Universität Stuttgart
Fakultät Informatik

| | |
|---|---|
| Prüfer: | Prof. Dr. rer. nat. Jochen Ludewig |
| Betreuer: | Dipl.-Inform. Stefan Krauß |
| Beginn am: | 30.05.2000 |
| Beendet am: | 30.11.2000 |
| CR-Nummer: | D.2.7, H.4.3, H.5.2, I.7.2 |

Studienarbeit-Nr. 1787

# Ein Browser für Software-Dokumentation

Steffen Maier

# Zusammenfassung

SESAM-2 ist ein Forschungsprojekt, das in der Abteilung Software Engineering durchgeführt wird. Es handelt sich hierbei um einen Simulator für Software-Entwicklungsprojekte. Zu SESAM-2 entsteht eine Vielzahl verschiedener Dokumente, z.B. Spezifikationen, Entwürfe, ein Begriffslexikon sowie weitere Entwickler-Dokumente. Es wird ein umfassendes, flexibles Dokumentationssystem benötigt.

Mit der Einführung eines solchen Systems beschäftigt sich das Teilprojekt SESAMDoc. Als grundlegendes Dokumentformat wird hierzu die Auszeichnungssprache DocBook in ihrer XML-Ausprägung eingesetzt. Im Rahmen dieser Studienarbeit wurde nun ein Online-Hilfe-Browser für SESAMDoc konzipiert und realisiert.

Nach Darstellung der Motivation und Beschreibung der Aufgabenstellung folgt ein kurzer Überblick über die wichtigsten Konzepte, auf denen diese Studienarbeit aufbaut. Anschließend werden die Ergebnisse eines Vergleichs bestehender Hilfesysteme erläutert, auf deren Basis dann die Anforderungen an den Online-Hilfe-Browser formuliert werden. Nach einer kurzen Beschreibung der Spezifikation erfolgt ein Überblick über das System aus der Sicht des Benutzers. Nachfolgend wird der modulare und weitgehend komponentenbasierte Entwurf vorgestellt. Die Implementierung des SESAMDoc-Browser erfolgte in der objektorientierten Programmiersprache Java-2 unter Verwendung freier Klassenbibliotheken, die benötigte Standards aus dem XML-Umfeld implementieren. Den Abschluß bilden Informationen zum Verlauf des Projekts mit einer persönlichen Beurteilung und einem Ausblick.

# Inhaltsverzeichnis

**1 Einführung ... 1**
1.1 Motivation ... 1
1.2 Aufgabenstellung ... 2

**2 Grundlagen ... 3**
2.1 Online-Hilfesystem ... 3
2.2 Hypermedia, Hypertext ... 3
2.3 XML (Extensible Markup Language) ... 3
2.4 DocBook ... 5

**3 Vergleich bestehender Online-Hilfesysteme ... 7**
3.1 Auswahl der zu untersuchenden Software ... 7
3.2 Gemeinsamkeiten der untersuchten Produkte ... 8
3.3 Besonderheiten der einzelnen Produkte ... 9
3.3.1 DynaText ... 9
3.3.2 doc.sun AnswerBook2 ... 12
3.3.3 CDE Help Viewer ... 15
3.3.4 FrameMaker ... 16
3.3.5 Panorama Viewer ... 18
3.3.6 JavaHelp ... 19
3.3.7 HTML Help ... 21

**4 Anforderungen ... 23**
4.1 Allgemeine Anforderungen ... 23
4.2 Ansichten der Hilfe-Dokumente ... 24
4.2.1 Fließtext ... 25
4.2.2 Inhaltsverzeichnis ... 25
4.2.3 Index ... 26
4.2.4 Suche ... 26
4.3 Zusätzliche Ansichten ... 27
4.3.1 Lesezeichen ... 27
4.3.2 Betrachtungs-Geschichte ... 28
4.4 Weitere Funktionalität ... 28
4.4.1 Automatische Zusammenführung mehrerer Hilfe-Dokumente und -Pakete ... 28
4.4.2 Programmierschnittstelle ... 29
4.4.3 Internationalisierung ... 29

**5 SESAMDoc-Browser ... 31**
5.1 Unterschiede zu Anforderungen ... 31
5.2 Ausgewählte Punkte der Spezifikation ... 32
5.2.1 Standardsprache der graphischen Oberfläche ... 32
5.2.2 Auslieferungsform des Browsers ... 33
5.2.3 Benutzerspezifische Metadaten des Systems ... 33
5.2.4 Strukturierte Suche ... 33
5.2.5 Fließtext-Ansicht ... 35
5.2.6 Hyperlinks in DocBook ... 36
5.2.7 Nichtfunktionale Anforderungen ... 36
5.3 Produkt aus Sicht des Benutzers ... 37

**6 Systemaufbau ... 41**
6.1 Übersicht ... 41
6.2 Systemkern ... 42
6.3 Datenmodelle ... 44
6.4 Ein-/Ausgabe-Module ... 45
6.4.1 Automatische Erzeugung des Index ... 45
6.5 Eingesetzte Entwurfsmuster ... 49
6.5.1 Java-Interface (Schnittstelle) ... 49
6.5.2 Observer-Muster und Kommunikation mit Ereignissen ... 50
6.5.3 JavaBeans ... 50
6.6 Einteilung und Organisation der graphischen Oberfläche ... 52
6.6.1 Die gebundene Eigenschaften „markedNode“ und „markedEntry“ ... 54
6.6.2 Das Ereignis „LayerEvent“ ... 55
6.7 Modul für die Fließtextansicht ... 55
6.7.1 Anforderungen an den Entwurf des Moduls ... 55
6.7.2 Eigenschaften der Swing Text-Komponenten ... 56
6.7.3 Lösungsansätze für ein Adapter-Modul zwischen DOM-Modell und Text-Komponenten-Ansicht ... 59
6.7.4 Umsetzung der Fließtext-Ansicht ... 61

**7 Realisierung ... 63**
7.1 Verwendete Werkzeuge ... 63
7.1.1 Programmiersprache Java-2 ... 63
7.1.2 Integrierte Entwicklungsumgebung Sun Forté ... 64
7.2 Geplante, noch nicht realisierte Funktionen ... 65
7.3 Nicht geplante, bereits realisierte Funktion ... 67

**8 Projektinformationen ... 69**
8.1 Verlauf ... 69
8.2 Bewertung ... 71
8.3 Mögliche Erweiterungen ... 72
8.4 Ausblick ... 76

**Anhang A Schnittstellen der JavaBeans ... 77**
A.1 Schnittstelle der Dokument-Ansicht-JavaBeans ... 77
A.2 Schnittstelle der Metadaten-JavaBeans ... 78

**9 Literatur ... 81**

**10 Glossar ... 83**

**11 Index ... 89**

# Abbildungsverzeichnis


2–1 Einfaches XML-Dokument aus Beispiel 2–1 in Baumform ..... 5
3–1 Strukturierte Suche mit vordefinierten Anfragetypen in DynaText ..... 10
3–2 AnswerBook2-Benutzerhandbuch, angezeigt im Netscape-Browser ..... 12
3–3 Umleitungen und deren Aufhebung bei AnswerBook2-Servern ..... 14
3–4 Hilfe zur Hilfe im CDE Help Viewer ..... 16
3–5 Online-Hilfe im FrameMaker ..... 17
3–6 Panorama Viewer nach erfolgreicher Suche in einem SGML-Dokument ..... 19
3–7 JavaHelp nach erfolgreicher Volltextsuche ..... 20
3–8 Suchfunktion in HTML Help ..... 22
5–1 Dialog zum Suchanfragetyp „Struktur-Element“ ..... 33
5–2 Dialog zum Suchanfragetyp „Volltext“ ..... 34
5–3 Leeres Hauptfenster des SESAMDoc-Browser ..... 37
5–5 Inhaltsverzeichnis mit Kontextmenü im SESAMDoc-Browser ..... 37
5–6 Lesezeichen mit Kontextmenü im SESAMDoc-Browser ..... 38
5–7 Index mit Suchfunktion im SESAMDoc-Browser ..... 38
5–4 Hauptfenster des SESAMDoc-Browser mit geöffnetem Hilfe-Dokument ..... 38
5–8 Betrachtungs-Geschichte im SESAMDoc-Browser ..... 39
6–1 Systementwurf ..... 41
6–2 Systemkern ..... 43
6–3 Datenmodelle ..... 44
6–4 Aufgebaute Datenstruktur nach der Indexgenerierung ..... 48
6–5 Observer-Entwurfsmuster in UML-Notation ..... 51
6–6 Schematisches, statisches Klassendiagramm der graphischen Oberfläche ..... 53
6–7 Die gebundenen Eigenschaften „markedNode“ und „markedEntry“ ..... 54
6–8 Swing Textmodell und DOM in der Gegenüberstellung ..... 57
6–9 Datenmodell der Swing-Text-Komponente JTextPane ..... 58
6–10 Eingehüllte DOM-Struktur gibt sich als Swing-Textmodell aus. ..... 60
7–1 XPath zur persistenten Adressierung beliebiger DOM-Elemente. ..... 67
8–1 Mehrfache Sprungziele eines Indexeintrags ..... 73


# Tabellenverzeichnis


8–1 Arbeitspakete und entstandene Ergebnisse ..... 69
8–2 Kalenderwochen-Übersicht ..... 70

# 1 Einführung

## 1.1 Motivation

In der Abteilung Software Engineering wird das Forschungsprojekt SESAM-2 durchgeführt. Es handelt sich hierbei um einen Simulator für Software-Entwicklungsprojekte. Der Simulator dient der Ausbildung von Projektleitern. Hierzu „spielt“ der Projektleiter ein Software-Projekt mit Hilfe des Simulator durch und lernt aus den Reaktionen des Simulator. SESAM-2 enthält Modelle zur Simulation solcher Software-Projekte und kann dadurch auf die Entscheidungen des Projektleiters reagieren, um so möglichst realitätsnah den Fortgang des simulierten Projekts nachzubilden [Drappa et al. 1995].

SESAM-2 enthält eine Vielzahl verschiedener Dokumente. Während der Entwicklung der Software selbst entstehen Spezifikationen, Entwürfe und Quelltext-Dokumentation sowie verschiedene Entwickler-Dokumente. Weiterhin ist ein umfangreiches Begriffslexikon vorhanden. Zur Unterstützung des Spielers wird außerdem eine Online-Hilfe benötigt. Diese ermöglicht es, dem Spieler kontextsensitive, d.h. vom Zustand der Simulations-Benutzungsoberfläche abhängige, Hilfestellung zu geben.

Die Dokumentation wird bislang in unterschiedlichen Textverarbeitungssystemen erstellt und bearbeitet. Das Begriffslexikon ist beispielsweise ein LaTeX-Dokument. Andere Dokumentation stammt aus der Textverarbeitung FrameMaker der Firma Adobe. All diese Dokumente haben gemeinsam, daß sie unterschiedliche, teilweise proprietäre Formate verwenden. Somit kann ein Dokument nur mit dem jeweiligen Programm, in dem es erstellt wurde, angezeigt und weiterverarbeitet werden.

Zur zwischenzeitlichen Lösung dieses Problems wurde die Seitenbeschreibungssprache Postscript als zusätzliches, allen verschiedenen Dokumentationsformaten gemeinsames, Format verwendet. Dies ermöglicht den Austausch der Dokumente und deren Anzeige unabhängig vom Vorhandensein der ursprünglichen Anwendung, in der sie erstellt wurden. Außerdem kann ein Dokument jederzeit problemlos im Postscript-Format auf einem Drucker zu Papier gebracht werden, da viele Drucker diese Seitenbeschreibungssprache verstehen. Selbst die Dokument-Ausgabe auf einem Drucker ohne Postscript-Unterstützung oder am Bildschirm ist möglich. Mit Hilfe von Software-basierten Postscript-Interpretern wie beispielsweise Ghostscript kann das Dokument in die Sprache, die der Drucker versteht, transformiert und anschließend ausgedruckt bzw. angezeigt werden. In eingeschränktem Maße kann auf Postscript-Dokumenten auch eine Volltextsuche durchgeführt werden, um bestimmte Zeichenketten innerhalb eines Dokuments aufzufinden.

Die strukturierte Suche nach Information in Dokumenten im Postscript-Format ist hingegen nicht möglich. Es kann beispielsweise keine Suche nach Stichwörtern in den Überschriften eines Dokuments durchgeführt werden. Auch die Suche nach Simulations-Kommandos in der Benutzungsanleitung des SESAM-Systems ist in Postscript nicht möglich.

Weiterhin kann mit Hilfe von Postscript-Dokumenten keine kontextsensitive Hilfe für den SESAM-Spieler realisiert werden. Die kleinste adressierbare Einheit für die Anzeige solcher Dokumente ist eine Seite. Dies ist eine zu grobe Aufteilung, da auf einer Seite unter Umständen mehr Information enthalten ist, als zur Hilfe angezeigt werden soll. Um dieses Manko zu beheben, müßte ein solches Dokument so erzeugt werden, daß auf einer Seite nur die für den Zusammenhang notwendige Information enthalten ist und der Rest evtl. unbeschrieben bleibt. Allerdings ist ein derart speziell formatiertes Dokument wiederum schlechter für den Ausdruck geeignet, da der Papierverbrauch unnötig ansteigt.

Eine Darstellung in Hypermedia- bzw. Hypertext-Systemen ist mit Postscript-Dokumenten ebenfalls nicht möglich, da das Format keine Assoziationen zwischen verschiedenen Stellen im Dokument bzw. zwischen Dokumenten kennt.

Aus den eben beschriebenen Einschränkungen von Postscript als gemeinsames Format für Dokumentation im SESAM-2 Projekt heraus ist es also notwendig, ein allgemeines, übergreifendes Dokumentationsformat einzuführen. Mit der Einführung eines umfassenden, flexiblen Dokumentationssystems für das Projekt beschäftigt sich das Teilprojekt SESAMDoc. Neben einem Autorenwerkzeug zur Erstellung von Dokumentation werden in SESAMDoc Filter zur Konvertierung zwischen verschiedenen Dokumentformaten entwickelt. Zur Betrachtung der Dokumentation am Bildschirm wird weiterhin ein Online-Hilfesystem benötigt.

Im Rahmen dieser Studienarbeit war es meine Aufgabe einen Browser für Software-Dokumentation in XML zu konzipieren und zu entwickeln. Das grundlegende Dokumentformat in SESAMDoc ist die standardisierte Auszeichnungssprache DocBook, wobei die Dokumente im XML-Format abgespeichert werden. Zur Erklärung dieser Begriffe sei an dieser Stelle auf die Abschnitte 2.3 „XML (Extensible Markup Language)“ auf Seite 3 und 2.4 „DocBook“ auf Seite 5 verwiesen.

## 1.2 Aufgabenstellung

Ziel dieser Studienarbeit ist die Konzeption und Realisierung eines Online-Browsers für Software-Dokumentation im XML-Format. Der Browser soll Hilfe-Dokumente am Bildschirm entsprechend aufbereiten (d.h. Überschriften in größerer Schrift und mit Abstand anzeigen usw.). Er soll Lesezeichen unterstützen, die Volltextsuche in einem und über mehrere Dokumente hinweg zulassen sowie einen Gesamtindex verwalten und mit Dokumenten in verschiedenen Sprachen und Versionen umgehen können. Weitere Anforderungen an ein solches System sind als Teil der Studienarbeit zu ermitteln. Dazu sollen im Wesentlichen die Fähigkeiten von bestehenden Online-Hilfesystemen untersucht werden.

Im einzelnen umfaßt diese Studienarbeit die folgenden Teilaufgaben:

1. Erheben der Anforderungen an einen Online-Dokumenten- und Hilfe-Browser.
2. Vergleich der Möglichkeiten verbreiteter Online-Hilfesysteme (Windows, CDE, Solaris, FrameMaker).
3. Erstellen einer Spezifikation für einen Online-Browser für die SESAM-Dokumentation.
4. Entwurf des Werkzeugs.
5. Implementierung und Test.

Eine wichtige Bedingung der Aufgabenstellung ist die Vorgabe, daß das entstehende Produkt unter der GNU General Public License (GNU GPL) veröffentlicht werden soll. Dies stellt unter anderem sicher, daß eine Weiterentwicklung durch Dritte nicht ausgeschlossen werden kann. D.h. der Browser kann unter anderem innerhalb der Abteilung Software Engineering jederzeit unter Zuhilfenahme des Quelltextes an die gegebenen Bedürfnisse angepaßt werden.

# 2 Grundlagen

In diesem Kapitel werden die wichtigsten Konzepte, auf deren Grundlage diese Studienarbeit aufbaut, in kurzen Überblicken erläutert.

## 2.1 Online-Hilfesystem

Zur Betrachtung der SESAM-2 Dokumentation am Bildschirm wird ein Online-Hilfesystem benötigt. Es dient nicht nur dazu, Entwickler-Dokumente zum Durchlesen anzuzeigen, sondern auch um dem SESAM-Spieler kontextsensitive Hilfe anzubieten. Derartige Hilfesysteme sind heutzutage Teil von vielen Software-Systemen und dienen der Unterstützung des Benutzers bei der Bedienung des Systems. Um den Begriff Online-Hilfesystem genauer abzugrenzen, möchte ich eine Definition aus [Herczeg 1994] heranziehen:

„Ein Hilfesystem ist eine Systemkomponente zur Analyse des aktuellen Systemkontextes mit folgenden Eigenschaften:

1. Es erklärt den Systemzustand, d.h. die vorhandenen Objekte (der Anwendung und des Dialogs), ihre Eigenschaften sowie ihre Relationen zu anderen Objekten.
2. Es erklärt die Systemfunktionalität, d.h. die verfügbaren sowie die aktuell möglichen Funktionen.
3. Die Erklärungen finden in einer zur vorhandenen Dialogsprache zusätzlichen Repräsentation statt.

Hilfe ist in diesem Sinn immer eine kontextabhängige Unterstützung des Benutzers.“

## 2.2 Hypermedia, Hypertext

Hypermedia wurde in den 60er Jahren als Organisationsform für Information geschaffen. Die Information wird dabei in einzelne, abgeschlossene Teilbereiche, sogenannte Knoten, aufgeteilt. Diese Knoten sind mittels Assoziationen miteinander verknüpft und ergeben auf diese Weise einen Informationsraum. Hypermedia kann Information verschiedener Formate einbinden, sowohl Text als auch Bilder, Graphiken, Video oder Töne.

Hypertext ist eine spezielle Form von Hypermedia, die sich hauptsächlich auf textbasierte Information stützt. Durch die Verbreitung des World Wide Web (WWW) hat diese Ausprägung von Hypertext einen hohen Bekanntheitsgrad erreicht. Das WWW besteht aus einer Vielzahl von Hypertext-Seiten, die mit Hilfe von Hyperlinks logisch miteinander verbunden werden. Betrachtet werden die Seiten mit einem Webbrowser, der die Seiten auf einem graphischen Bildschirm darstellt. Meist werden Hyperlinks in Form von blauem, unterstrichenem Text optisch hervorgehoben. Üblicherweise wird die Verfolgung eines Hyperlinks durch einen Mausklick aktiviert. Durch Verfolgung der Hyperlinks können Seiten mit verwandtem oder weiterführendem Inhalt angezeigt werden. Außerdem ist die Einbindung von Bildern und Graphiken zur Veranschaulichung der dargestellten Information möglich.

Der in dieser Studienarbeit zu erstellende Browser soll Hilfe-Dokumente als Hypertext am Bildschirm darstellen. Insofern stellt er ähnlich den bekannten Webbrowsern Hilfe-Information als Text mit optionalen Abbildungen dar und bietet Hyperlinks an.

