiert [Aberer, Böhm, Hüser 1993, Kap. „Hytime“].

Als Verfeinerung dieses, bereits in der Diplomarbeit von Toussi ([Toussi 1999, S. 58]) favorisierten Lösungsansatzes, empfiehlt der Autor das im Rahmen dieser Arbeit dokumentierte und in dem Importfilter implementierte *Document Object Model* (DOM) einzusetzen.
Das DOM stellt eine *vom W3C standardisierte Umsetzung* des Konzeptes von [Aberer, Böhm, Hüser 1993] dar. Die zur Verarbeitung von Links geforderte Integration von Anwendungslogik in den Objektbaum wird dabei mit sogenannten XML-Beans (vgl. S. 52) realisiert. Die Persistenz dieser Daten kann mit Hilfe eines OO-DBMS leicht erreicht werden – eine Technik, die in Verbindung mit XML allgemein als „State of the Art“ angesehen wird.

Nach Ansicht des Autors sollte deshalb geprüft werden, ob die Integration des DOM in das KHS zur Aufnahme, hochvernetzter, in XML codierter Daten sinnvoll ist. Der auf Basis des DOM implementierte Importfilter „XML4JHS“ könnte dabei als technologische Studie herangezogen werden.

9 Anhang

9.1 Parser Technologie: Fachtermini

9.1.1 Gültigkeit und Wohlgeformtheit von XML Dokumenten

Nach dem eben Erläuterten lassen sich XML-Dokumente - in Bezug auf ihre lexikalische und syntaktische Struktur - wie folgt klassifizieren.

Die Definitionen aus der XML 1.0 Sprachspezifikation [W3C 1998e] lauten wie folgt:

„2.1 Well-Formed XML Documents

A textual object is a well-formed XML document if:

1. Taken as a whole, it matches the production labeled document.
2. It meets all the well-formedness constraints given in this specification.
3. Each of the parsed entities which is referenced directly or indirectly within the document is wellformed.

Document
[1] document ::= prolog element Misc*

Matching the document production implies that:

1. It contains one or more elements.
2. There is exactly one element, called the **root**, or document element, no part of which appears in the content of any other element. For all other elements, if the start-tag is in the content of another element, the end-tag is in the content of the same element. More simply stated, the elements, delimited by start- and end-tags, nest properly within each other.

As a consequence of this, for each non-root element **C** in the document, there is one other element **P** in the document such that **C** is in the content of **P**, but is not in the content of any other element that is in the content of **P**. **P** is referred to as the **parent** of **C**, and **C** as a **child** of **P**“. [W3C 1998e, Kap. 2.1]

„**2.8 Prolog and Document Type Declaration**

An XML document is valid if it has an associated document type declaration and if the document complies with the constraints expressed in it“. [W3C 1998e, Kap. 2.8]

9.1.2 Syntaktische Analyse/ Parser

[Duden 1993][100] definiert den Begriff des Parsers folgendermaßen:

> “In einem Übersetzer hat die syntakische Analyse die Funktion, ein Quellprogramm [oder Quelldokument], dargestellt als Folge von Token[101] (diese Folge wird durch die lexikalische Analyse geliefert), in einen Ableitungsbaum zu übertragen. Gleichzeitig wird untersucht, ob das Quellprogramm syntaktisch korrekt ist, also der zugrundeliegenden Grammatik entspricht. Gegebenenfalls werden Fehler angezeigt. Ein Programm, das diese Funktionen erfüllt, nennt man Parser (dt. = Zerteiler). Ein Parser liest die zum Quellprogramm gehörende Tokenfolge ein, meldet eventuelle Fehler und gibt den dazugehörigen Ableitungsbaum aus. … Der Ableitungsbaum ist somit das Bindeglied zwischen syntaktischer und semantischer Analyse”.

---

[98] mathematisch ausgedrückt: es kann eine DTD gefunden werden, zu der das Dokument gültig ist.