## 2.3 XML (Extensible Markup Language)

Mit der starken Verbreitung und Nutzung des Internet werden weltweit immer mehr Daten auf elektronischem Wege ausgetauscht. Voraussetzung für eine fehlerlose Kommunikation zwi-

schen zwei Partnern ist die Verständigung auf ein gemeinsames Datenformat, das beide Partner verstehen. Die derzeit im WWW eingesetzte Seitenbeschreibungssprache HTML (Hypertext Markup Language) ist durch eine Vielzahl proprietärer Erweiterungen an einem Punkt angelangt, an dem der HTML-Standard nicht mehr sinnvoll erweitert werden kann. Aus diesem Grunde versuchten SGML-Experten die Sprache SGML (Standard Generalized Markup Language) ins WWW zu bringen. Da diese generische Auszeichnungssprache in ihrer vollen Ausprägung äußerst komplex ist, mußte eine einfachere Form davon entwickelt werden, die problemlos im Internet einsetzbar ist. Das Ergebnis dieser Arbeit ist XML.

Die Extensible Markup Language (XML) ist eine Metasprache zur Definition von Auszeichnungssprachen. Die definierbaren Auszeichnungssprachen haben grundsätzlich ein textbasiertes Format und dienen dazu, Texten Semantik und logische Struktur zu verleihen. Dies bedeutet, im Text sollen Informationen enthalten sein, die Aussagen über bestimmte Teile eines Dokuments beinhalten. Das Layout, also die visuelle Erscheinung eines Dokuments, soll von der Dokumentstruktur völlig getrennt werden. Es macht viel mehr Sinn, ein Zitat als eben solches zu beschreiben, als beispielsweise eine bestimmte Schriftart und den Schriftschnitt anzugeben. Eine solche Beschreibung eines Textabschnitts nennt man generische Auszeichnung. Die Struktur eines XML-Dokuments läßt sich mit Hilfe einer DTD (Dokumenttyp-Definition) exakt und überprüfbar festlegen.

Die XML-Spezifikation des W3C [Bray et al. 1998] enthält eine recht formale Beschreibung: „Die Extensible Markup Language (XML) beschreibt eine Klasse von Datenobjekten, die XML-Dokumente genannt werden. Außerdem beschreibt XML zum Teil das Verhalten von Programmen, die solche Dokumente verarbeiten. XML ist ein SGML-Anwendungsprofil und somit eine echte Teilmenge von SGML.“

Bedingt durch die gemeinsame Obermenge SGML sehen XML- und HTML-Dokumente auf den ersten Blick recht ähnlich aus. XML ist aber keine Erweiterung von HTML, sondern eine eigene Metasprache, mit der sich jeder selbst seine eigene Auszeichnungssprache definieren kann. Ein einfaches XML-Dokument kann wie in Beispiel 2–1 aussehen.

**Beispiel 2–1: Einfaches XML-Dokument**

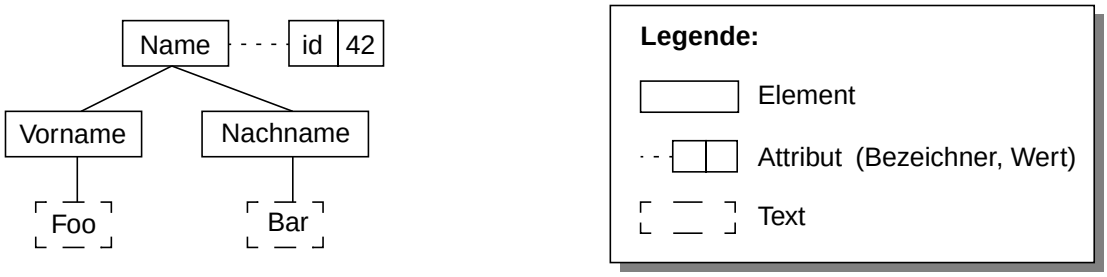
```
<?xml version="1.0"?>
<Name id="42">
  <Vorname>Foo</Vorname>
  <Nachname>Bar</Nachname>
</Name>
```

In spitzen Klammern werden sogenannte Elemente notiert. Elemente sind die Bausteine zur strukturierten, hierarchisch organisierten Auszeichnung von Inhalt. Im Beispiel wird ein Name bestehend aus dem Vornamen „Foo“ und Nachnamen „Bar“ beschrieben. Zur Parametrisierung von Elementen bietet XML Attribute. Sie bestehen aus einem Paar von Bezeichner und zugehörigem Wert. Im Beispiel verleiht das Attribut „id“ dem Namen eine eindeutige Nummer „42“, über die der Name beispielsweise adressiert werden kann. Mit Hilfe dieser textbasierten Mechanismen läßt sich Information in eine Struktur verpacken, die einfach durch Maschinen verarbeitet werden kann. Durch das Textformat läßt sich die Information darüberhinaus jederzeit vom Menschen lesen und bearbeiten.

Alle XML-Dokumente haben die Eigenschaft, daß ihre Struktur als Baum dargestellt werden kann. In Abbildung 2–1 ist das Dokument aus Beispiel 2–1 in Form eines Baumes dargestellt. In leicht veränderter Form repräsentiert das DOM (Document Object Model) XML-Dokumente in dieser Baumform. DOM spezifiziert die Programmierschnittstelle zum Zugriff auf ein

geparstes XML-Dokument, das sich im Speicher befindet. Der Zugriff erfolgt analog zu anderen vergleichbaren, baumartigen Datenstrukturen [Apparao et al. 1998].

**Abbildung 2–1: Einfaches XML-Dokument aus Beispiel 2–1 in Baumform**




## 2.4 DocBook

DocBook ist eine Auszeichnungssprache, um strukturierte Dokumente mit Hilfe von XML oder SGML schreiben zu können. Dies bedeutet, DocBook existiert in Form einer DTD (Dokumenttyp-Definition) sowohl für XML als auch für SGML und beschreibt somit die Struktur eines DocBook-Dokuments [Walsh et al. 1999].

Die Struktur selbst orientiert sich relativ nah am Aufbau eines Buches. Hierzu bietet DocBook Elemente, welche die Gliederung eines Buches beschreiben. Es existieren beispielsweise Artikel, Bücher, Kapitel, Abschnitte und Unterabschnitte sowie Elemente zur Auszeichnung von Überschriften und Absätzen. Im Besonderen ist DocBook geeignet für technische Dokumentationen, beispielsweise zu Computer-Hardware und Software, ist aber keinesfalls auf diesen Anwendungsbereich beschränkt. Die Erstellung technischer Dokumentation wird unterstützt durch Elemente zur Beschreibung von Produkten, Komponenten graphischer Benutzungsoberflächen, Befehlen und Funktionen bzw. Klassen mit Methoden und Variablen, Programm- und Fehlermeldungen sowie mathematischen Formeln. Da sich DocBook wie bereits beschrieben am Aufbau eines Buches orientiert, lassen sich unter Vernachlässigung der speziellen Auszeichnungen für technische Dokumentation durchaus auch Bücher oder Artikel mit prosaischem Inhalt in DocBook erstellen.

Bedingt durch die generische Auszeichnung, welche die Basis für DocBook bietet, ist die Ausgabe ein und desselben DocBook-Dokuments auf einer Vielzahl verschiedener Ausgabemedien möglich, z.B. hochwertige, gedruckte Bücher, Online-Präsentation in Hypermediasystemen, aber auch Ausgabe auf Braillezeile oder Sprachsynthese.

In dieser Studienarbeit wird die XML-DTD von DocBook eingesetzt. Der Grund dafür ist die leichtere Verarbeitung von XML gegenüber SGML. Dadurch ist die Verfügbarkeit freier XML-verarbeitender Software wie beispielsweise Parser ungleich höher. Zusätzlich zeichnet sich in den Mailinglisten zu DocBook ein allgemeiner Trend für die Zukunft in Richtung XML-DocBook ab, der sicherlich auch auf den eben genannten Grund zurückzuführen ist.

**Beispiel 2–2: Einfaches DocBook-XML-Dokument**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1//EN"
  "http://www.oasis-open.org/docbook/xml/4.1/docbookx.dtd">
<book lang="de">
  <chapter>
    <title>Ein einfaches DocBook-Buch</title>
    <section>
```

```
        <title>Ein Abschnitt</title>
        <para>Ein Absatz in diesem Abschnitt.</para>
        <para>Ein weiterer Absatz.</para>
        <section>
          <title>Ein Unterabschnitt</title>
          <para>Das <xref linkend="example.xml"/> zeigt
            ein einfaches XML-Dokument.</para>
          <example id="example.xml">
            <title>Einfaches XML-Dokument</title>
            <programlisting format="linespecific"><![CDATA[
<?xml version="1.0"?>
<Name id="42">
  <Vorname>Foo</Vorname>
  <Nachname>Bar</Nachname>
</Name>
            ]]></programlisting>
          </example>
        </section>
      </section>
    </chapter>
  </book>
```

Beispiel 2–2 zeigt ein einfaches DocBook-XML-Dokument.

Es beginnt mit dem üblichen XML-Prolog, der die verwendete XML-Version und den Zeichensatz spezifiziert. Anschließend wird die DTD festgelegt, über die jederzeit die Struktur des Dokuments auf Gültigkeit überprüft werden kann.

Ab Zeile 4 beginnt das eigentliche DocBook-Buch (book). Das Attribut „lang“ definiert die Sprache, in welcher der Inhalt des Buches verfaßt ist, hier im Beispiel deutsch. Innerhalb des Buches befindet sich ein Kapitel (chapter) mit seiner Überschrift (title) und diversen enthaltenen Abschnitten (section).

Als Beispiel für eine technische Auszeichnung ist das einfache XML-Dokument aus Beispiel 2–1 (example) in einem Unterabschnitt enthalten. Damit die Elemente des eingebetteten XML-Quelltextes (programlisting) nicht als DocBook-Auszeichnung mißverstanden werden, wird der Bereich mit Hilfe eines CDATA-Konstrukts geschützt. Dadurch wird der Inhalt des Konstrukts Zeichen für Zeichen als reiner Text interpretiert.

Im Absatz (para), der im DocBook-Buch dem Beispiel vorangeht, wird mit Hilfe eines Querverweises (xref) auf das nachfolgende Beispiel verwiesen. Das Ziel wird anhand des ID-Attributs spezifiziert. Ein solcher Querverweis kann bei der Darstellung in Hypertextsystemen als Hyperlink umgesetzt werden, so daß bei Aktivierung des Links das referenzierte Beispiel angezeigt wird. Bei einer Konvertierung dieses Absatzes nach HTML könnte das Ergebnis wie folgt aussehen: „Das Beispiel 1 zeigt ein einfaches XML-Dokument.“

# 3 Vergleich bestehender Online-Hilfesysteme

Um die Anforderungen an den zu entwerfenden Online-Hilfe-Browser zu ermitteln, habe ich das Projekt direkt nach der Aufstellung des Projektplans mit einem Vergleich bestehender Online-Hilfesysteme begonnen. Ein Vergleich ist insofern sinnvoll, als der Browser zunächst an der Funktionalität bestehender Systeme orientiert sein soll.

## 3.1 Auswahl der zu untersuchenden Software

Die Auswahl der untersuchten Software erfolgte nach verschiedenen Gesichtspunkten. Zu Beginn stand eine Internet-Recherche zum Thema Software-Dokumentation. Allerdings ergaben sich dabei wenig verwertbare Fakten. Meist wurde auf den gefundenen Web-Seiten das Stichwort zwar erwähnt, aber keine weiteren grundsätzlichen Aussagen zu Software-Dokumentation getroffen.

Die einschlägig bekannten Web-Seiten, beispielsweise zu professioneller technischer Dokumentation mit Hilfe von SGML, führten dagegen zu einer Vielzahl von Einsatzgebieten und Software-Produkten. Robin Covers SGML-Seite[1] zeigt eine nahezu unüberschaubare Aufzählung von Einsatzgebieten für SGML mit geeigneter Software. Erwähnung finden dort die Produkte 1. bis 3.

1. **Sun Microsystems, doc.sun AnswerBook2.** Als Standard-Dokumentationssystem für Sun-Produkte existiert AnswerBook2 durch die Ausstattung mit Sun-Rechnern in der Abteilung Software Engineering und bietet sich zur Untersuchung an. Siehe hierzu auch `http://www.oasis-open.org/cover/gen-apps.html#sunAnswerbook`
2. **CDE (Common Desktop Environment), Help Viewer.** Ebenfalls in der Abteilung vorhanden ist das CDE als Standard-Desktop-Oberfläche auf Sun-Rechnern. Siehe hierzu auch `http://www.oasis-open.org/cover/gen-apps.html#cde`
3. **SoftQuad, Panorama Viewer.** Dieses Produkt wird auf diversen Web-Seiten, die sich mit SGML-basierten elektronischen Bibliotheken befassen, als ein möglicher Browser erwähnt. Siehe hierzu auch `http://www.oasis-open.org/cover/ceth-panoramaTEI.html`
4. **Electronic Book Technologies, DynaText.** Als Browser für die Dokumentation von Novells Netware 4.1 war mir DynaText noch bekannt. Außerdem befindet sich DynaText als Browser für die Dokumentation des CASE-Tools Objectteam in der Abteilung Software Engineering im Einsatz.
5. **Adobe, FrameMaker.** Wird in der Abteilung Software Engineering eingesetzt, um verschiedenste Dokumente zu erstellen.
6. **JavaSoft, JavaHelp.** Im Zusammenhang mit dem SESAMDoc-Projekt befindet sich JavaHelp auf der, extra für dieses Projekt zusammengestellten, Software-Liste und wurde ebenfalls in den Vergleich mit aufgenommen.
7. **Microsoft, HTML Help.** Der große Marktanteil von Microsoft Windows bei den PC-Betriebssystemen und die dazugehörigen Entwicklerinformationen aus dem sogenannten MSDN (Microsoft Developers Network), die im HTML Help Format vorliegen, sind die Gründe für die Betrachtung des HTML Help Produktes.

Wie sich beispielsweise an der Aufnahme des schon etwas älteren Panorama Viewer in den Produktvergleich erkennen läßt, war das Ziel der Auswahl keinesfalls eine möglichst aktuelle Marktübersicht. Vielmehr wurde Wert gelegt auf eine breite Auswahl an Software, die gewinnbringend im Bereich Software-Dokumentation praktisch eingesetzt werden kann.

---

1. `http://www.oasis-open.org/cover`

## 3.2 Gemeinsamkeiten der untersuchten Produkte

Die im folgenden aufgezählten Eigenschaften sind allen untersuchten Produkten gemeinsam. Im Anschluß an die Aufzählung werden die einzelnen Eigenschaften jeweils näher erläutert.

- Auszeichnungssprache als Dokumentformat.
- Hypertextsystem mit graphischer Benutzungsoberfläche.
- Ansichten: Dokumentinhalt als Fließtext, Inhaltsverzeichnis.
- Volltextsuche.

**Dokumentformat**

Als Basis für die Erstellung der darzustellenden Hilfe-Dokumente werden verschiedene Formen von Auszeichnungsformaten verwendet. Bei den professionelleren Produkten ist dies SGML, bei den weniger aufwendigen oftmals HTML oder ein proprietäres Format. FrameMaker bietet mit MIF (Maker Interchange Format) alternativ ein dokumentiertes, textbasiertes Format zum Im- und Export an.

**Graphische Benutzungsoberfläche**

Sämtliche betrachtete Software bietet als Schnittstelle zwischen Mensch und Maschine eine zeitgemäße graphische Benutzungsoberfläche. Mit Hilfe dieser graphischen Oberfläche realisieren alle Produkte mehr oder weniger aufwendige Hypertext-Systeme. Dazu gehört die Aufteilung der Hilfe-Dokumentation in Informationsknoten, den strukturgebenden Elementen eines Hypertext-Systems. Neben dem streng sequentiellen Lesen eines Dokuments wird dem Leser mittels Hyperlinks eine intuitive Navigationsvariante ermöglicht. Hyperlinks sind Assoziationen zwischen verschiedenen Informationsknoten. Sie ermöglichen jederzeit zu thematisch verwandten Stellen im Dokument zu springen. „Ihr Nutzen und ihre Wirksamkeit gründet sich auf der psychologischen und vielleicht physiologischen Bedeutung von Assoziationen im menschlichen Langzeitgedächtnis“ [Herczeg 1994]. Die Navigationsvariante, die durch Verfolgen von Verweisen entsteht, wird Browsing genannt und steht für das ziellose Herumstöbern im Informationsraum eines Hypertext-Systems. Darauf begründet sich auch der Begriff des Browsers, der Anwendung, mit der ein Benutzer diese Navigationsart durchführen kann.

**Dokumentansichten**

Durchweg enthalten die verglichenen Online-Hilfesysteme mindestens zwei verschiedene Ansichten eines Dokuments.

Zum einen wird der Hilfe-Dokument-Inhalt als formatierter Fließtext dargestellt. Darin enthalten sind die bereits erwähnten Hyperlinks, wodurch nach [Herczeg 1994] die Such- und Zugriffstrategien Browsing, Exploration sowie Navigation ermöglicht werden. Browsing, das ziellose Herumstöbern wurde bereits oben beschrieben. Die Exploration ist ein wichtiges Suchverfahren in neuen, unbekannten Informationsräumen, bei dem der Benutzer durch systematisches Verfolgen von Verweispfaden die Struktur ergründet. Die Navigation ist die zielorientierte Suche durch Verfolgen von Verweisen, wobei die Zielsuche durch globale Übersichtsinformationen unterstützt wird.

Die zweite Art der Ansicht ist eine Dokumentgliederung, die fast immer als Baum realisiert ist. Die einzelnen Äste dieses Baumes lassen sich unabhängig voneinander ein- bzw. ausklappen, um dem Benutzer eine übersichtliche Darstellung der Dokumentgliederung zu ermöglichen. Nach [Herczeg 1994] ist über eine derartige Ansicht die Suchstrategie der indexbasierten Suche möglich. Die Auswahl von Baumknoten in Preorder-Traversierung bzw. in entgegengesetzter Richtung entspricht einer sequentiellen Bewegung vorwärts bzw. rückwärts durch das

Dokument. Mit der Auswahl eines übergeordneten Vaterknotens im Baum, navigiert der Benutzer zielgerichtet zum thematisch übergeordneten Informationsknoten. Die indexbasierte Suche wird durch die hierarchisch geordnete Gliederung im Baum unterstützt.

Bei der Auswahl eines Baumknotens durch den Benutzer ändert sich die Fließtext-Ansicht, indem sie entweder an die zugehörige Textstelle springt oder, falls diese in einem anderen Teil des Dokuments liegt, diesen Teil öffnet und anzeigt. Teilweise ist diese logische Verbindung zwischen Gliederungsknoten und Fließtext-Bereich beidseitig synchron. Dies bedeutet, daß auch bei Bewegung im Fließtext, beispielsweise durch Scrolling, stets der zugehörige Baumknoten markiert wird. Auf diese Weise kann der Benutzer die aktuelle Position im Dokument, bezogen auf die Gliederung, erkennen.

Einzig das untersuchte Produkt AnswerBook2 fällt bezüglich diesem Verhalten etwas aus dem Rahmen. Da als Ausgabeformat HTML in der Version 3.2 erzeugt und in einem beliebigen HTML-Browser angezeigt wird, sind die Ansichten, die derartige Browser erzeugen können ohne Bezug zueinander. Ohne diesen Bezug ist eine Beeinflussung der dargestellten Position in einer anderen Ansicht nicht möglich. Framesets oder aktive HTML-Inhalte über JavaScript oder DynamicHTML, die eine geteilte Ansicht ermöglichen würden, deren Teile Einfluß aufeinander nehmen könnten, kommen aus Kompatibilitätsgründen nicht in Frage. Zum Ausgleich wird bei diesem System jedoch zu jedem Informationsknoten ein durch den Kontext begrenzter Teil der Gliederung zur Orientierung im Informationsraum mitausgegeben.

**Suche**

Mit Ausnahme des CDE Help Viewer ist mit allen Produkten eine Volltextsuche möglich. Da diese oftmals in einem vorgenerierten Volltextindex stattfindet, wird somit nicht unbedingt jedes Vorkommen beliebiger Wörter wiedergefunden. Diesem Punkt steht dafür eine sehr effiziente Durchführung der Suchanfrage gegenüber. Außerdem werden oftmals Wörter einer sogenannten Stopliste nicht in den Volltextindex mitaufgenommen. Bei diesen Wörtern handelt es sich um Artikel, Konjunktionen und andere Wörter, deren Auftreten in Suchergebnissen unerwünscht, da irrelevant, ist. Durch eine Stopliste werden diese bereits bei der Generierung eines Volltextindex ausgespart.

## 3.3 Besonderheiten der einzelnen Produkte

Natürlich sind die oben erwähnten Gemeinsamkeiten der untersuchten Produkte noch nicht ausreichend für ein benutzerfreundliches Online-Hilfesystem. Um einen Eindruck von den Möglichkeiten zu bekommen, die ein solches System darüberhinausgehend anbieten kann, werden in den folgenden Abschnitten ausgewählte Besonderheiten der verglichenen Produkte dargestellt.

### 3.3.1 DynaText

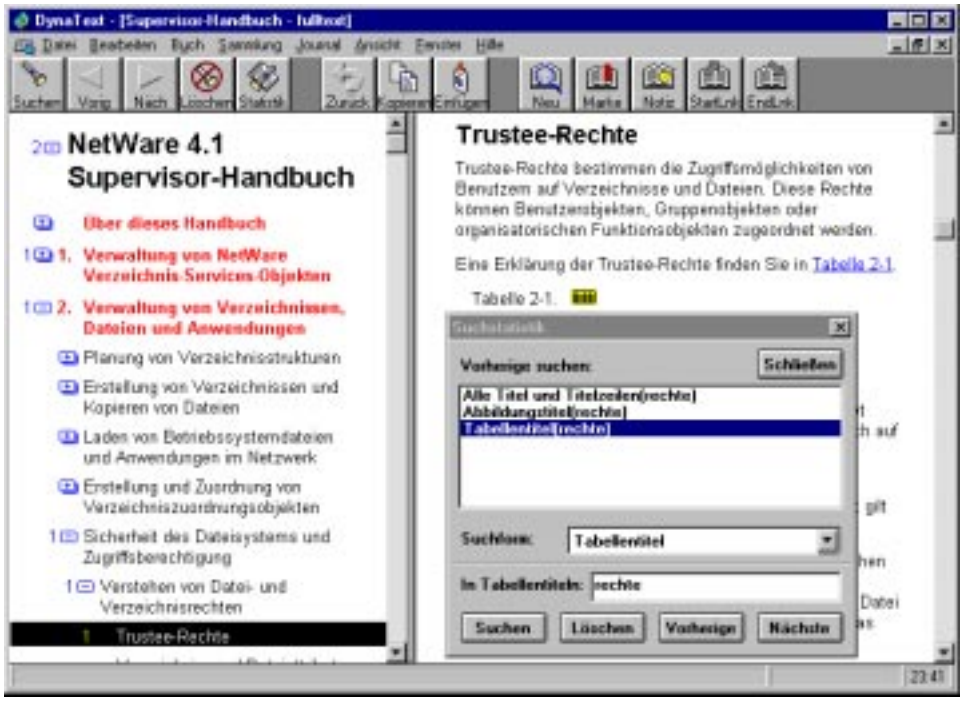
DynaText von Electronic Book Technologies ist ein professioneller Browser für technische Dokumentation. Basierend auf beliebigen SGML-Dokumenttyp-Definition werden SGML-Dokumente in Zusammenspiel mit Stylesheets für die entsprechenden Ansichten wie Inhaltsbaum und formatierter Fließtext zur Darstellung gebracht. Die vom Dokument-Autor erstellten SGML-Dokumente werden in ein vorkompiliertes Binärformat umgewandelt. Auf diesem Format können dann Stylesheet- und Suchmaschinen effizient arbeiten, da die Dokumente beim Zugriff nicht von Grund auf neu geparst werden müssen. Erstellt werden die Dokumente mit dem separat zu erwerbenden DynaText Publisher, in dem sowohl die Eingabe der Dokumente als auch die Erstellung der zugehörigen Stylesheets erfolgt.

Diese Software ist in der untersuchten Form (Version 2.2(b)) schon etwas älter und stammt aus den Mittneunzigern. Die Firma Electronic Book Technologies war bei einer Recherche im Internet nicht mehr aufzufinden. Stattdessen führt die Firma INSO in ihrem aktuellen Angebot die Nachfolgeprodukte DynaText und DynaText/TE. Als Hilfesystem zum CASE-Tool Objectteam befindet sich DynaText in der Abteilung Software Engineering im Einsatz. Allerdings erschien die Implementierung unter Solaris etwas instabil. Zum Test wurde deshalb die Software und die Testdokumente in Form einer DynaText-Buchsammlung von einer Netzwerk Shareware-CD mit einer kostenfreien 2-Benutzer Lizenz für Novell Netware 4.1 bezogen.

**Suche**

Eine herausragende Eigenschaft von DynaText ist die Suche. Vom Dokument-Autor vorgegebene Suchanfragetypen in einem Buch-Dokument abstrahieren von der SGML-Struktur des Hilfe-Dokuments. Der Benutzer muß nur vordefinierte Eingabefelder im Suchdialog ausfüllen und kann somit strukturierte Suchanfragen ausführen, ohne die bisweilen sehr komplizierte Dokument-Struktur kennen zu müssen. Abbildung 3–1 zeigt den DynaText-Browser mit eingeblendetem Suchdialog „Suchstatistik“. In diesem Dialog wurde eine strukturierte Suche in den Titeln aller Tabellen im „Netware 4.1 Supervisor-Handbuch“ durchgeführt. Als Suchergebnis wurden zwei Tabellen gefunden. Im Inhaltsverzeichnis in der linken Fensterhälfte sind die akkumulierten Suchtrefferanzahlen links der Baumknoten zu sehen.

**Abbildung 3–1: Strukturierte Suche mit vordefinierten Anfragetypen in DynaText**




Hinzu kommt die Suchsyntax für die Volltextsuche, die neben den typischen Platzhaltern für ein oder mehrere Buchstaben (? bzw. *) Schlüsselwörter zur Verknüpfung der Suchbegriffe bietet. Neben booleschen Operatoren (und, oder, nicht) sind vor allem Operatoren, die die Nähe von Suchbegriffen festlegen, nützlich, um das gewünschte Suchergebnis zu erhalten. Mit diesen Operatoren kann festgelegt werden, daß mehrere, vom Benutzer spezifizierte Suchbegriffe nur dann zu einem Suchtreffer führen, wenn diese innerhalb einer bestimmten Anzahl von aufeinanderfolgenden Wörtern in der Hilfe enthalten sind. Damit wird ausgeschlossen,

daß unerwünschte Suchtreffer entstehen, bei denen die Suchbegriffe weit verstreut und möglicherweise in völlig unterschiedlichen Zusammenhängen in einem Dokument enthalten sind.

**Hyperlinks**

Hyperlinks werden in DynaText nicht nur direkt vom Dokument-Autor in den Dokumenten vorgegeben, sondern sind zusätzlich auch vom Benutzer selbst definierbar. Außer den, aus dem WWW bekannten einfachen einseitigen Links, werden auch mehrfache und zweiseitige Links geboten.

Zweiseitig bedeutet, daß nach der Verfolgung eines Hyperlinks vom Ziel aus wieder zurück zum Ausgangspunkt gesprungen werden kann. Dies ist ungleich der Funktionalität des Zurückspringens in der Betrachtungs-Geschichte wie sie derzeit gängige Webbrowser zur Verfügung stellen. Zweiseitige Hyperlinks können jederzeit in beide Richtungen verfolgt werden. Es ist nicht notwendig, direkt zuvor das eine Verweisende betrachtet zu haben, um nach dem Wechsel zum anderen Ende wieder zurückspringen zu können.

Mehrfache Links können von ein und demselben Ausgangspunkt zu mehreren verwandten Zielpositionen verweisen. Somit muß nicht zwanghaft für jede Assoziation ein eigener Link dargestellt werden, obwohl sich alle auf dasselbe verwandte Thema beziehen. Meist werden solche mehrfachen Hyperlinks in der graphischen Oberfläche als Auswahl-Box, Kontext-Menü oder Auswahlliste in einem eigens zu diesem Zweck geöffneten Dialogfenster realisiert, aus denen der Benutzer ein gewünschtes Verweisziel auswählen kann.