[99] mathematisch ausgedrückt: es kann *keine* DTD gefunden werden, zu der das Dokument gültig ist.

[100] (Schlagwort “Syntaktische Analyse”)

[101] [Duden 1993] Schlagwort “Lexikalische Analyse”: “ Ein Token bezeichnet eine Folge von Zeichen, die bedeutungsmäßig zusammengehören.

9.2 XML-Prozessor Benchmark

Clark Cooper [Cooper 1999] verglich verschiedene Prozessor-Implementationen der Programmiersprachen C, Java, Perl, und Python auf einem Linux System (Redhat 5.2, Hardware PII mit 64MB). Dabei wurde ein Testprogramm namens “humble XMLstats program”, das einen statistischen Report über die *Elemente eines XML-Dokumentes* ausgibt, verwendet. Die Prozessoren wurden unter Anwendung einer ereignis-basierten API (SAX oder proprietär) betrieben. Zum Test wurden verschiedengroße XML-Files herangezogen, die folgende Eigenschaften aufweisen.

Test document statistics

	REC	chrmed	med	chrbig	big
size (bytes)	159339	893821	1264240	3417181	5052472
markup density	34%	6%	33%	2%	33%

Abbildung 26: Eigenschaften der getesteten Dateien, Quelle: [Cooper 1999]

9.2.1 Vergleich: Gesamt

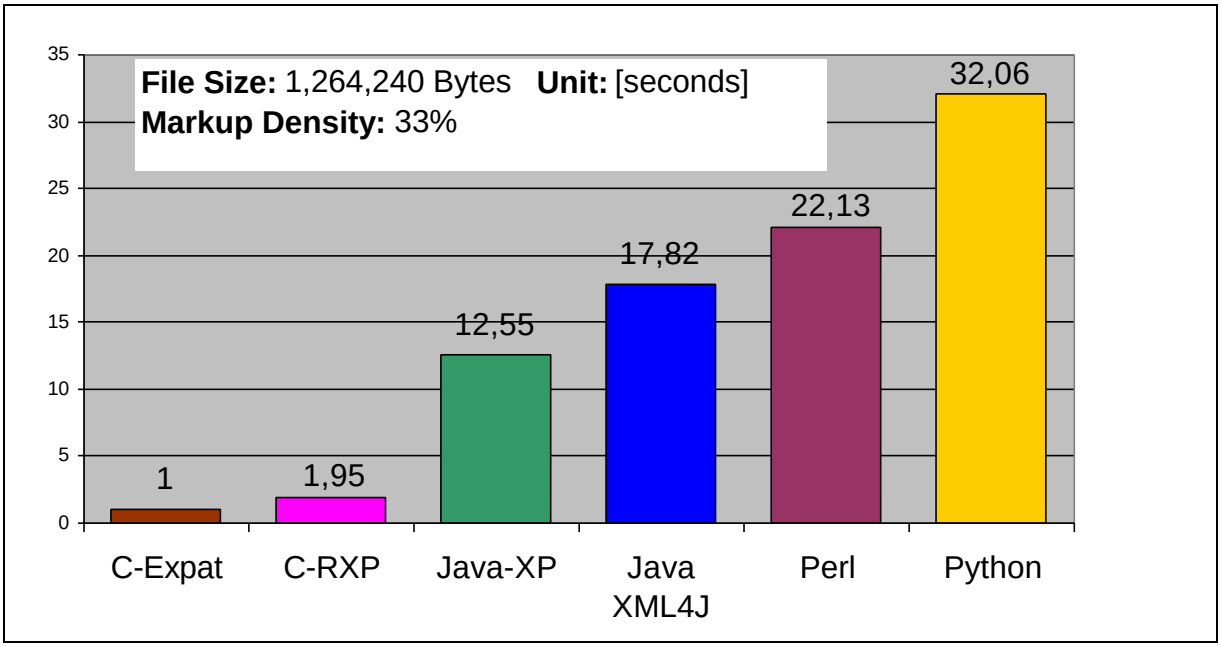


Abbildung 27: Parser Benchmark Gesamtvergleich Quelle: modfiziert nach [Cooper 1999]