Benutzerdefinierte Hyperlinks sind zwar logisch mit dem Dokument, in dem sie dargestellt werden, verknüpft, werden aber unabhängig davon in eigenen Dateien abgelegt. Diese externe Speicherung ist schon allein deshalb notwendig, da die Hilfe-Dokumente für den Benutzer nur-lesbar, d.h. unveränderbar, sind. Dadurch können die Hyperlink-Definitionen zwischen Benutzern ausgetauscht werden, ohne das gesamte Dokument transferieren zu müssen. Neben privaten Hyperlinks, die nur der erstellende Benutzer im Dokument sieht, ist es im Netzwerk möglich, öffentliche Hyperlinks zu definieren, die alle Netzwerkbenutzer im Dokument sehen können. Diese werden auf einem freigegebenen Netzwerklaufwerk abgelegt. Zur Unterscheidung der Benutzer erfolgt die Dateiablage in verschiedenen Unterverzeichnissen mit dem Namen des Benutzers.

**Lesezeichen und Anmerkungen**

Ebenso wie bei Hyperlinks gibt es auch private und öffentlich sichtbare Lesezeichen und Anmerkungen. Damit lassen sich abteilungs- oder unternehmensweit Lesezeichen anbringen bzw. Anmerkungen zu Dokumenten vornehmen.

**Journal**

Als einziges Produkt in diesem Vergleich kann mit DynaText in einem sogenannten Journal der Verlauf einer Sitzung aufgezeichnet, bearbeitet und später abgespielt werden. Denkbar sind hiermit realisierte Tutorials, die den Erstbenutzer einer zu dokumentierenden Software automatisch durch eine Auswahl der Dokumentation führen.

**Fazit**

DynaText ist ein hochinteressantes, professionelles Produkt für verschiedenste Arten von Dokumentation. Allerdings ist der Einsatz des Systems kostenintensiv, da zur Dokumentations-Erstellung die zugehörigen Publishing-Werkzeuge benötigt werden. Über ebenfalls erhältliche Entwickler-Kits ist das System an persönliche Anforderungen anpaßbar, wiederum mit dem Nachteil, daß diese Unterstützung separat zu erwerben ist.

### 3.3.2 doc.sun AnswerBook2

AnswerBook2 von Sun Microsystems ist zur Zeit die Dokumentationsumgebung, mit der die Firma Sun ihre sämtlichen Produkte – sowohl Hardware als auch Software – dokumentiert. Das Vorgängersystem AnswerBook1, welches auf DisplayPostscript basierte, wird dadurch abgelöst. Gründe dafür waren unter anderem die Voraussetzung, über ein DisplayPostscript-fähiges System zur Anzeige verfügen zu müssen, und die eingeschränkten Suchmöglichkeiten in den zugrundeliegenden Postscript-Dokumenten. Offensichtlich ist hier eine Parallele zum Vorgehen innerhalb des SESAM-2-Projekts zu erkennen. Auch hier wurde bislang Postscript als gemeinsames Dokumenten-Format verwendet und soll in Zukunft von einem besser geeigneten Format ergänzt bzw. abgelöst werden.

**Entwicklung**

Die folgende Beschreibung der Entwicklung und Fähigkeiten von AnswerBook2 stammt aus [Gutentag et al. 1997]. Beginnend mit dem Frühjahr 1997 wurde über einen Zeitraum von 3 Jahren der gesamte Bereich Online-Dokumentation der Firma Sun in einem großen Projekt neu organisiert. Ziel des neuen Systems war die Bereitstellung von Online-Dokumentation im HTML- bzw. XML-Format. Grundlage dafür ist SGML mit einer Dokumenttyp-Definition namens Solbook. Dabei handelt es sich um eine echte Teilmenge von DocBook 2.4.1, die aus Gründen der einfacheren Dokumentenerstellung in enger Zusammenarbeit zwischen den AnswerBook2-Projektbeteiligten und den Dokumentations-Autoren erarbeitet wurde. Durch die Teilmengeneigenschaft sind die entstehenden Solbook-Dokumente gleichwohl gültige DocBook 2.4.1 SGML-Dokumente und können dadurch mit bereits bestehender SGML-Software verarbeitet werden. Intern wurde bei Sun ein Autorenwerkzeug aus drei am Markt befindlichen kommerziellen Werkzeugen ausgewählt. Zusätzlich vereinfachen weitere interne Werkzeuge, beispielsweise zur Bearbeitung von Hyperlinks innerhalb von Büchern sowie zwischen Büchern, die Erstellung von Dokumentation. Die fertigen Dokumente werden in ein binäres Zwischenformat übersetzt. Aus diesem Zwischenformat generieren AnswerBook2-Server dynamisch zur Anfragezeit (on-the-fly) die Ausgabeformate zur Anzeige im Browser. Bei den Servern handelt es sich um Webserver unter Solaris. Gesteuert durch CGI-Skripten wird die Dokumentation mit Hilfe von DSSSL-Stylesheets in die jeweiligen Formate transformiert. Abbildung 3–2 zeigt den Zugriff auf das Inhaltsverzeichnis des Benutzerhandbuchs zu

**Abbildung 3–2: AnswerBook2-Benutzerhandbuch, angezeigt im Netscape-Browser**

AnswerBook2, wobei der Netscape Navigator Browser eingesetzt wird, um das HTML-Ausgabeformat des AnswerBook2-Servers anzuzeigen. Außer HTML 3.2 zur Anzeige in einem beliebigen kompatiblen HTML-Browser kann auch Postscript zum Ausdrucken von Dokumentation ausgegeben werden. Auf dem zentralen Dokumentationsserver von Sun[1] werden statt Postscript vorgenerierte PDF-Dateien für die einzelnen Bücher zum Herunterladen zur Verfügung gestellt.

**Lesezeichen**

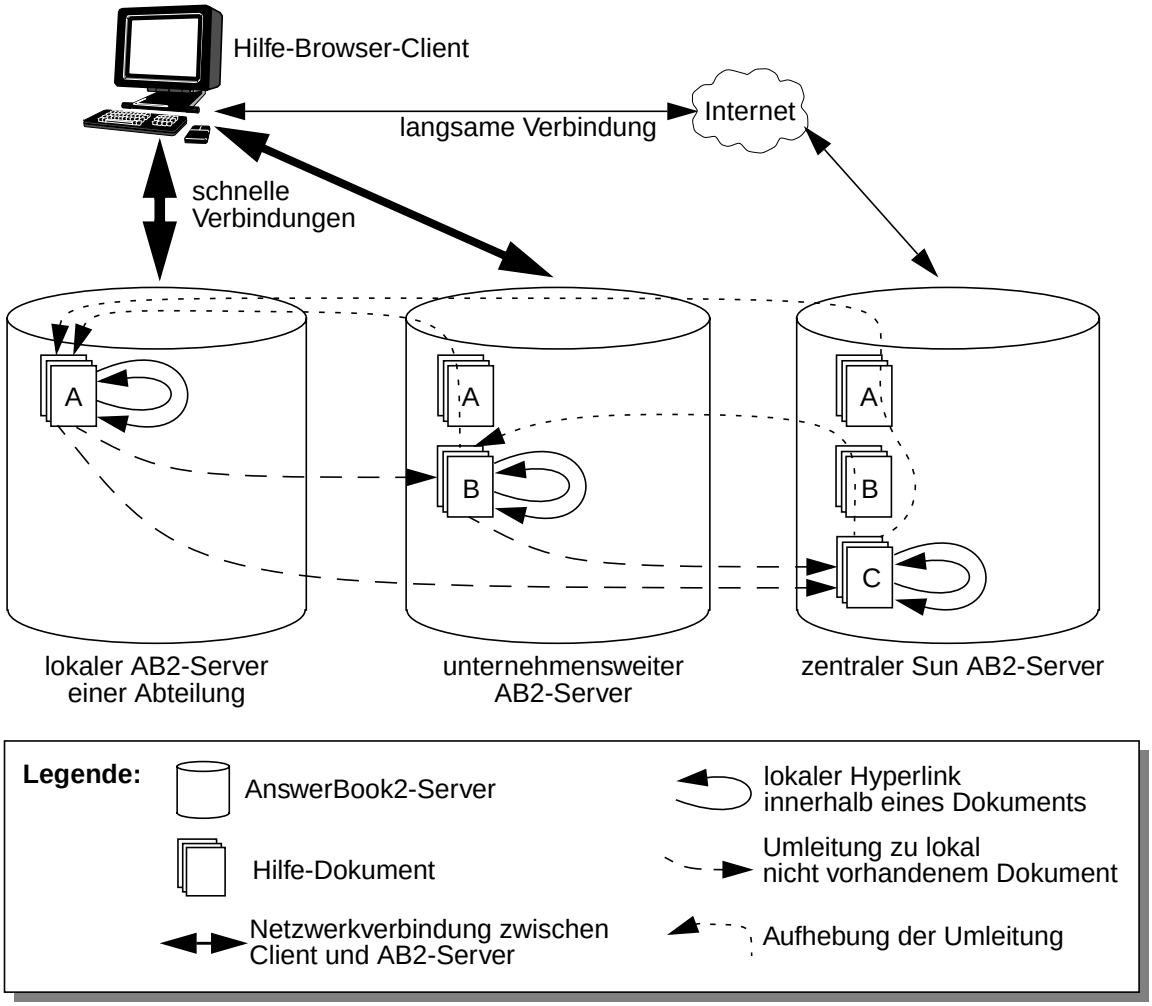
Zur Erstellung und Organisation von Lesezeichen dient die eingebaute Funktionalität des HTML-Browsers, der zum Betrachten der AnswerBook2-Dokumente verwendet wird. Gängige Browser speichern hierzu einfach den URL (Uniform Resource Locator) der betreffenden HTML-Seite. Im Web führt dieser Ansatz zu Problemen, da sich mit Änderungen an Web-Seiten oftmals der Speicherungsort oder der Name einer HTML-Seite verändert. In solchen Fällen werden die gespeicherten Lesezeichen ungültig und verweisen auf nicht mehr vorhandene oder möglicherweise andere als die ursprünglichen Ziele. Dies ist vergleichbar mit Zeigern auf nicht mehr vorhandene Datenstrukturen, den sogenannten „dangling references“, bei Programmiersprachen und wird im Web analog dazu „dangling links“ genannt. Eine der Besonderheiten von AnswerBook2 ist nun die Tatsache, daß die URLs der Hyperlinks im HTML- bzw. XML-Ausgabeformat stabil und dauerhaft sind. Dadurch lassen sich URLs als Lesezeichen verwenden, die auch nach Änderungen am verlinkten Buch niemals ungültig werden. Erreicht wird diese Eigenschaft durch die Verwendung von formalen Bezeichnern statt festen Links in den SGML-Dokumenten. Die formalen Bezeichner bleiben konstant und werden über eine Namensauflösung zu einem konkreten, für alle Zeit eindeutigen Hyperlink aufgelöst, ähnlich wie das DNS (Domain Name System) im Internet Host- und Domainnamen in Internet-Adressen auflöst. Die Auflösung findet spät statt, d.h. erst während der Aufbereitung eines Dokuments auf dem AnswerBook2-Server zur Auslieferung an den Browser.

**Skalierbarkeit**

Hohe Skalierbarkeit wird durch mehrfache, konfigurierbare AnswerBook2-Server sichergestellt. So lassen sich beispielsweise lokale Server im LAN bevorzugt ansprechen und, falls diese ein Dokument nicht vorrätig haben, Anfragen automatisch an andere weiter entfernte Server – bis hin zum zentralen Server bei Sun – umleiten. Bei einer solchen Umleitung auf andere möglicherweise externe Server ergibt sich allerdings das Problem, daß der Benutzer bei der weiteren Verfolgung von Verweisen auf einem externen Server „hängen“ bleibt. Dieses Verhalten entsteht dadurch, daß die Hyperlinks ihr Ziel relativ zur aktuellen Position adressieren. Wird nun ein Link verfolgt, so würde das Ziel zwangsläufig wieder vom externen Server geladen werden, selbst wenn die Zielressource wieder auf einem der bevorzugten lokalen Server verfügbar wäre. Zur Lösung dieses Problems verwendet AnswerBook2 bei Umleitungen Zusatzinformationen, die, im Fall von HTML als Ausgabeformat, an die URLs der Hyperlinks angehängt werden. Mit Hilfe dieser Zusatzinformationen kann ein externer AnswerBook2-Server den Benutzer durch entsprechend generierte Verweise automatisch wieder auf die ursprünglichen Server zurückleiten und somit die Umleitung aufheben. Dies ist vor allem aus Effizienzgründen erwünscht. In Abbildung 3–3 auf Seite 14 ist ein Szenario mit drei verschiedenen AnswerBook2-Servern dargestellt, die jeweils eine Teilmenge der Dokumentation bereithalten.

1.http://docs.sun.com/

**Abbildung 3–3: Umleitungen und deren Aufhebung bei AnswerBook2-Servern**




### Internationalisierung

Teil der flexiblen Konfigurierbarkeit der Dokumentations-Server ist auch die Internationalisierung sowohl der Bedienelemente als auch der Dokumentation selbst. Der Benutzer wählt hierzu eine bevorzugte Standard-Sprache und bekommt daraufhin die Dokumentation in seiner Sprache ausgeliefert. Ist ein Buch in dieser Sprache nicht vorhanden wird auf eine andere Übersetzung ausgewichen.

### XML

In einer Nachricht vom 17. Mai 1997[1] an die Mailingliste für XML-Entwickler[2] kündigte der, am XML-Standard beteiligte, Sun-Mitarbeiter Jon Bosak die Auslieferung von XML als Ausgabeformat des zentralen Sun-AnswerBook2-Servers an. Da es sich um eine direkte Abbildung der Solbook-SGML-Struktur auf eine identische XML-Struktur handelt, bleibt bei der Transformation die gesamte Struktur des Dokuments mit den darin enthaltenen Metadaten erhalten. Mit einem XML-Browser, der XLinks (XML Linking Language) unterstützt, lassen sich damit AnswerBook2-Bücher lesen. Zum testweisen Einstieg ist derzeit allerdings manuell ein URL zu bilden, der die entsprechenden Formatierungsskripten auf dem Server veranlaßt,

1.http://www.oasis-open.org/cover/bosakXMLAnswerbook.html
2.http://www.lists.ic.ac.uk/hypermail/xml-dev/

XML auszuliefern. Für Inhaltsverzeichnisse wird ein Skript namens `@xmlToc` und für Buchabschnitte ein Skript namens `@xmlChunk` verwendet. Möchte man zum Beispiel den Inhalt der AnswerBook2-Dokumentation in XML bekommen, so genügt folgende URL: `http://docs.sun.com:80/ab2/coll.522.1/ONLINEACCESS/@xmlToc`. Den Einführungsabschnitt dieses Buches fordert folgende URL an: `http://docs.sun.com:80/ab2/coll.522.1/ONLINEACCESS/@xmlChunk/171`.

**Suche**

Etwas eingeschränkt sind leider die Suchmöglichkeiten in der Dokumentation. Es gibt keine strukturelle Suche, was wahrscheinlich in der Vielfalt der Dokumentation und ihrer Struktur – auch wenn durchgehend Solbook verwendet wird – für den Benutzer schwierig wäre. Zur Auswahl stehen dem Benutzer eine komplette Volltextsuche bzw. die Suche nur in den Buchtiteln. Die Suche kann auf ein, mehrere oder alle verfügbaren Bücher limitiert werden. Außerdem stehen für Suchanfragen einfache boolesche Operatoren (and, or) zur Verfügung. Zusätzlich zur reinen Suche wird das Browsing nach Kategorien wie Büchersammlungen, Themen oder Produkten angeboten.

**Fazit**

Bedingt durch das Client-Server-Prinzip dieser Lösung ist ein solches Verfahren für die Dokumentation im SESAM-2-Projekt weniger geeignet. Das Projekt erfordert ein System für die alltägliche Arbeit mit sich ständig ändernden und neu hinzukommenden Dokumenten. Der Prozeß von der Erstellung eines neuen Dokuments über die Bereitstellung auf einem AnswerBook2-Server sowie die Formatierung und Auslieferung an einen Browser erfordert zu viele Schritte und somit Zeit. Selbst wenn zusätzliche Werkzeuge zur Unterstützung dieses Prozesses entwickelt und zur Verfügung gestellt würden, dauert der Weg zur Veröffentlichung sich ständig aktualisierender Dokumentation noch zu lange.

### 3.3.3 CDE Help Viewer

Der CDE Help Viewer ist der Standard-Browser für die Dokumentation zum Common Desktop Environment (CDE). Das CDE ist eine Benutzungsumgebung für kommerzielle Unix-Varianten wie beispielsweise Sun Solaris, HP UX oder IBM AIX. Der Browser ist eine X-Window Anwendung, die als Oberflächen-Toolkit den CDE-Standard Motif verwendet und dadurch auf den verschiedenen Unix-Systemen dasselbe, einheitliche Aussehen aufweist. In Abbildung 3–4 auf Seite 16 ist der Help Viewer unter Solaris zu sehen wie er das Benutzerhandbuch zum Hilfesystem selbst anzeigt.

**Grundlagen**

Die folgenden Informationen zur Dokumentations-Umgebung des CDE stammen von Robin Covers SGML-Web-Seiten[1]. Das Dokumentformat ist die SGML-Dokumenttyp-Definition Helptag. In diesem Format verfaßte Dokumente werden in die SGML-DTD SDL (Semantic Description Language) transformiert und wahlweise komprimiert. Daraufhin stehen sie als Dokumentation für den Help Viewer zur Verfügung. Die benötigten Programme zur Transformation und Zusammenstellung von Dokumentations-Paketen, sowie die DTDs, liegen dem CDE bei. Sie ersetzen aber keinesfalls ein Autorenwerkzeug, sondern können nur der Vorbereitung bereits geschriebener Dokumentation zur Verwendung im Hilfesystem dienen.

1. `http://www.oasis-open.org/cover/gen-apps.html#cde`

**Suche**

Obwohl die SGML-Quelldokumente strukturell ausgezeichnet sind, ist die Suche ausschließlich nach einem Stichwort in den Titeln der Bücherabschnitte möglich. Ferner stehen keine Lesezeichen zur Verfügung.

**Abbildung 3–4: Hilfe zur Hilfe im CDE Help Viewer**

**Programmierschnittstelle**

Eine Programmierschnittstelle zum Hilfesystem und der Help Viewer als graphische Komponente zur Einbettung in anderen CDE-Anwendungen vervollständigen das System zu einer standardisierten Umgebung zur Dokumentation von CDE-Software. Über die Programmierschnittstelle ist die beidseitige Interaktion zwischen Anwendung und Hilfe möglich. Einerseits wird dadurch kontextsensitive Hilfe realisiert. Andererseits lassen sich durch Aktivierung entsprechend ausgezeichneter Hilfe-Elemente Aktionen in der Anwendung auslösen. Ein Anwendungsbeispiel hierfür könnte ein Hilfe-Abschnitt zur Konfiguration eines bestimmten Anwendungsparameters sein. Hierzu bietet der Hilfe-Text beispielsweise einen Knopf an, dessen Aktivierung den zugehörigen Konfigurationsdialog der Anwendung öffnet. Der Benutzer muß also nicht mehr wissen, auf welchem Weg er diesen Dialog direkt im Programm erreicht hätte.

**Fazit**

Der geringe Benutzerkomfort dieses Hilfesystems, der sich bereits an fehlender Unterstützung für Lesezeichen und Volltextsuche ablesen läßt, führt trotz Programmierschnittstelle und Einbettung in das CDE zu einer abwertenden Beurteilung. Der CDE Help Viewer mit seinen Hilfsprogrammen hat wenig Funktionsumfang und scheint durch seine geschlossene Architektur auch nicht erweiterbar.

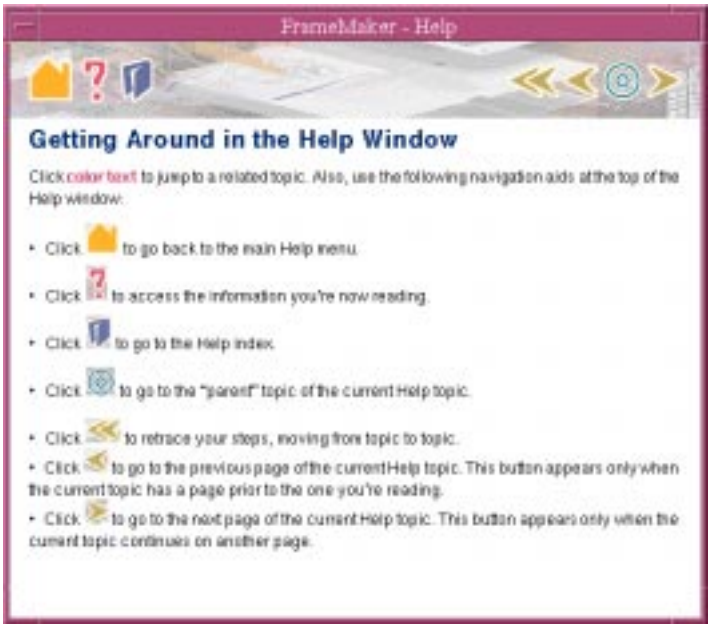
### 3.3.4 FrameMaker

FrameMaker von Adobe ist primär ein Autorenwerkzeug, das meist für technische Dokumentation eingesetzt wird. Da es sich um ein umfassendes Textverarbeitungssystem mit Unterstützung für Desktop-Publishing handelt, ist es aber keinesfalls auf dieses Anwendungsgebiet beschränkt. Mit Hilfe nur-lesbarer Dokumente läßt es sich als Browser für FrameMaker-Dokumente verwenden. Das Dokumentformat ist zwar proprietär, jedoch steht mit dem öffentlich dokumentierten Austauschformat MIF (Maker Interchange Format) ein textbasiertes Import-/Export-Format zur Verfügung.

Die Erstellung von Dokumenten erfolgt in FrameMaker nach dem WYSIWYG (what you see is what you get) Prinzip. Die Darstellung während des Bearbeitens entspricht bereits weitgehend der auf dem Ausgabemedium, in diesem Fall einem Ausdruck auf Papier.

### Hypertext

**Abbildung 3–5: Online-Hilfe im FrameMaker**




Verschiedene generierbare Dokumente, die aus Informationen in anderen Dokumenten automatisch erstellt werden können – beispielsweise Inhaltsverzeichnis, Index oder Glossar, aber auch benutzerdefinierte Listen – können zur Realisierung verschiedener Dokument-Ansichten verwendet werden. Durch den Einsatz von sogenannten Hypertext-Markern, die bei Aktivierung bestimmte Befehle auslösen, werden Hyperlinks und andere Navigationsformen – erste, nächste, letzte Seite usw. – unterstützt. Nach wie vor stellt aber die (Papier-)Seite das zentrale Unterteilungselement für Informationsknoten in FrameMaker dar. Eine andere Unterteilung wie sie für Hypertext-Systeme üblich ist, ist nicht möglich. Dies äußert sich z.B. in der beiliegenden Hilfe als teilweise umständlich, da Hilfe-Information zu einem bestimmten Thema des öfteren unnötigerweise auf eine oder mehrere weitere Seite umgebrochen werden muß und der Benutzer diese über einen Hyperlink explizit anzusteuern hat. Abbildung 3–5 zeigt eine solche Hilfe-Seite bei der oben rechts die einfachen Navigationspfeile für das sequentielle Zurück- bzw. Vorwärtswandern durch das Hilfethema andeuten, daß das Thema noch auf weiteren Seiten behandelt wird. Umgehen läßt sich dieses Verhalten, indem der Schreibschutz des Dokuments aufgehoben wird, wodurch normales sequentielles Scrolling durch den Text wieder möglich ist. Dieser Umweg ist aber sicherlich nicht im Sinne des Erfinders.