Legende: **X-Achsen-Teilung:** C: C-Expat (Version 19990307) , RXP (Version 1.0), Java: JAVA-XP (Version 0.4), JAVA-XML4J (Version 2.0.2) , Perl: XML::Parser (Version 2.23), Python: Pyexpat (0.5 Distribution) **Y-Achsen-Teilung**: Verarbeitungszeit, normiert auf den schnellsten Parser: C-Expat

Obwohl Cooper einräumt

> "These tests only measure execution performance. Note that sometimes *programmer performance* is more important than parser performance. I have no numbers, but I can report that for ease of implementation, the Perl and Python programs were easiest to write, the Java programs less so, and the C programs were the most difficult".

fällt zunächst eine grobe Tendenz deutlich auf: Am schnellsten sind die C-Implementierungen (Compiler), gefolgt von den Java-Produkten (Bytecode-Interpreter). Das Schlußlicht bilden die Skriptsprachen (Interpreter).

Die Ergebnisse von Cooper wurden zusätzlich unter zwei Gesichtspunkten tabellarisch und grafisch aufbereitet. Von Interesse waren die Gegenüberstellungen:

a) Gleiche Implementierung – unterschiedliche Programmiersprache
b) Gleiche Programmiersprache – unterschiedliche Implementierung

9.2.2 Vergleich: Unterschiedliche Programmiersprache

Vier der teilnehmenden Produkte (Java XP, C-Expat, Perl XML::Parser und Pyexpat) beruhen auf der Arbeit eines *einzigen* Programmierers, James Clark. Da bei diesen Implementierungen von einer fast identischen algorithmischen Konzeption der Parser auszugehen ist, treten die Performance-Eigenschaften der *Plattformen* hervor. Normiert man die Ergebnisse des schnellsten Prozessors auf 1, so erhält man folgende Vergleichtabelle der Parser, die auf EXPat beruhen.

Tabelle 2: Parser Benchmark gesamt Quelle: modifiziert nach [Cooper 1999]

Produkt	REC	Chrmed	Med	Chrbig	big
C-Expat	1,00	1,00	1,00	1,00	1,00
JAVA-XP	48,00	24,48	12,55	11,79	8,50
Perl	28,26	31,09	22,13	31,62	21,86
Python	33,00	43,61	32,06	46,74	32,75

9.2.3 Vergleich: Unterschiedliche Implementierung

Programmiersprache Java

Tabelle 3: Parser Benchmark bei Programmiersprache Java Quelle: modifiziert nach [Cooper 1999]

Produkt	REC	Chrmed	med	Chrbig	big
Java-XP	1,00	1,00	1,00	1,00	1,00
Java-Xml4J	1,26	1,29	1,42	1,32	1,53

Programmiersprache C

Tabelle 4: Parser Benchmark bei Programmiersprache C Quelle: modifiziert nach [Cooper 1999]

Produkt	REC	Chrmed	med	Chrbig	big
C-Expat	1,00	1,00	1,00	1,00	1,00
C-RXP	2,00	2,91	1,95	3,12	1,98

9.2.4 Interpretation der Ergebnisse

Der Performance-Vergleich anhand der "mittelgroßen" Datei med[102], deren Meßwerte in den vorangegangenen Tabellen grau unterlegt sind, liefert folgende „typische" Performance-Faktoren.

Tabelle 5: Interpretation der Parser Benchmark Ergebnisse Quelle: modifiziert nach [Cooper 1999]

Kombination	Produkte	Typischer Faktor
Identische Programmiersprache (Java) Unterschiedliche Implementierung	Java-XP / Java Xml4J	1,42
Identische Implementierung (Expat von J. Clark) Unterschiedliche Programmiersprache	Java-XP/ C-Expat C	12,6

[102] „med" wie: medium – mittlere Dateigröße (1,3 MB, Markup-Dichte 33%)

Es kann davon ausgegangen werden, daß im Bereich (semi-)professioneller Parser-Produkte die Implementierung bzw. deren zugrunde liegender Algorithmus vergleichbar ist. Wie oben beispielhaft vorgeführt, ist mit einem “typischen” Performancefaktor von 1,42 ein wesentlich geringerer Einfluß auf die Performance vorhanden, als der, welcher von der verwendeten Programmiersprache (=Plattform) ausgeht. Hier liegt ein typischer Faktor von 12,6 vor.