### Suche

Suchen läßt sich zum einen im Volltext eines Dokuments. Zum anderen sind sämtliche Metainformationen eines Dokuments wie beispielsweise Zeichen- und Absatzformate, Querverweise oder Textmarken über die eingebaute Suchfunktion auffindbar. Die Suchsyntax erlaubt erweiterte reguläre Ausdrücke, womit sich auch kompliziertere Anfragen stellen lassen.

### Fazit

Da FrameMaker auch die Konvertierung seiner Dokumente in Formate wie HTML und XML erlaubt, ist prinzipiell eine Online-Darstellung möglich. Allerdings ist die Konvertierung explizit zu veranlassen. Danach steht die Suchfunktion des FrameMakers natürlich nicht mehr zur Verfügung. Darüberhinaus hängt es ausschließlich vom Dokument-Autor und von zur Verfügung gestellten einheitlichen, konsistenten Formatvorlagen (Absatz-, Zeichenformat, Querverweise, Marker) ab, inwieweit die erstellten Dokumente einem Hypertext-System gerecht werden.

### 3.3.5 Panorama Viewer

Beim Panorama Viewer von SoftQuad handelt es sich um ein Webbrowser-Plugin für Netscape Navigator und Microsoft Internet Explorer. Das Plugin stellt einen SGML-Browser zur Verfügung. Es wird aktiviert durch die Dateiendung „`.sgm`“ bzw. „`.sgml`“ oder durch MIME-Types, die auf ein SGML-Dateiformat hinweisen.

Panorama Viewer ist ein Browser, der beliebige SGML-Dokumente anzeigen kann. Zur sinnvollen Darstellung sind allerdings Stylesheets notwendig, die verschiedene Ansichten direkt zur Laufzeit aus dem Dokument erzeugen. Die Darstellung am Bildschirm erfolgt horizontal zweigeteilt. Im einen Fensterteil wird ein Baum als Gliederungsansicht und im anderen der formatierte Fließtext des Dokuments dargestellt. Vom Dokument-Autor lassen sich für beide Darstellungsarten jeweils verschiedene Ansichten vordefinieren, aus denen der Benutzer seine Auswahl trifft.

**Hyperlinks, Lesezeichen und Anmerkungen**

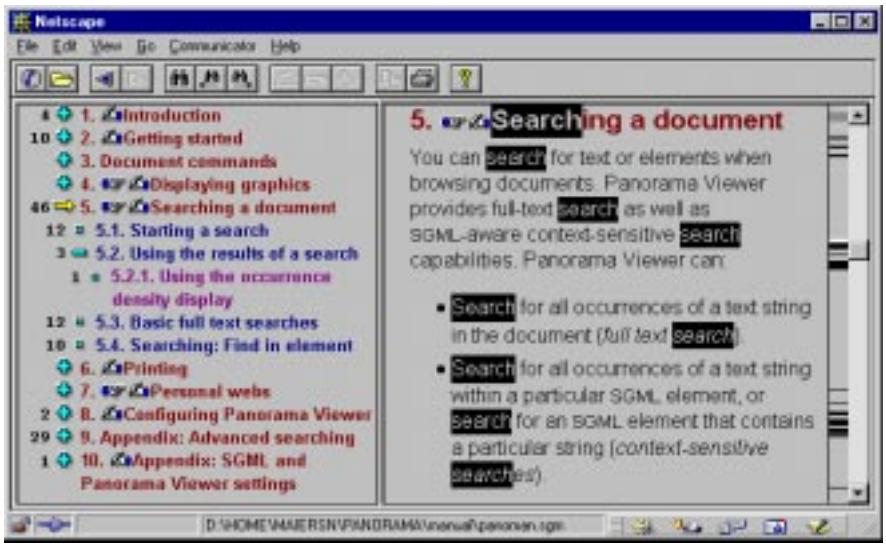
Neben den im Dokument definierten, SGML-typischen Hyperlinks (einfach/mehrfach, einseitig/zweiseitig) lassen sich benutzerdefinierte Links erstellen. Diese werden ebenso wie Anmerkungen und Lesezeichen (hier eine Sonderform der Anmerkung ohne Anmerkungstext) in sogenannten Web-Dateien abgelegt. Diese sind nicht zu verwechseln mit dem World Wide Web. Vielmehr deutet der Name an, daß es sich um ein Netz von Hyperlinks, Lesezeichen und Anmerkungen handelt, die das Dokument umgeben bzw. wie eine transparente Schicht über ihm liegen. Mehrere Web-Dateien können während einer Plugin-Sitzung geöffnet (mounted) sein, wodurch die enthaltenen Einstellungen wirksam werden. Bei Änderungen nimmt ein, aus den geöffneten, ausgewähltes aktives Web diese auf. Die übrigen geöffneten Webs sind schreibgeschützt. Gespeichert werden die Webs in einem standardisierten SGML-Format.

**Suche**

Panorama Viewer erlaubt eine wahlweise strukturierte Volltextsuche auf dem geöffneten Dokument. Reguläre Ausdrücke in der Suchsyntax ermöglichen eine sehr flexible Suche im Text. Die Bedeutung besonderer Such-Metazeichen bei regulären Ausdrücken, wie der Zeilenanfang (^) oder das Zeilenende ($), sind an die SGML-Struktur angepaßt. Dies ist notwendig, da ein SGML-Dokument zunächst keine signifikanten Zeilenumbrüche wie beispielsweise ASCII-Text enthält. Leerraum wird zur besseren Lesbarkeit des SGML-Quelltextes verwendet. Nur innerhalb ausgewählter SGML-Elemente sind die Zeilenumbrüche und anderer Leerraum signifikant für die spätere Darstellung. Diese Elemente sind aber einem allgemeinen Browser wie dem Panorama Viewer nicht bekannt. Der Benutzer übernimmt bei der Suche die Aufgabe, sich an den Grenzen von Elementen statt an Zeilenumbrüchen orientieren. So findet man mit ‚^‘ Text direkt nach dem beginnenden Start-Tag des umgebenden SGML-Elements. ‚$‘ bezeichnet das Textende beim schließenden End-Tag des umgebenden Elements.

Die Suchergebnisse werden im Fließtext markiert hervorgehoben. Eine Liste der Treffer existiert nicht aber eine Dichteverteilung der Suchtreffer. Diese ist dargestellt als Balken parallel zur vertikalen Scroll-Leiste. Für jeden Treffer ist ein horizontaler Strich auf einer y-Position eingetragen, die der Trefferposition relativ zum gesamten Dokument entspricht. Durch Anklikken eines Trefferstrichs mit der Maus springt der Text zur entsprechenden Fundstelle. In Abbildung 3–6 ist der Panorama Viewer als Plugin im Netscape Navigator Browser nach der Durchführung einer Suchanfrage dargestellt. Neben der beschriebenen Suchtrefferdichte in der rechten Fensterhälfte zwischen dem Fließtext und der vertikalen Scroll-Leiste ist die Hervorhebung der Suchtreffer im Fließtext zu sehen. Darüberhinaus tragen die Knoten im Baum des

**Abbildung 3–6: Panorama Viewer nach erfolgreicher Suche in einem SGML-Dokument**




Inhaltsverzeichnisses in der linken Fensterhälfte zu ihrer Linken jeweils die akkumulierte Anzahl der Trefferstellen innerhalb des gesamten, zugehörigen Unterbaums.

**Fazit**

Das Panorama Viewer System ist sehr interessant, zumal sich unter anderem mit ein wenig manueller Nachhilfe DocBook-SGML-Dokumente mit ihm darstellen und durchsuchen lassen[1]. Allerdings ist diese Software schon etwas älter und es existieren keine neueren Versionen des Viewers. Auch unter dem Gesichtspunkt, daß zur Erstellung von Stylesheets zusätzlich die separaten Publishing-Werkzeuge benötigt werden, ist fraglich, inwieweit noch Unterstützung seitens des Herstellers zu erhalten ist. Außerdem kommt hinzu, daß es sich beim Panorama Viewer um ein Browser-Plugin handelt. Es wird also zum Beispiel ein Netscape Navigator zur Ausführung benötigt, wobei das Plugin zudem plattformabhängig ist und nur unter den 32-Bit Windows-Varianten auf Intel-kompatiblen Rechnern lauffähig ist.

### 3.3.6 JavaHelp

JavaHelp von JavaSoft ist eine plattformunabhängige Online-Hilfe-Umgebung. Sie wird von JavaSoft favorisiert als Standard-Hilfesystem, sowohl für reine Java-Anwendungen, als auch für Anwendungen aus nativem, plattformabhängigem Code. Die Spezifikation von JavaHelp beinhaltet das Dokumentformat in Form von HTML 3.2, Metadatenformate – für Hilfepakete, Inhaltsverzeichnisse und Indizes – in verschiedenen XML-DTDs, sowie eine Programmierschnittstelle mit zugehöriger Referenzimplementierung.

**Abstraktion von eingesetztem Dokumenten-Format**

Zur Abstrahierung vom grundlegenden HTML-Dokumentformat werden in den Metadateien eines Hilfepaketes (Paketdefinition, Inhaltsverzeichnis, Index) und bei Benutzung der Programmierschnittstelle eindeutige Bezeichner statt direkter HTML-Links verwendet. Diese Abstraktionsstufe entkoppelt die Bezeichner von Änderungen in den HTML-Dokumenten und ermöglicht prinzipiell auch die Verwendung eines gänzlich anderen Dokumentformates als HTML. Die Abbildungen der Bezeichner auf konkrete Dokument-Positionen sind in soge-

1. Die mitgelieferten Stylesheets für DocBook passen zu einer alten Version von DocBook und haben offensichtlich kleine Fehler, welche sich aber intuitiv ohne Dokumentation entsprechend beheben lassen, so daß sich tatsächlich DocBook-Dokumente in einer sehr simplen Darstellung betrachten lassen.

nannten Maps abgelegt. Erlaubte Positionen sind typische HTML-URLs, die andere HTML-Dokumente innerhalb desselben Hilfepakets bzw. bestimmte Ankerstellen darin referenzieren. Außerdem erlaubt sind Verweise auf andere Hilfepakete durch die Verwendung des JAR-Protokolls. Dieses Protokoll existiert ab der Java-2-Plattform, um Dateien direkt in einem Java-Archiv zu benennen und darauf zuzugreifen, ohne dieses Archiv dazu vorher entpacken zu müssen.

**Aktive Inhalte**

In den HTML-Dokumenten lassen sich als Besonderheit beliebige Java-Komponenten des graphischen Oberflächen-Toolkit Swing bzw. davon abgeleiteten Komponenten einsetzen. Dadurch wird Interaktion der Hilfe mit dem Anwendungsprogramm möglich. Beispielsweise kann in der Hilfe darauf hingewiesen werden, ein bestimmtes Dialogfenster aufzurufen, um zum gewünschten Ergebnis zu kommen. Über eine Knopf-Komponente im Hilfe-Dokument ließe sich nun das Dialogfeld der Anwendung direkt vom Benutzer öffnen, ohne wissen zu müssen über welchen Weg er normalerweise diesen Dialog aufgerufen hätte.

**Automatische Zusammenführung von Hilfepaketen**

JavaHelp ist nicht nur für alleinstehende Anwendungen gedacht, sondern auch zur Dokumentation von Java-Komponenten, den sogenannten JavaBeans. Solche Komponenten kommen oft in einer Sammlung als Klassenbibliothek zum Programmierer. Zusätzlich setzt dieser möglicherweise verschiedenste solcher Klassenbibliotheken ein und möchte gerne die Dokumentation aller verwendeten Komponenten auf einmal zur Verfügung haben. Zu diesem Zweck kann JavaHelp einzelne Hilfepakete automatisch zusammenführen. Somit ist dann die Suche in Index und Volltext über die zusammengeführten Hilfen möglich.

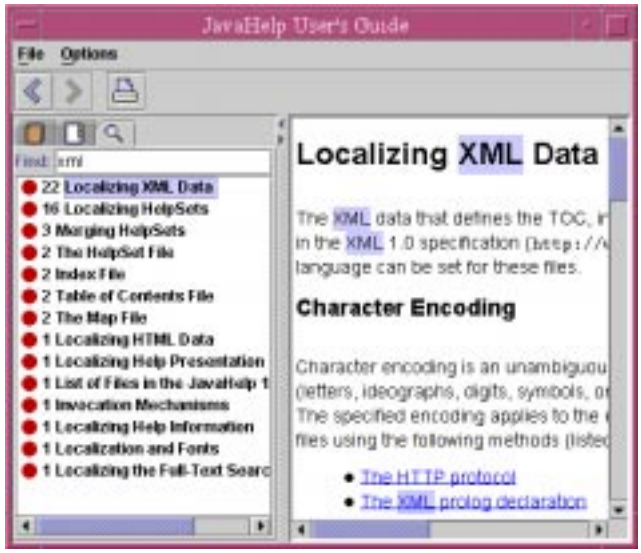
**Internationalisierung**

Sämtliche Bestandteile eines Hilfepakets sind internationalisierbar. Dies bedeutet, daß dem Benutzer verschiedene Sprachen in der Benutzerführung und den Hilfe-Dokumenten zur Verfügung gestellt werden, um eine bevorzugte Sprache auswählen zu können.

**Suche**

Das Suchen im Hilfepaket erfolgt über einen vorgenerierten Volltextindex. Die in der Referenzimplementierung enthaltene Suchmaschine läßt nur die Angabe von einem oder mehreren Suchbegriffen ohne Operatoren zu. Dafür verwendet die Maschine aber für die Erstellung des Suchergebnisses grammatikalische Regeln, die beispielsweise Deklinationen eines Wortes nicht unterscheiden und als ein und dasselbe Wort betrachten (morphing). Schließlich folgt eine Trefferein-stufung, basierend auf der Auftrittswahrscheinlichkeit, Reihenfolge der Suchbegriffe und anderen Kriterien (relaxation ranking). In Abbildung 3–7 ist der JavaHelp-Browser zu sehen, nachdem eine Volltextsuche zu dem Begriff „xml“ veranlaßt wurde. In der linken Fensterhälfte sind die nach Relevanz sortierten Suchtreffer zu

**Abbildung 3–7: JavaHelp nach erfolgreicher Volltextsuche**



sehen. Nach der Auswahl des ersten Treffers zeigt die rechte Fensterhälfte den zugehörigen Text an und hebt die Treffer, durch Rechtecke hinterlegt, hervor.

Da JavaHelp-Hilfepakete auch über das Netzwerk mit Hilfe von HTML-Browsern in Verbindung mit einem Java-Applet angezeigt werden können, ist es möglich, die Volltextsuche auf einem Server ausführen zu lassen. Dadurch können Netzwerkbandbreite gespart und lange Wartezeiten vermieden werden. Ansonsten müßte der gesamte Volltextindex vor der Suche komplett auf den Client geladen werden.

**Fazit**

JavaHelp ist ein offenes, durchdachtes, plattformunabhängiges und umfangreiches Hilfesystem, für das auch eine Referenzimplementierung vorliegt. Ausgehend von dieser Implementierung läßt sich das System an spezifische Benutzeranforderungen anpassen. Seine Plattformunabhängigkeit, die vor allem durch die Implementierung in Java erreicht wird, geht soweit, daß das System nicht nur für die Verwendung als Hilfesystem in Java-Programmen geeignet ist, sondern auch für beliebige Anwendungen in anderen Programmiersprachen. Einzig die derzeitige Verwendung von HTML 3.2 als Auszeichnungssprache schränkt die Verwendung vor allem innerhalb des SESAM-2 Projekts sehr ein. HTML erlaubt keine strukturelle, semantische Auszeichnung wie sie beispielsweise für eine strukturelle Suche benötigt wird. Unabhängig davon ist anzumerken, daß Konvertierungs-Stylesheets für DocBook existieren, mit deren Hilfe jederzeit eine Überführung von DocBook-Dokumenten in das JavaHelp-Format möglich ist. Dies schließt vor allem auch die automatische Generierung der benötigten Metadaten (Paketdefinition, Inhaltsverzeichnis, Index) ein.

### 3.3.7 HTML Help

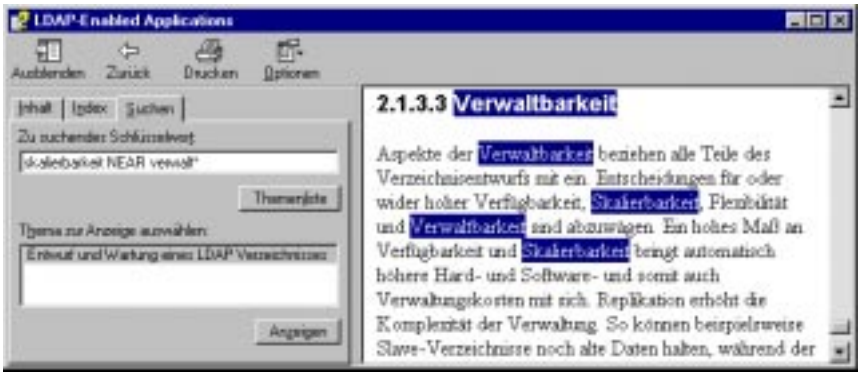
HTML Help von Microsoft ist eine Online-Hilfeumgebung, die hauptsächlich unter den verschiedenen Microsoft Windows Betriebssystemen zum Einsatz kommt. Zwar ist das grundlegende Hilfe-Dokumentformat HTML, jedoch fördert Microsofts Dokumentation zu HTML Help den Einsatz aller Möglichkeiten, die die Microsoft Internet Explorer Webbrowser ab der Version 3.x und höher zu bieten haben. Dies beinhaltet vor allem auch aktive Inhalte, wie ActiveX-Komponenten, die ausschließlich auf dieser Plattform lauffähig sind. Dies ist nicht weiter verwunderlich, basiert der zugehörige Hilfe-Browser doch auf der ActiveX-Komponente des Internet Explorer. Hält sich der Dokument-Autor allerdings an offene Standards, wie beispielsweise HTML 3.2, ist es durchaus möglich, plattformunabhängige Hilfepakete zusammenzustellen. Ein beliebiger HTML-Browser in Zusammenspiel mit einem Java-Applet ist zur Anzeige einer solchen Dokumentation ausreichend. Einzig die Volltextsuche ist in dieser Konfiguration dann nicht möglich.

**Suche**

Die Volltextsuche in einem vorgenerierten Volltextindex erlaubt die Angabe boolescher Operatoren (and, or, not). Außerdem läßt sich spezifizieren, daß Suchbegriffe in örtlicher Nähe zueinander stehen sollen (near). Abbildung 3–8 zeigt das Ergebnis einer Suche mit Hilfe des near-Operators. Suchtreffer beschränken sich dann auf das Vorkommen eines Stichwortes „Skalierbarkeit“ sowie eines Wortes, das mit „Verwalt“ beginnt. Durch den near-Operator müssen diese beiden Begriffe innerhalb von 8 aufeinanderfolgenden Wörtern liegen. Die Liste mit den Suchergebnissen läßt sich nach den Titeln der gefundenen Textstellen oder einer Treffereinstufung sortieren. Die formulierten Anfragen werden benutzerspezifisch, dauerhaft gespeichert und sind mit dem Hilfepaket, in dem sie gestellt wurden, assoziiert. Bei einer Wiederöffnung eines Hilfepakets stehen die ehemals durchgeführten Suchanfragen des Benutzers in einer Auswahl-Box wieder zur Verfügung. Zur Eingrenzung des Suchergebnisses, läßt sich

ein vorhergehendes Ergebnis als Basis für neue Suchanfragen verwenden. Damit kann auch in sehr großen Hilfepaketen, wie beispielsweise dem MSDN (Microsoft Developers Network), das gewünschte Thema aufgefunden werden, ohne zu einer komplexen Anfrage gezwungen zu werden, die bereits in einem einzigen Suchvorgang das gewünschte Ergebnis bringt.

**Abbildung 3–8: Suchfunktion in HTML Help**




**Fazit**

Der Ansatz, eine bereits vorhandene HTML-Browser-Komponente für ein Hilfesystem einzusetzen gleicht der Vorgehensweise bei JavaHelp. Im Gegensatz zu JavaHelp ist HTML Help als plattformabhängig einzustufen. Zwar lassen sich Hilfe-Dokumente wie oben beschrieben mit einem HTML-Browser und einem Java-Applet betrachten, die äußerst wichtige und umfangreiche Suchfunktionalität steht in dieser Konstellation aber nicht zur Verfügung und macht somit den Einsatz fragwürdig. Unabhängig davon ist zu bemerken, daß auch für HTML Help als Zielformat Stylesheets für DocBook-Dokumente existieren, die ebenfalls die benötigten Metadaten zu einem Hilfepaket automatisch generieren.

# 4 Anforderungen

Aus den Ergebnissen des im vorhergehenden Kapitel beschriebenen Systemvergleichs und Gesprächen mit dem Betreuer dieser Studienarbeit, Stefan Krauß, der in diesem Fall die Position des Kunden in Bezug auf das SESAMDoc-Projekt eingenommen hat, ergaben sich die in den folgenden Abschnitten dargestellten Anforderungen an ein Online-Hilfesystem.

## 4.1 Allgemeine Anforderungen

In diesem Abschnitt werden allgemeine Anforderungen an das Online-Hilfesystem festgehalten. Diese betreffen unter anderem die Benutzungsoberfläche, die Eigenschaft eines Hypertext-Systems, das grundlegende Dokumentenformat, Autorenwerkzeuge, den Export von Dokumenten und die Plattformen, auf denen ein System verfügbar ist.

**Graphische Benutzungsoberfläche**

Beinahe selbstverständlich ist die Forderung nach einer zeitgemäßen graphischen Benutzungsoberfläche für das Hilfesystem. Diese Oberfläche ermöglicht Benutzern – vom Einsteiger bis hin zum Fortgeschrittenen – auf einfache Weise mit dem Programm zu interagieren. Neben der reinen Bedienung mit der Maus sollte allerdings für fortgeschrittene Anwender ein Großteil der Funktionalität auch über die Tastatur zugänglich gemacht werden, um schnelles, effektives Arbeiten zu ermöglichen. Über die Tastatur müssen mindestens die folgenden Aktionen möglich sein: die Umschaltung zwischen den verschiedenen Ansichten der Hilfe-Dokumente, die Navigation innerhalb einer Ansicht sowie die Auswahl von Menüpunkten aus einer Menüleiste.