9.3 W3C – Technical Reports

Das W3Consortium klassifiziert die unterschiedlichen Empfehlungen bzw. Normen wie folgt:
Auszug aus: [W3C 1999b]

„About W3C Technical Reports

As described in the Process Document, W3C publishes several types of technical reports:

Notes

A Note is a dated, public record of an idea, comment, or document. A Note does not represent commitment by W3C to pursue work related to the Note.

Working Drafts

A Working Draft represents work in progress and a commitment by W3C to pursue work in this area. A Working Draft does not imply consensus by a group or W3C.

Proposed Recommendations

A Proposed Recommendation is work that (1) represents consensus within the group that produced it and (2) has been proposed by the Director to the Advisory Committee for review.

Recommendations

A Recommendation is work that represents consensus within W3C and has the

> Director's stamp of approval. W3C considers that the ideas or technology specified by a Recommendation are appropriate for widespread deployment and promote W3C's mission.

Specifications developed within W3C must be formally approved by the Membership. Consensus is reached after a specification has proceeded through the review stages of Working Draft, Proposed Recommendation, and Recommendation“.

9.4 Code-Beispiel für die SAX-API

Filename: MyHandler.java

```
package SaxBasedDocInfo;

import org.xml.sax.HandlerBase;
import org.xml.sax.AttributeList;

/**
 * This class specifies objects which hold the event handlers for
 * common document events
 *
 * @author Uwe Speck (1999)
 */

public class MyHandler extends HandlerBase {

/**
 * This method is called if characters occur in XML-File
 *
 * Task: displays characters
 *
 * @param ch char[] - character array
 * @param start int - start index of character data
 * @param length int - endindex of character data
 */

public void characters(char[] ch, int start, int length) {

      //Create empty String buffer to hold character data
      StringBuffer strBuff = new StringBuffer("");

      //put character data into  String buffer
      strBuff.append(ch, start,length);

      //display String buffer
      System.out.println("Characters: " + strBuff);

}
/**
 * This method is called if an end element occurs  XML-File
 *
 * Task: diplays element name
 *
 * @param name String - name of element
 */

public void endElement (String name)
  {
      //display element name
      System.out.println("End element: " + name);
  }
/**
 * This method is called if a start element occurs  XML-File
 *
 * Task: diplays element name and attribute data
 *
```

```
 * @param name String - name of element
 * @param atts AttributeList - attribute data array
 */


public void startElement (String name, AttributeList atts)
{

      //process elements of type "name" separately

      if (name.equals("name")) {

            //display message, thate name element was found
            System.out.println("Yeah! A NAME started! " + name);

      } else {

            //display element name
            System.out.println("Start element: " + name);

      }


      //step through all attributes and display them

      for (int i = 0; i < atts.getLength(); i++) {

             //Get attribute name
             String attName = atts.getName(i);

             //Get attribute type
             String attType = atts.getType(i);

             //Get attribute type
             String attValue = atts.getValue(i);

             //display attribute data
             System.out.println("Attribute found *** Name: " + attName
                                          + "  Type: " + attType + "
Value: " + attValue);

   }


}
}
```

Filename: MyErrHandler.java

```
package SaxBasedDocInfo;

import org.xml.sax.HandlerBase;
import org.xml.sax.AttributeList;
import org.xml.sax.SAXParseException;

/**
 * This class specifies objects which hold the event handlers for
 * parse errors and warnings
 *
```

```
 * @author Uwe Speck (1999)
 */

public class MyErrHandler extends HandlerBase {

/**
 * This method is called if a (recoverable) error occurs while parsing
XML-File
 *
 * Task: diplays line number, column number and kind of exception
 *
 * @param e SAXParseException - Exception which was thrown by parser

 */


public void error(SAXParseException e) {

 showParseErrorData(e, "Error");

 }
/**
 * This method is called if a (NOT recoverable) error occurs while parsing
XML-File
 *
 * Task: diplays line number, column number and kind of exception
 *
 * @param e SAXParseException - Exception which was thrown by parser

 */

 public void fatalError(SAXParseException e) {

       showParseErrorData(e, "Fatal Error");

 }
/**
 * This method displays parse error/warning data
 *
 * Task: diplays line number, column number and kind of exception
 *
 * @param e SAXParseException - Exception which was thrown by parser
 * @param pStrMessage String - individual Message to specify kind of
error/warning
 */

public void showParseErrorData(SAXParseException e, String pStrMessage) {

 showParseErrorData(e, "Warning");

}
 /**
 * This method is called if a warning occurs while parsing XML-File
 *
 * Task: diplays line number, column number and kind of exception
 *
 * @param e SAXParseException - Exception which was thrown by parser

 */
```

```
public void warning(SAXParseException e) {

      showParseErrorData(e, "Warning");


 }
}
```

Filename: SaxBasedDocInfo.java

```
package SaxBasedDocInfo;

import org.xml.sax.Parser;
import org.xml.sax.DocumentHandler;
import org.xml.sax.ErrorHandler;
import org.xml.sax.helpers.ParserFactory;

/**
 * This class provides the runtime ability of SAX example
 * SAX based Document Info
 *
 * This code example was derived
 *
 * from D. Megginsons "SAX Quick Start"
http://www.megginson.com/SAX/quickstart.html
 *
 *
 * @author Uwe Speck (1999)
 */

public class SaxBasedDocInfo {

  //Parser Class name set to IBM-Parser XML4J class
  static final String parserClass = "com.ibm.xml.parser.SAXDriver";

/**
 * Main method
 *
 *
 * @author Uwe Speck (1999)
 */

  public static void main (String args[]) {

      try {

      //create parser   object
      Parser parser = ParserFactory.makeParser(parserClass);

      //create instance for document event handlers
      DocumentHandler handler = new MyHandler();

      //create instance for error/warning event handlers
      ErrorHandler errHandler = new MyErrHandler();

      //assign document handler to parser instance
      parser.setDocumentHandler(handler);

      //assign error handler to parser instance
      parser.setErrorHandler(errHandler);
```

```
      for (int i = 0; i < args.length; i++) {

            //parse all files given as command line arguments
        parser.parse(args[i]);
      }

      }     catch (Exception e) {

            //runtime error, which was not handled,  occurred
            System.out.println("Hmmh, ... There's something wrong" + e);


      }


  }
}
```

9.5 Abbildungsverzeichnis

9.6 Tabellenverzeichnis

9.7 Literaturverzeichnis

Aberer, Böhm, Hüser 1993

ABERER, Karl; BÖHM, Klemens; HÜSER, Christoph: The Prospects of Publishing Using Advanced Database Concepts. In: *ELECTRONIC PUBLISHING; ORIGINATION, DISSEMINATION AND DESIGN, 1993. JG. (1993), H. 6/4, S. 469-480.*

Aho et al. 1990

AHO, Alfred V.; SETHI, Ravi; ULLMANN, Jeffrey D. (Hrsg.): Compilerbau - Teil 1 (ADDISON-WESLEY) 1988.

Balzert 1996

BALZERT, Helmut (Hrsg.): Lehrbuch der Software-Technik, Bd. I Heidelberg - Berlin - Oxford (SPEKTRUM AKADEMISCHER VERLAG GMBH) 1996.

Bannert, Weitzel 1999

BANNERT, Gabriele; WEITZEL, Martin (Hrsg.): Objektorientierter Softwareentwurf mit UML Bonn (ADDISON WESLEY) 1999.