**Hypertext-System**

Das Hilfesystem sollte ein Hypertext-System sein. Dies bedeutet, daß das System die Funktionalität von Hyperlinks unterstützen muß. Hyperlinks sind logische Verknüpfungen von Ressourcen, in diesem Fall von Dokumenten bzw. Positionen in diesen Dokumenten. Grundsätzlich wird deren Funktionalität unterschieden nach der Richtung, in der sie wirksam sind und nach der Anzahl an Ressourcen, die sie verknüpfen. Bezüglich der Richtung soll das System einseitige (unidirektionale) und zweiseitige (bidirektionale) Links zur Verfügung stellen. Einseitige Links erlauben nur den Sprung zur Zielposition, zweiseitige erlauben zusätzlich auch den Sprung vom Ziel wieder zurück zur Ausgangsposition. Unabhängig von dieser Eigenschaft, muß ein Link die Möglichkeit bieten, nur zu einem verwiesenen Ziel (einfach) oder zu einem ausgewählten Ziel aus beliebig vielen (mehrfach) zu springen. Neben den von einem Dokument-Autor verfaßten Hyperlinks sollten dem Benutzer auch selbst definierbare Links zur Verfügung stehen. Diese können zum einen privat abgelegt werden, so daß sie nur für den anlegenden Benutzer sichtbar sind. Zum anderen existieren öffentliche Hyperlinks im System. Sie werden über das Netzwerk anderen Benutzer zugänglich gemacht und sind für all diese sichtbar. Die Aktivierung eines Hyperlinks durch den Benutzer – beispielsweise durch einen Mausklick – ersetzt wahlweise entweder die aktuelle Fließtext-Ansicht durch eine Ansicht der verwiesenen Zielposition oder öffnet das Ziel in einer neuen Ansicht. Zur Thematik Hypertext und den möglichen Zugriffstrategien auf dessen Informationsraum sei an dieser Stelle auch auf Abschnitt 3.2 „Gemeinsamkeiten der untersuchten Produkte“ auf Seite 8 zurückverwiesen.

**Suche**

Optimale Suchmöglichkeiten müssen durch ein Datenformat der Hilfe-Dokumente, das strukturelle Auszeichnung unterstützt, gewährleistet werden. Auf diese Weise wird von der visuel-

len Darstellung abstrahiert und das Einbringen wichtiger struktureller und semantischer Information in die Dokumente gewährleistet. Die hierdurch enthaltenen Informationen lassen eine gezielte Suche, sowohl nach Inhalt als auch nach Metainformation zu. Bei Verzicht auf ein proprietäres Dokumentenformat führt diese Anforderung zur Verwendung von standardisierten Formaten wie SGML bzw. XML. In diesem Fall ist das Format DocBook-XML durch die Aufgabenstellung entsprechend vorgegeben.

**Autorenwerkzeug**

Zur Erstellung der Hilfe-Dokumente wird ein Autorenwerkzeug benötigt. Ansonsten wäre der Aufwand zur Erzeugung von Online-Hilfen bei ernsthafter Anwendung zu groß. Das Werkzeug führt den Dokument-Autor durch die verschiedenen Erstellungsschritte. Dazu gehören die Erfassung des Textes, dessen strukturelle Auszeichnung sowie evtl. das Generieren von Hilfepaketen, die aus mehreren Dokumenten bestehen.

**Export von Dokumentinhalt**

Für den Benutzer einer Online-Hilfe ist es wünschenswert, wenn er die angebotene Information weiterverarbeiten kann. Dazu ist es notwendig, Teile der Hilfe exportieren zu können. Als einfachstes, allen Anwendungen gemeinsames Format wird unformatierter Text im ASCII-Format – oder evtl. in einem der ISO- oder Unicode-Zeichensatz-Formate – verwendet. Darüber hinaus ermöglicht der Export im strukturierten Datenformat XML die Weiterverarbeitung von Hilfe-Texten durch den Benutzer ohne Verlust an Information.

**Plattformunabhängigkeit**

Das Hilfesystem muß auf mehreren Plattformen zur Verfügung stehen. Speziell bei plattformunabhängigen Hilfe-Inhalten ist es wichtig, daß das System die persönlich bevorzugte Plattform des Benutzers unterstützt. Idealerweise ist ein Hilfesystem durchgängig plattformunabhängig. Dies ist beispielsweise erreichbar durch die Entwicklung mit einer portierbaren Programmiersprache und ebenfalls portierbaren Bibliotheken. Portierbare Programmiersprachen sind durch den Einsatz von Bytecode-Interpretern bzw. Softwaremaschinen, die ein virtuelles System simulieren, möglich. Zu diesen Sprachen zählen unter anderem Java, Python, Perl, Scheme oder Tcl/Tk.

## 4.2 Ansichten der Hilfe-Dokumente

Dieser Abschnitt beschreibt, welche verschiedenen Ansichten der Hilfe-Dokumente das Online-Hilfesystem darstellen sollte. In Anlehnung an das Software-Entwurfsmuster „model view controller“ (MVC; Modell Ansicht Steuerung) ist unter einer Ansicht eine bestimmte visuelle Darstellung eines Hilfe-Dokuments zu verstehen. Das Dokument dient dabei als Modell, das die reinen Daten enthält. Die Ansicht verwendet diese Daten, um eine angepaßte Darstellung vorzunehmen. Der „controller“ dient im oben genannten Entwurfmuster als Schnittstelle zwischen Benutzer und dem Datenmodell. Er soll hier keine weitere Rolle spielen. Stattdessen wird das Verhalten der Ansichten in Bezug auf Benutzeraktionen informell beschrieben. Neben der Darstellung des eigentlichen Hilfe-Inhalts als formatierter Fließtext muß ein Hilfesystem die folgenden spezialisierten Ansichten eines Hilfe-Dokuments zur Verfügung stellen: Ein Inhaltsverzeichnis, das die Dokument-Gliederung ersichtlich macht sowie einen Index zur Suche nach bestimmten Begriffen. Eine Benutzerschnittstelle zur Ausführung von Suchanfragen wird ebenfalls in diesem Zusammenhang beschrieben, da die Suche im Dokument erfolgt und die Suchergebnisse insofern eine spezialisierte Dokument-Ansicht darstellen.

### 4.2.1 Fließtext

Um dem Benutzer den Zugang zum eigentlichen Hilfe-Inhalt zur ermöglichen, muß er vom System als formatierter Fließtext dargestellt werden. Diese Darstellung ist die wichtigste Ansicht eines Online-Hilfesystems. Teil dieser Ansicht sind Hyperlinks. Sie sind optisch erkennbar durch eine spezielle Formatierung hervorzuheben, sowie durch einen bei Berührung geänderten Mauszeiger kenntlich zu machen. Paßt der automatisch umgebrochene Text nicht in das umgebende Fenster, erhält dieser eine vertikale Scroll-Leiste, womit sich der sichtbare Ausschnitt verändern läßt. Falls weiterhin die horizontale Abmessung des umgebenden Fensters nicht ausreicht, wird auch in dieser Richtung eine Scroll-Leiste zur Verfügung gestellt.

Die Ansicht muß entweder das gesamte Hilfepaket als durchgehenden Fließtext darstellen oder als Fließtext-Abschnitte, die durch Unterteilung auf einer sinnvollen Ebene der Dokumentgliederung entstehen. Falls eine unterteilte Ansicht verwendet wird, muß dem Benutzer die Möglichkeit gegeben werden, zum jeweils nächsten bzw. vorhergehenden logischen Dokumentteil zu springen. Dies ermöglicht das sequentielle Durchlesen der Hilfe.

Bei Einsatz einer unterteilten Ansicht stellt sich die Frage, wie eine sinnvolle Aufteilung in einzelne Informationsknoten vorgenommen werden kann. Prinzipiell sollte bei einer kontextsensitiven Hilfe nur die Information angezeigt werden, die im aktuellen Zusammenhang von Bedeutung ist. Werden Hilfe-Dokumente nun in der Auszeichnungssprache DocBook geschrieben, so ist ihre Gliederung von der vorgegebenen Buchform beeinflußt. Bei der Erstellung der Dokumente muß darauf geachtet werden, daß die Gliederung in Abschnitte erfolgt, die jeweils nur Information zu einem bestimmten Kontext enthalten. Bei der Anzeige im Browser gilt es dann, durch Aufteilung der Dokumente in ihre Abschnitte, wieder die Information zu den jeweiligen Kontexten zu trennen. Da der Umfang von Information zu einem bestimmten Zusammenhang sehr unterschiedlich sein und vom Dokument-Autor frei gewählt werden kann, ist eine simple Unterteilung in Abschnitte möglicherweise nicht ausreichend. Hier sollte ein Algorithmus den Informationsumfang abschätzen können und bei zu kleinen Abschnitten evtl. weitere nachfolgende zur Anzeige mithinzunehmen. Bei zu großen Abschnitten, sollte wiederum eine noch tiefergehende Aufteilung vorgenommen werden, um den Benutzer nicht mit zu viel Information auf einmal zu überfluten. Verschiedene Ansätze zur Aufteilung von Hilfe-Dokumenten befinden sich in Abschnitt 8.3 „Mögliche Erweiterungen“ auf Seite 72.

### 4.2.2 Inhaltsverzeichnis

Das Hilfesystem muß die Gliederung eines Hilfe-Dokuments in einem Inhaltsverzeichnis darstellen. Das Verzeichnis besitzt eine hierarchische Form und wird als Baum dargestellt. Die einzelnen Baumknoten und die daran hängenden Unterbäume sind vom Benutzer ein- und ausklappbar. Auf diese Weise kann eine übersichtliche Teilansicht hergestellt werden. Die Baumknoten werden entsprechend dem Dokumentteil, den sie repräsentieren, mit Sinnbildern versehen. Somit werden dem Benutzer die verschiedenen Formen der Hilfe-Abschnitte ersichtlich. Vom Hilfesystem müssen die folgenden Gliederungsstufen aus DocBook dargestellt werden: Buchsammlung, Buch, Artikel, Kapitel und Abschnitte. Dies entspricht den DocBook-Elementen set, book, article, chapter und section. Sofern vom Dokument-Autor verwendet, müssen die zum Element section alternativen Abschnitts-Elemente sect1 bis sect5 ebenfalls in der Gliederung angezeigt werden. Die Beschriftung der Knoten ergibt sich aus den Titeln der zugehörigen Dokumentabschnitte.

Bei der Auswahl eines Baumknotens durch den Benutzer wird das zugehörige Dokument zur Anzeige gebracht bzw. springt die Fließtext-Ansicht des Hilfe-Dokuments an die zugehörige Dokumentposition.

Der Benutzer muß über ein Kontextmenü für jeden Baumknoten Funktionalitäten, die den jeweiligen Knoten bzw. dessen Unterbaum betreffen, auslösen können. Notwendige Aktionen hierfür sind:

- sämtliche Knoten in diesem Unterbaum vollständig, rekursiv ausklappen,
- den gesamten Baum von der Wurzel aus bis zur Tiefe dieses Knotens ausklappen, um dem Benutzer die Möglichkeit zu geben, die Gliederung bis zu einer bestimmten Tiefe expandiert anzuzeigen und darüberhinausgehende Details zu verbergen,
- eine Suche beschränkt auf den Inhalt dieses Unterbaums,
- Kopieren des Inhalts dieses Unterbaumes in die Zwischenablage, wahlweise als unformatierten Text oder im XML-Format,
- die Dokumentposition dieses Knotens als Lesezeichen merken,
- den Beginn bzw. das Ziel eines benutzerdefinierten Hyperlinks an diesem Knoten festlegen.

Nach der Durchführung einer Suchanfrage kann entweder im Gesamtinhaltsverzeichnis oder in einem auf Suchtreffer reduzierten Baum die Anzahl der Treffer im Unterbaum eines jeden sichtbaren Knotens angezeigt werden. Dies erlaubt dem Benutzer, auf einen Blick festzustellen, in welchem Teil der Hilfe sich Treffer häufen und somit wahrscheinlich relevante Textstellen schnell aufzufinden.

### 4.2.3 Index

Das Hilfesystem muß einen Index zu Begriffen anbieten, die der Dokument-Autor für die Aufnahme in einen Index vordefiniert hat. Der Index ist ebenfalls wie das Inhaltsverzeichnis hierarchisch aufgebaut. Allerdings beschränkt er sich auf eine Tiefe von 3 Stufen. Die oberste Stufe enthält primäre Indexbegriffe. Die optionale 2. Stufe enthält beliebig viele sekundäre Indexbegriffe, die mit dem darüberstehenden Begriff in Beziehung stehen. Die 3. Stufe enthält analog keinen oder mehrere tertiäre Indexbegriffe, die zu einem vorhandenen sekundären Begriff in Beziehung stehen. Als Ansicht des Index wird ein Baum verwendet. Sinnbilder sind für die Knoten nicht erforderlich. Die Beschriftung der Baumknoten erfolgt mit dem jeweiligen Indexbegriff. Die 1. Stufe sowie die jeweils untergeordnete 2. bzw. 3. Stufe müssen aufsteigend alphabetisch sortiert werden.

Bei der Auswahl eines Baumknotens durch den Benutzer springt die formatierte Fließtext-Ansicht des Hilfe-Dokuments an die Dokument-Position, an welcher der Indexbegriff definiert wurde.

Ein dem Baum zugeordnetes Texteingabefeld dient der Suche nach Indexbegriffen. Die Suche erfolgt ohne Berücksichtigung der Groß-/Kleinschreibung und findet den alphabetisch ersten Indexbegriff, dessen Wortanfang mit dem eingegebenen Suchbegriff übereinstimmt. Bei erfolgreicher Suche springt die Markierung auf den betreffenden Baumknoten.

### 4.2.4 Suche

Zusätzlich zur Suche nach Indexbegriffen wird oftmals eine Suche im Volltext der Hilfe benötigt. Darüber hinaus soll die Suchanfrage nicht nur Stichworte im Volltext spezifizieren können, sondern die Struktur der Hilfe-Dokumente zur Eingrenzung der Suchbereiche berücksichtigen. Hierzu kommen vom Dokument-Autor vordefinierte Anfragetypen zum Einsatz. Auf diese Weise wird von der bisweilen komplizierten Dokument-Struktur abstrahiert. Der Benutzer wählt anhand einer beschreibenden Bezeichnung einen Anfragetyp aus einer Liste aus. Passend zur Anfrage werden Eingabefelder für benötigte Angaben angezeigt, die der Benutzer ausfüllen kann. Anschließend kann die Suchanfrage ausgeführt werden, wobei das

System die Benutzereingaben auswertet und intern für die Erstellung und Durchführung einer Suchanfrage verwendet.

Dem fortgeschrittenen Benutzer werden möglicherweise die vordefinierten Anfrage-Typen nicht ausreichen. Deshalb muß ihm die direkte Eingabe einer Suchanfrage, die einer bestimmten Syntax genügt, angeboten werden. Diese Syntax muß die Suche in der Dokumentstruktur in Kombination mit der Suche nach Teilzeichenketten im Dokumenttext erlauben. Beispielsweise muß eine Suche nach einer Zeichenkette im Titel aller Abschnitte, die in der Gliederung direkt unterhalb eines Kapitels stehen, formulierbar sein. Eine Sprache, die diese Funktionalität auf strukturierten XML-Dokumenten ermöglicht, ist die XML Path Language (XPath).

Suchanfragen, die der Benutzer während seiner Sitzungen stellt, sollten in einer Art Logbuch festgehalten werden. Dabei handelt es sich um eine Liste, die für jede getätigte unterschiedliche Anfrage einen benutzerdefinierten Bezeichner und die zugehörige Anfrage – als vordefinierten Typ oder ausformuliert in Anfragesyntax – speichert. Aus dieser Liste kann eine Anfrage ausgewählt werden, um zu einem späteren Zeitpunkt nochmals dieselbe oder eine abgewandelte Suche durchzuführen.

Um das optimale Suchergebnis schrittweise zu erhalten, muß das System dem Benutzer die Möglichkeit geben, erneute Suchanfragen auf Basis des vorhergehenden Ergebnisses zu stellen. Dadurch wird mit jedem Schritt die Treffermenge weiter eingeschränkt, bis der Benutzer das gewünschte Ergebnis erhält.

Das Ergebnis einer Suchanfrage muß entweder als eine Liste mit Treffern oder eventuell als ein auf Trefferknoten reduzierter Inhaltsbaum angezeigt werden. Die Darstellung als Baum hat den Vorteil, daß der Benutzer die Dokumentgliederung erkennen kann. Außerdem kann die Verteilung der Treffer über die gesamte Hilfe deutlich gemacht werden, indem akkumulierte Trefferanzahlen aus den jeweiligen Unterbäumen der Knoten angezeigt werden. Bei der Auswahl eines Baumknotens durch den Benutzer springt die Fließtext-Ansicht des Hilfe-Dokuments an die zugehörige Dokument-Position (wie auch in Abschnitt 4.2.2 „Inhaltsverzeichnis“ auf Seite 25 beschrieben). Die Information jedes Suchergebnisses, besteht aus dem Titel der zugehörigen Trefferstelle. Zusätzlich wird bei der Darstellung in einer Liste noch eine Einstufung der Treffer nach verschiedenen Kriterien vorgenommen, die die Relevanz eines Treffers bezüglich der Anfrage bestimmen. In der Fließtext-Ansicht der Hilfe werden die Trefferstellen optisch auffällig markiert. Grundsätzlich muß das System neben der direkten Auswahl von Einträgen, eine Navigation zum vorhergehenden bzw. nachfolgenden Suchtreffer anbieten.

## 4.3 Zusätzliche Ansichten

In den folgenden Abschnitten werden die Anforderungen an zusätzliche Ansichten formuliert, die sich nicht als spezialisierte Form aus einem Hilfe-Dokument ergeben. Vielmehr handelt es sich bei diesen Ansichten um die Darstellung von Metadaten des Hilfesystems wie beispielsweise den Lesezeichen oder der Betrachtungs-Geschichte.

### 4.3.1 Lesezeichen

Das System muß die Funktion von Lesezeichen bieten. Mit deren Hilfe kann der Anwender wichtige Positionen in der Hilfe zur späteren Wiederanzeige abspeichern. Lesezeichen werden hierarchisch sortiert in einem Baum verwaltet. Die Hierarchie ist durch den Benutzer selbst definierbar, so daß dieser die Lesezeichen nach seinen Vorgaben sortiert ablegen kann. Der Beschreibungstext eines Lesezeichens wird bei der Neuanlage automatisch aus dem umgebenden Titel der Dokumentposition abgeleitet und läßt sich durch den Benutzer beliebig abändern.