Behme 1999

BEHME, Henning: Wozu die extensible Markup Language gut ist In: *IX - DAS MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK, 1999. JG. (1999), H. 1999, S. 36-41.*

Behme, Mintert 1998

BEHME, Henning; MINTERT, Stefan (Hrsg.): XML in der Praxis (ADDISON WESLEY) 1998.

Bosak 1997

BOSAK, John: XML, Java, and the future of the Web URL: http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm 1997.

Bourret 1999

BOURRET, Ronald: XML Schema Languages (Power Point Präsentation) http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/bourret.htm 1999.

Buschmann et al. 1999

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, Michael (Hrsg.): Pattern-orientierte Software-Architektur - Ein Pattern System (ADDISON WESLEY) 1999.

Cooper 1999

COOPER, Clark: ***Benchmarking XML Parsers - A performance comparison of six stream-oriented XML parsers URL: http://www.xml.com/xml/pub/Benchmark/article.html 1999***.

DuCharme 1999

DUCHARME, Bob (Hrsg.): ***XML- the annotated Specification (PRENTICE HALL) 1999***.

Duden 1993

CLAUS, Volker; *SCHWILL, Andreas (Hrsg.)*: ***Duden - Informatik (DUDENVERLAG) 1993***.

Fuchs 1999

FUCHS, Matthew: ***Why XML Is Meant for Java - Exploring the XML/Java Connection Online: http://www.xml.com/xml/pub/1999/06/fuchs/fuchs.html*** In: ***WEB TECHNIQUES, 1999. JG. (1999)***, [Seitenangaben fehlen!]

Gamma et al. 1996

GAMMA, E.; ***HELM, RICHARD,JOHNSON, R. VLISSIDES, J. (Hrsg.)***: ***Entwurfsmuster (ADDISON WESLEY) 1996***.

Goldfarb 1998

GOLDFARB, Charles (Hrsg.): ***Future Dimensions in XML in: Möhr/Schmidt "SGML und XML" (SPRINGER) 1999***.

Goldfarb, Prescod 1999

GOLDFARB, Charles; *PRESCOD, Paul (Hrsg.)*: ***XML Handbuch (PRENTICE HALL) 1999***.

Heise 1999

HEISE VERLAG [HRSG.]: ***XML and EDI - XML/ EDI page http://ftp.heise.de/ix/raven/Web/xml/edi.html 1999***.

Hueskes 1997

HÜSKES, Ralf: ***Dynamisches HTML: Inkompatibilität zwischen Microsoft und Netscape - Mein Web gehört mir*** In: ***IX - DAS MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK, 1997. JG. (1997), H. 8, S. 140-143***.

IBM 1998

AKERLEY, J.; *LI, N.*; *PARLAVECCHIA, A. (Hrsg.)*: ***Programming with Visulage for Java Version 2 URL: http://www.redbooks.ibm.com 1998***.

IBM 1999

IBM CORP. [HRSG.]: ***IBM XML4J License Agreement URL: http://www.alphaworks.ibm.com/registerXML4J 1999***.

IBM 1999a

IBM CORP. [HRSG.]: *XML Parser for Java API 1999*.

Koch 1999

KOCH, *Stefan (Hrsg.)*: *JavaScript - Einführung, Programmierung, Referenz - 2. aktual. und erw. Auflage (DPUNKT.VERLAG) 1999*.

Kuhlen 1996

KUHLEN, *Rainer (Hrsg.)*: *Informationsmarkt (UNIVERSITÄTSVERLAG KONSTANZ (UVK)) 1996*.

Kuhlen 1997

KUHLEN, *Rainer (Hrsg.)*: *Hypertext - in: Buder, M.; Rehfeld, W.; Seeger, T.;Strauch, D. [Hrsg.] "Grundlagen der praktischen Information und Dokumentation" (KG SAUR) 1997*.