Eine Änderung des Beschreibungstextes muß auch zu einem späteren Zeitpunkt jederzeit möglich sein.

Bei der Auswahl eines Lesezeichens aus dem Baum durch den Benutzer wird das zugehörige Dokument zur Anzeige gebracht bzw. springt die Fließtext-Ansicht des Hilfe-Dokuments an die zugehörige Dokument-Position (wie auch in Abschnitt 4.2.2 „Inhaltsverzeichnis“ auf Seite 25 beschrieben).

Lesezeichen müssen zum einen privat abgelegt werden können, so daß sie nur für den anlegenden Benutzer sichtbar sind. Zum anderen sollten öffentliche Lesezeichen im System existieren. Sie werden über das Netzwerk anderen Benutzer zugänglich gemacht und sind für all diese sichtbar.

### 4.3.2 Betrachtungs-Geschichte

Dokumentpositionen, die der Benutzer während seiner Sitzungen in der Hilfe anwählt und betrachtet, müssen in einem Art Logbuch festgehalten werden. Dies ermöglicht dem Benutzer, zu bereits betrachteten Positionen zurückzuspringen, um beispielsweise einen anderen Pfad durch den Informationsraum des Hypertext-Systems zu verfolgen. Auf diese Weise unterstützt die Geschichte die Zugriffsart der Exploration. Bei der Geschichte handelt es sich um eine Liste von Einträgen, bestehend aus einem Titel und dem Datum, an dem die betroffene Stelle angewählt wurde. Der Titel wird automatisch aus Informationen des Dokuments generiert und läßt sich nicht durch den Benutzer abändern.

Bei der Auswahl eines Geschichtseintrags aus der Liste durch den Benutzer wird das zugehörige Dokument zur Anzeige gebracht bzw. springt die Fließtext-Ansicht des Hilfe-Dokuments an die zugehörige Dokument-Position. Grundsätzlich muß das System dem Benutzer die schrittweise Bewegung vorwärts und rückwärts durch die Betrachtungs-Geschichte ermöglichen.

## 4.4 Weitere Funktionalität

Weitere Anforderungen an die Funktionalität, die nicht direkt mit Ansichten von Dokumenten oder Metadaten in Beziehung stehen, werden in den folgenden Abschnitten dieses Kapitels erläutert.

### 4.4.1 Automatische Zusammenführung mehrerer Hilfe-Dokumente und -Pakete

Üblicherweise besteht die Hilfe-Information, die ein Benutzer benötigt, nicht aus einem umfassenden Hilfepaket, in dem sämtliche Informationen zusammengefaßt sind. Stattdessen bekommt der Benutzer mehrere Hilfepakete zu den jeweiligen zu dokumentierenden Systemen bzw. deren Teilsystemen. Speziell wenn es gilt, ständig auf neue Versionen von Entwickler-Dokumenten Zugriff zu haben, wie dies im SESAMDoc-Projekt der Fall ist, existiert zwangsläufig eine Menge von Hilfe-Dokumenten oder kleinen Paketen. Es ist nicht lohnenswert, bei jeder Änderung an solchen Dokumenten ein neues umfassendes Hilfepaket zu erstellen.

In jedem Fall möchte der Benutzer aber in einer ausgewählten Menge oder allen vorhandenen Hilfe-Dokumenten bzw. -Paketen Suchanfragen im Volltext oder im Index durchführen. Deshalb müssen die Indizes und – falls vorhanden – Volltextindizes automatisch zusammengeführt werden können. Dies gewährleistet, daß eine Suche nach einem Indexbegriff bzw. im Volltext alle relevanten Stellen in den Dokumenten bzw. Paketen finden kann. Auch die strukturierte Suche mit Hilfe von vordefinierten Suchanfrage-Typen muß in einem Durchlauf über mehrere Dokumente und Pakete möglich sein.

### 4.4.2 Programmierschnittstelle

Eine Programmierschnittstelle zum Hilfesystem wird zur Kommunikation zwischen zu dokumentierendem Anwendungsprogramm und dem Hilfe-Browser benötigt. Nach [Herczeg 1994] ist die Möglichkeit zur kontextsensitiven Hilfe eine notwendige Bedingung für die Eigenschaft eines Hilfesystems. Die Programmierschnittstelle ermöglicht kontextsensitive Hilfe, indem das Anwendungsprogramm den Hilfe-Browser so ansteuert, daß die zum Kontext gehörende Hilfe-Information angezeigt wird. Darüberhinaus könnte der Browser, sofern er als visuelle Komponente realisiert ist, direkt in die graphische Oberfläche eines Anwendungsprogrammes eingebettet werden. Allerdings sollte eine solche Einbettung vermieden werden und stattdessen das Hilfesystem unabhängig vom Anwendungsprogramm in eigenen Fenstern der graphischen Benutzungsoberfläche realisiert werden [Herczeg 1994]. Auf diese Weise bleibt dem Benutzer die Organisation der Anzeigebereiche von Anwendung und Hilfesystem überlassen.

### 4.4.3 Internationalisierung

Die Beschriftungen der graphischen Benutzungsoberfläche des Hilfesystems müssen in verschiedenen Sprachen angezeigt werden können. Der Benutzer wählt hierzu aus den zur Verfügung stehenden Übersetzungen eine von ihm bevorzugte Sprache aus.

Ebenso wie in der graphischen Oberfläche des Browsers muß Unterstützung für Hilfe-Dokumente in verschiedenen Sprachen angeboten werden. Eine einfache Auswahl einer vom Benutzer favorisierten Sprache ist sicherlich nicht ausreichend. Existiert ein Dokument in der gewählten Sprache nicht, so ist es äußerst unerwünscht, wenn in diesem Fall nichts oder eine willkürliche Standardsprache angezeigt wird. Ein flexibler Mechanismus zur Festlegung von Sprachprioritäten sollte dem Benutzer angeboten werden. Damit lassen sich verschiedene bevorzugte Sprachen spezifizieren und das Hilfesystem hat die Möglichkeit, auf Dokumente in anderen Übersetzungen auszuweichen.

Problematisch ist im Zusammenhang mit verschiedensprachigen Hilfe-Dokumenten auch die Auswahl einer bestimmten Übersetzung unter dem Gesichtspunkt ihrer Aktualität. So ist es durchaus möglich, daß ein Dokument in einer bestimmten Sprache bereits aktualisiert wurde, während Übersetzungen davon noch einer älteren Version entsprechen. Der Benutzer sollte selbst entscheiden können, welches Dokument er nun bevorzugt. Beispielsweise ist es für einen Entwickler wichtig, die neueste Version zur Verfügung zu haben, auch wenn diese nicht in einer seiner bevorzugten Sprachen existiert. Für den Anwender eines Software-Systems wird es dagegen vorteilhafter sein, er erhält in einem solchen Fall eine ältere Dokumentversion oder unter Umständen die Meldung, daß keine Dokumentation vorliegt. Information in einer nicht bevorzugten Sprache sollte einem solchen Anwender nicht zugemutet werden, da dies eher Verwirrung stiftet als hilfreich ist. Die benutzerdefinierte Entscheidung für bestimmte Dokumentversionen sollte ebenfalls wie im vorangehenden Absatz beschrieben mit Hilfe von definierbaren Sprachprioritäten oder Benutzerabfragen bei Mehrdeutigkeiten erfolgen.

Die Verwaltung verschiedener Dokumentversionen und Übersetzungen kann der Hilfe-Browser kaum übernehmen. Hierfür sind spezielle Dokumentrepositorien eher geeignet. Jedoch sollte der Hilfe-Browser die Funktion der Schnittstelle zwischen einem solchen Repositorium und dem Benutzer übernehmen, um eine nahtlose Integration in das Hilfesystem herzustellen.

# 5 SESAMDoc-Browser

SESAMDoc-Browser ist der Name für den Online-Hilfe-Browser, der im Rahmen dieser Studienarbeit konzipiert und realisiert wurde.

Die Spezifikation des SESAMDoc-Browser Online-Hilfesystems basiert auf den Anforderungen wie sie im vorhergehenden Kapitel beschrieben werden. Sie enthält den Großteil der Anforderungen und spezifiziert diese detailliert in Bezug auf den konkreten SESAMDoc-Browser. Das Spezifikationsdokument setzt sich zusammen aus allgemeinen Grundlagen, der Beschreibung des Programmstarts und der graphischen Benutzungsoberfläche. Weiterhin enthalten ist eine Aufstellung aller Aktionen, die der Benutzer über Tastatur oder Maus auslösen kann, um den Zustand des Systems zur Laufzeit zu ändern. Die Programmierschnittstelle zur Realisierung kontextsensitiver Hilfe wurde in der Spezifikation noch nicht näher festgelegt, da dies ohne Informationen aus dem später erstellten Entwurf nicht sinnvoll ist. Aus diesem Grund befindet sich an dieser Stelle der Spezifikation ein Verweis auf den Entwurf. Weitere Teile der Spezifikation enthalten nichtfunktionale Anforderungen, mögliche Erweiterungen des spezifizierten Systems sowie zwei Anhänge. Ein Anhang besteht aus einem Glossar, das sowohl allgemeine Begriffe aus der Spezifikation definiert, als auch eine Zuordnung von verwendeten deutschen Bezeichnungen für graphische Oberflächenkomponenten zu ihren englischen Pendants und den Klassennamen aus dem Swing-Toolkit vornimmt. Der zweite Anhang spezifiziert die drei XML-Datenformate der System-Metadaten: Lesezeichen, Betrachtungs-Geschichte und Suchanfragen-Geschichte.

## 5.1 Unterschiede zu Anforderungen

Im Hinblick auf die spätere Umsetzbarkeit des spezifizierten Browsers, wurden bestimmte, ausgewählte Details aus den Anforderungen nicht in die Spezifikation mitaufgenommen. Im Folgenden sollen diese Unterschiede kurz beschrieben und begründet werden.

- Hyperlinks sind ausschließlich einseitig und einfach spezifiziert. Damit sind sämtliche Verweisarten, wie sie in einem DocBook-Dokument vorkommen können, abgedeckt. Zweiseitige bzw. mehrfache Hyperlinks wurden nicht mit in die Spezifikation aufgenommen, da sie nicht Teil von DocBook sind. Derartige Verweise benötigen zusätzliche Standards, die deren Form und Verhalten festlegen. Für XML-Dokumente wird an einem solchen Standard mit Namen XLink (XML Linking Language) gearbeitet. Da er noch nicht verabschiedet wurde, existieren auch noch wenige und vor allem frei verwendbare Implementierungen für XLink. Eine solche Implementierung wird aber für eine Umsetzung zweiseitiger und mehrfacher Verweise benötigt. Weiterhin sind benutzerdefinierte Hyperlinks nicht Teil der Spezifikation. Auch diese setzen zur Umsetzung beispielsweise XLink voraus. Außerdem wird zusätzlich ein Standard zur Adressierung von Positionen in schreibgeschützten Zieldokumenten benötigt. Unter dem Namen XPointer (XML Pointer Language) wird ebenfalls für XML-Dokumente an einem solchen Standard gearbeitet, der noch nicht verabschiedet ist.
- Die Darstellung der *Suchanfrage-Ergebnisse im Inhaltsbaum* durch akkumulierte Suchtrefferanzahlen an den Baumknoten mußte verworfen werden. Diese Funktionalität wäre vermutlich nur unter großem Aufwand realisierbar gewesen. Die akkumulierten Trefferanzahlen müßten für jeden Knoten rekursiv berechnet und danach zwischengespeichert werden. Dies würde eine komplette Dokumentbaum-Traversierung nach jeder Suchanfrage bedeuten. Zur Repräsentation eines XML-Dokuments siehe Abschnitt 2.3 „XML (Extensible Markup Language)“ auf Seite 3 und die dortige Abbildung 2–1. Die Berechnung und Zwischenspeicherung kann bei Verwendung von XPath-Suchausdrücken, die von

einer Maschine abgearbeitet werden, nicht während der Traversierung zur Suchzeit durchgeführt werden. Außerdem ist die Wahl eines optimalen Speicherungsort für die berechneten Werte notwendig. Eine eigenständige Datenstruktur in Form eines Wörterbuchs, das zu jedem Knoten den Wert speichert, ist für diesen Zweck zu speicheraufwendig. Die Speicherung direkt in jedem Dokumentknoten führt automatisch zu einer Abhängigkeit vom verwendeten XML-Parser, der den Dokumentbaum aufbaut, da benutzerspezifische Daten in DOM-Knoten nicht in diesem Standard spezifiziert und somit implementierungsabhängig sind.

- Auf eine *Relevanz-Einstufung* der Suchanfrage-Ergebnisse wurde verzichtet. Die Auswahl bzw. der Entwurf und Implementierung eines dazu notwendigen Algorithmus, der zudem auf strukturierten Daten und nicht nur auf reinem Volltext funktioniert, hätte den Umfang der Studienarbeit bei weitem überschritten.
- *Lesezeichen* sind aussschließlich in privater Form spezifiziert. Öffentliche Lesezeichen, die für alle Benutzer über das Netzwerk sichtbar gemacht werden existieren somit nicht. Durch die detaillierte Spezifikation des Speicherungsformats von Lesezeichen ist jedoch jederzeit zumindest ein manueller Datenaustausch zwischen Benutzern möglich.
- Auf eine *automatische Zusammenführung* mehrere Hilfepakete wurde verzichtet. Eine solche automatische Zusammenführung ist für DocBook-Dokumente nicht vorgesehen. Ohne Beschränkung der Allgemeinheit kann jedoch beispielsweise der Dokument-Autor mit geringem Aufwand mehrere Dokumente manuell zu einem größeren Dokument zusammenfassen. Dies ermöglicht das DocBook-Element „set“, das zur Zusammenfassung einer Menge von Büchern und Artikeln dient.
- Die *Internationalisierung* der Hilfe-Dokumente wurde nicht in die Spezifikation mitaufgenommen. Während Java mit dem Swing-Toolkit für damit erstellte graphische Benutzungsoberflächen direkte Unterstützung für verschiedene Sprachen anbietet, so ist dies für DocBook-Dokumente nicht gegeben. Auf den ersten Blick enthält DocBook zwar zu jedem Element ein „lang“-Attribut zur Angabe, welcher Sprache der Element-Inhalt genügt, dies ist aber nicht für den hier gewünschten Anwendungszweck geeignet. Das „lang“-Attribut dient Stylesheets zur länderspezifisch korrekten Behandlung des Element-Inhalts. Beispielsweise wird dadurch bei der Aufzählung von Autoren in einem Literaturverzeichniseintrag die Trennzeichenkette zwischen vorletztem und letztem Autor festgelegt. Für einen amerikanisch-englischen Verzeichniseintrag wird „, and “ verwendet, für einen deutschen Eintrag dagegen „ und “. Die Unterscheidung identischer Dokumente in verschiedenen Übersetzungen müßte somit von einer Art Hilfepaket-Verwaltung oder einem Repositorium, aus dem Dokumente geladen werden, übernommen werden.

## 5.2 Ausgewählte Punkte der Spezifikation

### 5.2.1 Standardsprache der graphischen Oberfläche

Als Standardsprache für die Beschriftung der graphischen Oberflächenkomponenten wurde Englisch gewählt. Der Grund dafür ist die geplante Veröffentlichung des SESAMDoc-Browsers unter der GNU General Public License (GNU GPL). Dadurch wird weltweit jeder Person die Weiterentwicklung der freien Software ermöglicht. Eine solche „verteilte“ Weiterentwicklung findet meist mit Hilfe des Internet als Kommunikationsmedium statt. Die Standardsprache des Internet ist Englisch. Somit sind durch die Wahl dieser Sprache die notwendigen Voraussetzungen geschaffen.

### 5.2.2 Auslieferungsform des Browsers

Die Auslieferung des Browsers erfolgt in einer JAR-Datei (Java Archive) und gewährleistet auf diese Weise den einfachen Transport, da es sich nur um eine einzige Datei handelt. Außerdem lassen sich Java-Programme in dieser Form am einfachsten vom Benutzer ausführen, da das Archiv alle benötigten Informationen zum Programmstart enthält. Unter Microsoft Windows lassen sich beispielsweise Programme in JAR-Dateien direkt durch einen simplen doppelten Mausklick auf das Sinnbild einer solchen Datei ausführen.

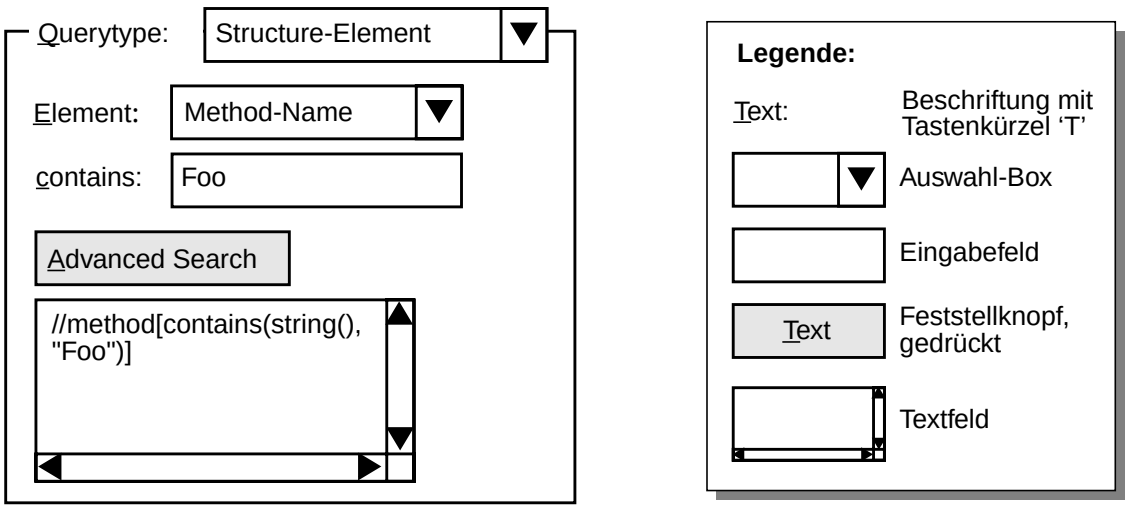
### 5.2.3 Benutzerspezifische Metadaten des Systems

Metadaten des Systems, d.h. Lesezeichen, Betrachtungs-Geschichte und Suchanfragen-Geschichte werden im Verzeichnis „`sesamdoc`“ innerhalb des Benutzerverzeichnisses abgelegt. Auf diese Weise sind die Metadaten automatisch dem erzeugenden Benutzer zugeordnet. In der Abteilung Software Engineering sind alle Arbeitsstationen in einer Netzwerk-Verwaltungsgruppe zusammengefaßt, die unter anderem die automatische, globale Verfügbarkeit des Benutzerverzeichnisses über das Netzwerk sicherstellt. Dadurch stehen dem Benutzer des SESAMDoc-Browsers jederzeit die Metadaten automatisch zur Verfügung.

### 5.2.4 Strukturierte Suche

Für die Durchführung strukturierter Suchanfragen sieht die Spezifikation zwei vordefinierte Suchanfragetypen vor. Ein Anfragetyp dient der Suche nach einem bestimmten *Struktur-Element*, das zusätzlich einen spezifischen Textinhalt haben soll. Die Dialoggestaltung der graphischen Benutzungsoberfläche hierzu ist in Abbildung 5–1 zu sehen. Das Struktur-Element

**Abbildung 5–1: Dialog zum Suchanfragetyp „Struktur-Element“**



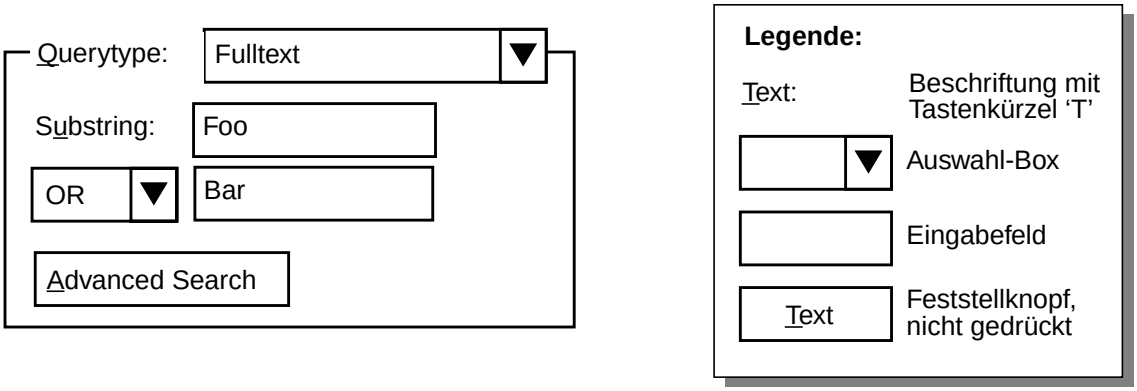

bestimmt der Benutzer über eine Auswahl-Box. Die Auswahlliste stellt folgende Einträge zur Verfügung: Title, Function-Name, Method-Name, Class-Name, Parameter-Name, Parameter-Type, Return-Type, Command. Auf diese Weise kann die Suche beispielsweise auf Überschriften (Title) beschränkt werden. Weiterhin ist es möglich nach Funktionen bzw. Methoden innerhalb eines Hilfe-Dokuments zu suchen, die anhand ihrer Signatur einen spezifischen Datentyp zurückliefern (Return-Type). Eine solche Funktionalität ist überaus nützlich für die Dokumentation von Software-Schnittstellen. Sie ermöglicht die gezielte Suche nach Information aus dem Programm-Quelltext wie sie beispielsweise zur Wartung von Software benötigt wird.