Laurent 1999

ST. LAURENT, *Simon*: *Toward a Layered Model for XML URL: http://www.simonstl.com/articles/layering/layered.htm (Stand: 5.8.1999) 1999*.

Laurent, Cerami 1999

ST. LAURENT, *Simon*; *CERAMI*, *Ethan (Hrsg.)*: *Building Xml Applications (COMPUTING MCGRAW-HILL) 1999*.

Ludwig 1999

LUDWIG, *Justin*: *An investigation of XML with Emphasis on Extensible Linking Language (XLL) College of Wooster 1999, independent study thesis - URL: http://pages.wooster.edu/ludwigj/xml/thesis.html (Stand 5.5.1999) ALS MANUSKRIPT GEDRUCKT.*

Macherius 1997

MACHERIUS, *Ingo*: *XML - Professionelle Alternative zu XML* In: *IX - DAS MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK*, *1997. JG. (1997)*, *H. 6*, *S. 106 FF.*.

Maler 1998

MALER, *Eve*: *Re: [xX][mM][lL] is reserved - Beitrag aus der xml-dev Mailing List: (xml-dev@ic.ac.uk) vom 5. Mai 1998 http://www.lists.ic.ac.uk/hypermail/xml-dev/xml-dev-May-1998/0100.html 1998*.

Maruyama et al. 1999

MARUYAMA, *Hiroshi*; *TAMURA*, *Kent*; *URAMOTO*, *Naohiko (Hrsg.)*: *XML and Java - Developing Web Applications (ADDISON WESLEY) 1999*.

Meyer 1998

MEYER, *André (Hrsg.)*: *JFC 1.1 mit Java Swing 1.0 - Ein Tutorial für die Gestaltung grafischer Benutzerschnittstellen (ADDISON WESLEY) 1998*.

Middendorf 1999

MIDDENDORF, Stefan: *XML-Verarbeitung mit Java - Wohlgeformte Bohnen* In: *IX - DAS MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK, 1999. JG. (1999)*, [Seitenangaben fehlen!]

Muenz 1999

MÜNZ, Stefan: *SELFHTML URL: http://www.teamone.de/selfhtml/ 1999*.

Naughton, Schildt 1999

NAUGHTON, PATRICK, SCHILDT, HERBERT (Hrsg.): *Java 2: The complete reference, 3rd Edition (OSBORNE/ MCGRAW-HILL) 1999*.

Object Arts 1999

OBJECT ARTS [HRSG.]: *Model-View-Controller framework http://www.object-arts.com/EducationCentre/Overviews/MVC.htm 1999*.

Odenthal, Aßfalg, Handschuh 1998

ODENTHAL, Anja; AßFALG, Rolf; HANDSCHUH, Siegfried: *KHS/J Ein java-basiertes, offenes Hypertextsystem für WWW-Anwendungen Konstanz 1998*.

Oestereich 1999

OESTEREICH, Bernd: *Glossar Objektorientierte Grundbegriffe Begriffe der UML (Unified Modeling Language) UML-Notationsübersicht http://www.oose.de/download/downl.htm 1998*.

Pardi 1999

PARDI, William J. (Hrsg.): *XML in Action - Dynamische und datangestützte Webseiten mit der neusten Web-technologie (MICROSOFT PRESS) 1999*.

Speck, Toussi 1998

SPECK, Uwe; TOUSSI, Mark: *Realisierung eines regionalen politischen Diskussionsforums im Vorfeld der Bundestagswahl 98 auf Basis des Konstanzer Hypertext Systems (KHS) - Projektkurs im Studiengang Informationswissenschaft WS 97/98 Konstanz 1998*.

SUN 1998

SUN MICROSYSTEMS [HRSG.]: *JAVA Development Kit Documentation 1.1.6 - integriert in IBM Visual Age for Java 2.0 auch: http://java.sun.com/products/jdk/1.1/docs.html 1998*.

Toussi 1999

TOUSSI, Mark: *Entwicklung und Abbildung einer Dokument Typ Definition auf ein Hypertext System mit objektorientierter Datenbank Konstanz, Diplomarbeit FG Informationswissenschaft, WS 1998/1999 1999 ALS MANUSKRIPT GEDRUCKT.*

Unicode 1998

UNICODE CONSORTIUM [HRSG.]: *The Unicode Standard - A Technical Introduction http://www.unicode.org/unicode/standard/principles.html 1998*.

VEO 1999

VEO SYSTEMS [HRSG.]: *Frequently asked questions about SOX http://www.veosystems.com/xml/sox/sox_faq.html 1999*.

W3C 1997

JAMES CLARK: *Comparison of SGML and XML - World Wide Web Consortium Note 15-December-1997 http://www.w3.org/TR/NOTE-sgml-xml-971215 1997*.

W3C 1998a

W3 CONSORTIUM [HRSG.]: *XML Linking Language (Xlink) - Working Draft 1998-03-03 URL:http://www.w3.org/TR/1998/WD-xlink-19980303 1998*.

W3C 1998c

W3 CONSORTIUM [HRSG.]: *XML Pointer Language (XPointer) - Working Draft 03-March-1998 URL: http://www.w3.org/TR/WD-xptr-19980303 (Stand 15.06.1999) 1998*.

W3C 1998d

W3 CONSORTIUM [HRSG.]: *Document Object Model (DOM) Level 1 Specification Version 1.0 W3C recommendation 1 October, 1998 URL: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/ 1998*.

W3C 1998e

W3 CONSORTIUM [HRSG.]: *Extensible Markup Language (XML) 1.0 - W3C Recommendation 10-February-1998 URL: http://www.w3.org/TR/1998/REC-xml-19980210 1998*.

W3C 1998f

W3 CONSORTIUM [HRSG.]: *Schema for Object oriented XML - NOTE-SOX-19980930 http://www.w3.org/TR/1998/NOTE-SOX-19980930/ 1998*.

W3C 1998g

W3 CONSORTIUM [HRSG.]: *XML-Data W3C Note 05 jan 1998 - Latest version: http://www.w3.org/TR/1998/NOTE-XML-data http://www.w3.org/TR/1998/NOTE-XML-data-0105/ 1998*.

W3C 1999b

W3 CONSORTIUM [HRSG.]: *W3C Technical Reports and Publications URL: http://www.w3.org/TR 1999*.

W3C 1999d

W3 CONSORTIUM [HRSG.]: *XML Schema Part 1: Structures - W3C Working Draft 6-May-1999 http://www.w3.org/1999/05/06-xmlschema-1/ 1999*.

W3C 1999e

W3 CONSORTIUM [HRSG.]: *XML Schema Part 2: Datatypes - W3C Working Draft 6-May-1999 http://www.w3.org/1999/05/06-xmlschema-2/ 1999*.

W3C 1999f

***W3 CONSORTIUM [HRSG.]*: *A Little History of the World Wide Web http://www.w3.org/History.html 1995*.**

W3C 1999g

***W3 CONSORTIUM [HRSG.]*: *Extensible Stylesheet Language (XSL) Specification - W3C Working Draft 21 Apr 1999 http://www.w3.org/TR/1999/WD-xsl-19990421/ 1999*.**

W3C 1999h

***W3 CONSORTIUM [HRSG.]*: *XML Linking Language (XLink) World Wide Web Consortium Working Draft 26 July 1999 http://www.w3.org/1999/07/WD-xlink-19990726 1999*.**

W3C 1999i

***W3 CONSORTIUM [HRSG.]*: *XML Pointer Language (XPointer) W3C Working Draft 9 July 1999 http://www.w3.org/1999/07/WD-xptr-19990709 (Stand: 5.8.1999) 1999*.**

9.8 KML4KHS: Schnittstellen-Dokumentation

Die "XML4KHS Schnittstellen -Dokumentation" und der zur Diplomarbeit gehörige Quell -Code können beim Autor per E -mail unter uwe.speck@arcormail.de angefordert werden.