Zusammenhänge zwischen den Schnittstellen eines dokumentierten Systems lassen sich auf diese Weise schneller erkennen. Falls das Struktur-Element, nach dem gesucht werden soll, zusätzlich einen bestimmten Textinhalt aufweisen soll, kann der Benutzer diesen Text im Eingabefeld unterhalb der Auswahl-Box festlegen.

Unterhalb der spezifischen Eingabekomponenten eines jeden vordefinierten Suchanfragetyps läßt sich über den Feststellknopf „Advanced Search“ die Suchanfrage-Syntax einblenden. Die Anfrage wird vom System automatisch bei Veränderungen an den oberen Eingabekomponenten gebildet. Dem fortgeschrittenen Benutzer erlaubt deren Einblendung, beliebige Anfragen direkt in der XPath-Syntax zu formulieren und auszuführen. Dabei kann jederzeit eine vom System generierte Anfrage als Basis für eigene Veränderungen dienen.

Der zweite in der Spezifikation vorgesehene Anfragetyp dient der Suche im *Volltext* eines Dokuments. Die Dialoggestaltung hierzu ist in Abbildung 5–2 zu sehen. Der Benutzer hat die

**Abbildung 5–2: Dialog zum Suchanfragetyp „Volltext“**




Möglichkeit zwei Zeichenketten in den Texteingabefeldern vorzugeben. Zusätzlich legt er fest wie die Verknüpfung der beiden Zeichenketten sein soll. Ist gewünscht, daß bei der Suche beide Zeichenketten gleichzeitig innerhalb eines DocBook-Elements enthalten sein müssen, kann in der Auswahl-Box die boolesche Und-Verknüpfung (AND) ausgewählt werden. Ist das unabhängige Vorkommen einer der beiden Begriffe oder das gemeinsame Auftreten ausreichend, so wird die Oder-Verknüpfung (OR) ausgewählt. Die Suche nach nur einem Begriff wird durchgeführt, indem die Oder-Verknüpfung gewählt und das zweite Texteingabefeld leer gelassen wird. In Abbildung 5–2 ist der Feststellknopf „Advanced Search“ nicht gedrückt, so daß die vom System generierte Suchanfrage-Syntax nicht sichtbar ist.

Vorläufig ist vorgesehen, daß die Groß-/Kleinschreibung bei Suchanfragen signifikant ist. Der Grund dafür ist die unzureichende Unterstützung für Zeichenkettenvergleiche ohne Berücksichtigung der Groß-/Kleinschreibung in der derzeit vorliegenden XPath Version 1.0. Die XPath-Spezifikation [Clark et al. 1999] kennt nur eine Funktion zur Abbildung einzelner Zeichen auf selbst definierte, andere Zeichen. Hiermit lassen sich vor einem Zeichenkettenvergleich alle Zeichen in eine gemeinsame Großschreibung umwandeln. Dies ist aber nicht für alle Sprachen, in denen DocBook-Dokumente vorkommen können, ausreichend. Zu dieser Problematik existiert in der XPath-Spezifikation bereits ein Hinweis auf Unterstützung für die Umwandlung von Groß-/Kleinschreibung in einer zukünftigen XPath-Version.

### 5.2.5 Fließtext-Ansicht

Die Fließtext-Ansicht repräsentiert den eigentlichen textuellen Inhalt eines Hilfe-Dokuments. Zu diesem Zweck wird den verschiedenen DocBook-Elementen im Dokument jeweils eine bestimmte Textformatierung zugewiesen. Grundsätzlich wird zwischen Block- und Inline-Formatierung unterschieden. Wird ein Element als Block formatiert bedeutet dies, daß der gesamte Elementinhalt als Absatz dargestellt wird. Eine Inline-Formatierung eines Elements geschieht im laufenden Text, also beispielsweise in einem Textteil innerhalb einer Blockformatierung.

Anhand der Referenzanleitung zu DocBook-Elementen in [Walsh et al. 1999] habe ich eine Unterteilung in Elemente, die als Block und Elemente, die inline zu formatieren sind, vorgenommen. Allerdings können einige Elemente je nach Kontext unterschiedlich formatiert werden. Zur Auflösung der Konflikte wurde dem Großteil dieser Elemente eine Blockformatierung zugewiesen. Der Grund dafür ist, daß der formatierte Text dann in einem eigenen Absatz dargestellt wird und somit lesbarer bleibt als eine Aneinanderreihung von inline-formatierten Zeichenketten. Darüberhinaus existieren in DocBook Elemente, deren Inhalte als Metadaten einzustufen sind. Ihnen wird keine Formatierung zugewiesen, so daß sie nicht im Fließtext erscheinen. Im Gegensatz zu einer Druckausgabe von Dokumenten, bei der solche Informationen z.B. auf den ersten Seiten eines Buches erscheinen, sind in einem Browser spezielle Dialoge zur Anzeige von Metadaten besser geeignet. So könnte beispielsweise der Inhalt des „BookInfo“-Elements in einem Dialog zu Dokumenteigenschaften angezeigt werden.

Natürlich ist eine bloße Unterscheidung in Block- und Inline-Formatierung nicht ausreichend, um ein lesbares Textbild zu erzeugen. Darum werden beide Mengen weiter unterteilt. Aus der Menge der Block-formatierten werden Elemente, die Überschriften auszeichnen, in einer serifenlosen Schriftart und größerer Schriftgröße gesetzt. Elemente, deren Inhalt bedeutungsvollen Leerraum (Folge von Leerzeichen, Tabulatoren und Zeilenumbrüchen) enthält – in DocBook „linespecific“ genannt – bekommen eine nicht-proportionale Schriftart zugewiesen. Auf diese Weise bleiben Programm-Quelltexte oder Abbildungen von Textbildschirmen lesbar. Die Menge der inline-formatierten Elemente wird wie folgt weiter aufgeteilt: Einige Auszeichnungen beispielsweise für Abkürzungen (Abbrev), Hervorhebung (Emphasis) oder Glossarbegriffe (GlossTerm) bekommen einen kursiven Schriftschnitt zugewiesen, um sie im Textfluß kenntlich zu machen. Wie bei den Block-formatierten existieren auch in der Menge der inline-formatierten Elemente Kandidaten, die in nicht-proportionaler Schriftart gesetzt werden. Dazu gehören unter anderem Klassen-, Schnittstellen-, Funktions-, Variablennamen sowie Befehle (Command), Bildschirmausgabe (ComputerOutput) oder E-Mail-Adressen (Email).

Betrachtet man die Vielzahl der existierenden Elemente in DocBook, von denen es ungefähr 360 verschiedene gibt, und nimmt man zusätzlich deren Attribute, die sie darüberhinaus noch entsprechend parametrisieren, hinzu, so fällt auf, daß die oben beschriebene Unterteilung in insgesamt 6 disjunkte Teilmengen im Vergleich dazu recht simpel anmutet. In der Tat ist die umfassende Erstellung von Formatieranweisungen für DocBook-Dokumente eine äußerst schwierige, langwierige und fehleranfällige Prozedur. Dies zeigt sich unter anderem an den inoffiziellen DSSSL- und XSL-Stylesheets zu DocBook. Auch nach jahrelanger Aktualisierung und Weiterentwicklung der sehr umfangreichen Formatiervorlagen finden sich immer wieder Kleinigkeiten, die fehlerhaft oder noch unvollständig sind. Als Zugeständnis an die Komplexität mußte ich mich mit der oben beschriebenen Vorgehensweise begnügen. Auch wenn dadurch möglicherweise wichtige Dinge wie Tabellen oder Anmerkungen zu Abbildungen noch nicht behandelt werden können und somit nicht darstellbar sind. Eine Erweiterung in dieser Richtung sollte jedoch jederzeit möglich sein.

### 5.2.6 Hyperlinks in DocBook

Im folgenden möchte ich einen Überblick über die verschiedenen Hyperlink-Arten von DocBook geben sowie auf deren Möglichkeiten und Einsatzgebiete eingehen. Eine ausführlichere Beschreibung zu Form und Verhalten befindet sich in der Spezifikation selbst. Wie bereits weiter oben beschrieben sind die Verweise allesamt einseitig und können auf exakt ein Ziel verweisen. Sofern nicht anders vermerkt, ist das Verweisziel ein Element, das ein – innerhalb des Dokuments eindeutiges – ID-Attribut trägt.

- FirstTerm ist ein Element, welches das erste Auftreten eines Begriffes im Text auszeichnet. Die Zielposition kann eine verwandte Stelle im Dokument sein, beispielsweise ein Glossareintrag.
- FootnoteRef ist ein Element zur Realisierung eines Hyperlinks zu einer Fußnote eines Dokuments. Ein solcher Verweis wird eingesetzt, wenn von anderen Stellen im Dokument auf eine vorhandene Fußnote verwiesen werden soll.
- GlossSee / GlossSeeAlso sind Elemente zur Realisierung eines Hyperlinks innerhalb eines Glossars im Dokument. GlossSee wird verwendet, um den Leser darauf hinzuweisen, daß sich die Begriffsdefinition unter dem verwiesenen Ziel befindet. GlossSeeAlso weist den Leser dagegen darauf hin, daß unter dem Verweisziel zusätzliche Informationen zu finden sind.
- GlossTerm ist ein Element zur Auszeichnung eines Glossarbegriffes sowohl bei Auftreten des Begriffs im Dokumenttext als auch im Glossareintrag selbst. Üblicherweise wird von einem mit GlossTerm ausgezeichneten Begriff im Text zu einem Glossareintrag verwiesen, um beim Lesen des Textes zur Begriffsdefinition springen zu können. Denkbar ist aber auch der umgekehrte Weg, mit dem man zu einem Glossareintrag die Stelle im Dokument finden kann, an welcher der Begriff verwendet wird, ähnlich der Funktionsweise eines Index.
- Link ist ein allgemein verwendbares Element zur Realisierung eines Hyperlinks innerhalb eines Dokuments.
- OLink ist ein Element zur Realisierung eines Hyperlinks, der als Ziel ein externes DocBook-Dokument referenziert. Die Spezifikation sieht für die Aktivierung eines solchen Links vor, daß das Zieldokument in einer neuen Ansicht geöffnet und angezeigt wird.
- ULink ist ein Element zur Realisierung eines Hyperlinks, der mit Hilfe eines URL auf ein beliebiges externes Ziel verweist. Sofern es portabel umsetzbar ist, soll bei Aktivierung des Links der Netscape Browser über den remote-Mechanismus beauftragt werden, das Ziel anzuzeigen. Prinzipiell sollte es auch möglich sein, mit der HTML-Text-Komponente aus dem Swing-Toolkit von Java das Ziel anzuzeigen, da es sich doch in vielen Fällen um HTML-Dokumente handelt, auf die verwiesen wird.
- XRef ist ein Element zur Realisierung von Querverweisen innerhalb eines Dokuments.

### 5.2.7 Nichtfunktionale Anforderungen

Der SESAMDoc-Browser wird im Rahmen einer Studienarbeit erstellt. Somit steht nur beschränkt Zeit zur Verfügung. Aus diesem Grund können an das System keine allzu hohen nichtfunktionale Anforderungen in Bezug auf Leistung, Sicherheit und Verfügbarkeit gestellt werden. Wichtige und in jedem Fall einzuhaltende Anforderungen sind jedoch die Plattformunabhängigkeit und die Wartbarkeit des Systems. In den folgenden Absätzen werden die einzelnen Anforderungen näher erläutert.

Entsprechend den „Richtlinien für das Projekt SESAMDoc“ soll der Browser plattformunabhängig sein. Aus diesem Grund verlangt die Spezifikation ebenfalls vom System die Lauffähigkeit unter den Betriebssystemen Solaris, Linux und Windows (NT/9x).

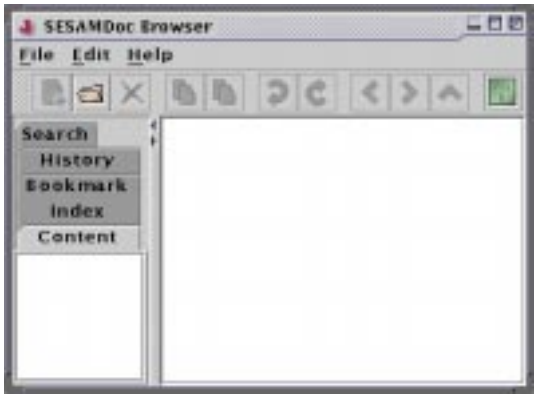
Das Ziel der Entwicklung ist ein wartbares Produkt. Dies bedeutet, daß Studenten oder auch andere projektfremde Personen den Browser im Quelltext verändern, anpassen und erweitern können müssen. Grundlage hierzu sind neben einem modularen Entwurf die Befolgung der folgenden von Sun herausgegebenen Richtlinien zur Implementierung in Java: “Code Conventions for the Java Programming Language”, “How to Write Doc Comments for Javadoc” und “Requirements for Writing Java API Specification”.

Der Browser soll DocBook-XML-Dokumente mit einer Größe von mindestens 1000 Elementen und 500 Attributen sinnvoll anzeigen. Dies entspricht je nach Anwendungsfall bei eingerücktem und auf 80 Spalten umgebrochenem XML-Format einer Dateigröße von ungefähr 100 Kilobytes. Eine für den Druck formatierte Ansicht eines Dokuments dieser Größe bewegt sich ungefähr zwischen 20 und 30 DIN-A4 Seiten. Genauere Aussagen hierzu sind äußerst schwierig, da die Zusammenhänge zwischen Element-/Attribut-Anzahl, Dateigröße und Seitenumfang in Druckform sehr stark variieren und vom Dokumentinhalt sowie dem Grad der Auszeichnung abhängen.

Es gibt keine besonderen Anforderungen an Sicherheit und Verfügbarkeit. Sobald dem System nicht mehr genügend Hauptspeicher zur Verfügung steht, bricht das Programm mit einer Ausnahme ab. Ein solcher Fall kann eintreten, falls bereits vor dem Programmstart zu wenig Speicher vorhanden war und deshalb das System gar nicht erst starten kann. Außerdem tritt ein solcher Fall ein, sobald versucht wird, Hilfe-Dokumente zu öffnen, die für die zur Verfügung stehenden Ressourcen zu groß sind.

## 5.3 Produkt aus Sicht des Benutzers

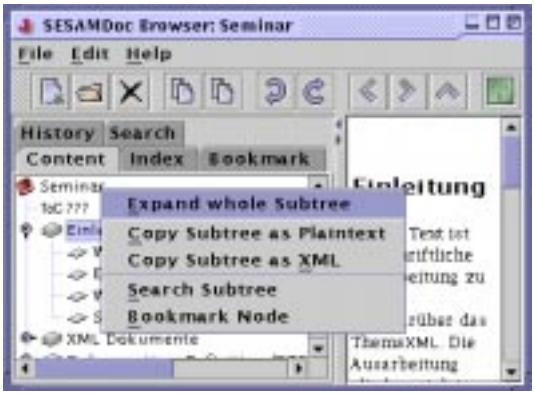
**Abbildung 5–3: Leeres Hauptfenster des SESAMDoc-Browser**




Nach dem Start des Programms, beispielsweise über den Befehl „**java –jar browser.jar**“, erscheint ein leeres Hauptfenster wie in der nebenstehenden Abbildung 5–3 gezeigt.

Über das File-Menü oder die darunterliegende Werkzeugleiste wird ein Standarddialog zum Öffnen eines Hilfe-Dokuments aufgerufen. Nach erfolgreicher Öffnung eines Dokuments wird dieses im bestehenden, leeren Hauptfenster angezeigt. Abbildung 5–4 auf Seite 38 zeigt das Fenster nach dem Öffnen eines Dokuments, wobei der Benutzer bereits einige Gliederungspunkte im Baum des Inhaltsverzeichnisses expandiert hat.

**Abbildung 5–5: Inhaltsverzeichnis mit Kontextmenü im SESAMDoc-Browser**




Über ein Kontextmenü kann für einen Baumknoten im Inhaltsverzeichnis (Content) eine bestimmte Aktion ausgelöst werden. Zu den Aktionen, die den gesamten Unterbaum des Kontextknotens betreffen, gehören das vollständig rekursive Expandieren, das Kopieren des Inhalts in die Zwischenablage als Text oder im XML-Format sowie die Veranlassung der Suche beschränkt auf den Unterbaum. Außerdem kann über das Kontextmenü ein Lesezeichen für den Kontextknoten gesetzt werden.

**Abbildung 5–4: Hauptfenster des SESAMDoc-Browser mit geöffnetem Hilfe-Dokument**

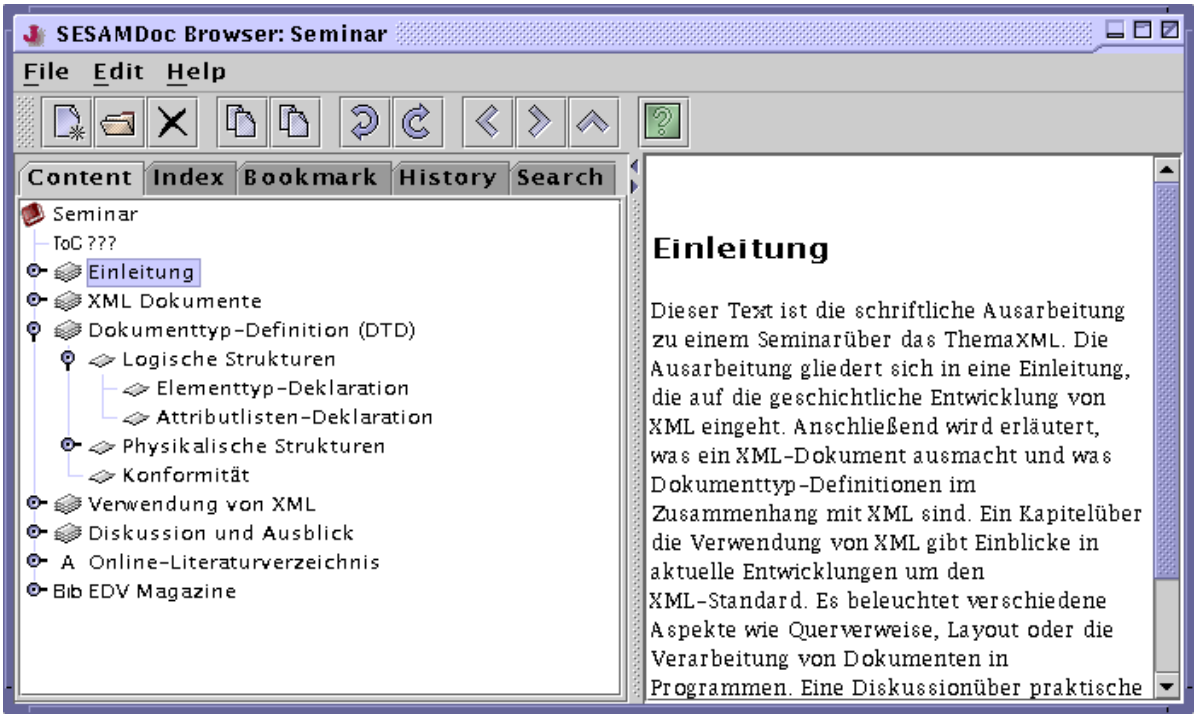
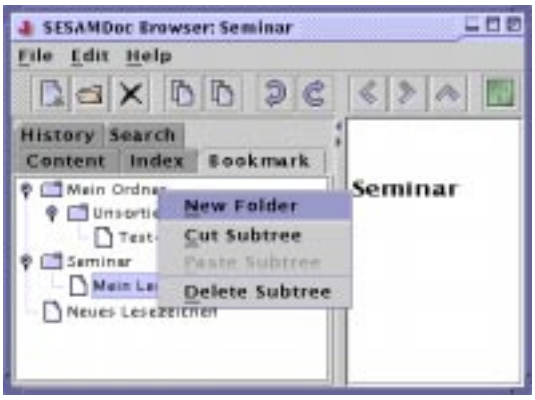
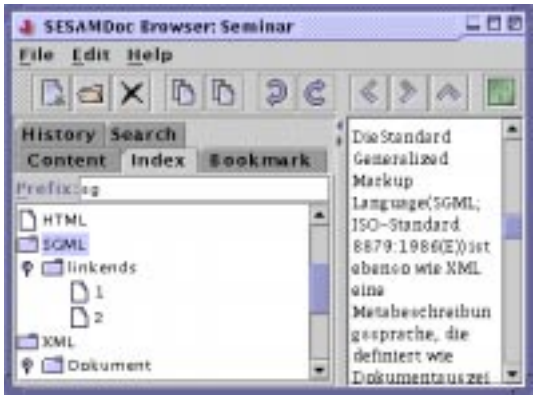



**Abbildung 5–6: Lesezeichen mit Kontextmenü im SESAMDoc-Browser**




Wechselt man die Karteireiter-Ansicht in der linken Fensterhälfte zu den Lesezeichen (Bookmark), lassen sich Lesezeichen auswählen, um die gespeicherte Dokumentposition anzuzeigen. Weiterhin können die Lesezeichen benutzerdefiniert, hierarchisch sortiert und verwaltet werden. Dies geschieht wiederum über ein Kontextmenü wie in Abbildung 5–6 gezeigt.

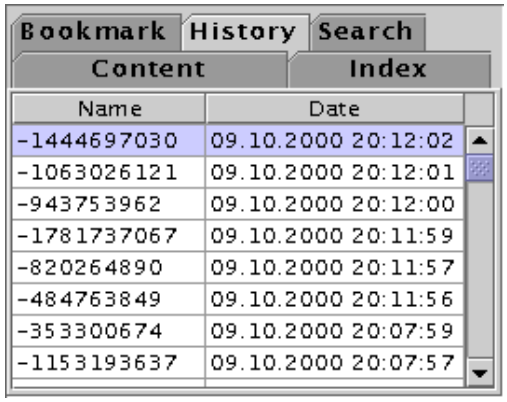
**Abbildung 5–7: Index mit Suchfunktion im SESAMDoc-Browser**




Auf der Karteikarte für die Index-Ansicht werden die Indexeinträge alphabetisch sortiert in einem Baum dargestellt. Der Baum dient der hierarchischen Verschachtelung primärer, sekundärer und tertiärer Begriffe. In der nebenstehenden Abbildung 5–7 wurde der Indexbegriff „SGML“ mit Hilfe der inkrementellen Suche nach dem Begriffspräfix „sg“ gefunden und markiert. Unterhalb des markierten Knotens ist der Unterbaum, der die Definitionsziele im Dokumenttext enthält, zu sehen. Die Ziele sind hier einfach durchnummeriert. Markiert der Benutzer einen solchen Ziel-

knoten springt die Fließtext-Ansicht in der rechten Fensterhälfte zu der Stelle im Hilfe-Text, an welcher der Indexbegriff definiert wurde.

**Abbildung 5–8: Betrachtungs-Geschichte im SESAMDoc-Browser**




Die Betrachtungs-Geschichte (History) wird in einer zweispaltigen Tabelle, wie in Abbildung 5–8 dargestellt, angezeigt. Die Einträge sind nach dem Zugriffsdatum in der rechten Tabellenspalte sortiert. Am oberen Tabellenende befinden sich die neuesten Einträge, d.h. die zuletzt besuchten Dokumentpositionen. Die rechte Tabellenspalte zeigt das Zugriffsdatum des Eintrags.

Die im folgenden beschriebenen Funktionalitäten konnten aus Zeitgründen nicht mehr implementiert werden. Eine Begründung und nähere Erläuterungen hierzu befinden sich in Abschnitt 7.2 „Geplante, noch nicht realisierte Funktionen“ auf Seite 65.

Die linke Tabellenspalte sollte einen automatisch generierten Namen für die Dokumentposition statt den abgebildeten Zufallszahlen zeigen. Außerdem sollten die Geschichts-Einträge ständig aktualisiert werden, während der Benutzer durch Hilfe-Dokumente navigiert. Falls eine Position bereits früher einmal besucht wurde und deshalb in der Geschichte bereits eingetragen ist, so wird das Zugriffsdatum aktualisiert und der Eintrag wandert aufgrund seines neueren Datums nach oben. Bei Auswahl eines Eintrags durch den Benutzer ist vorgesehen, daß die gespeicherte Dokumentposition zur Anzeige gebracht wird.

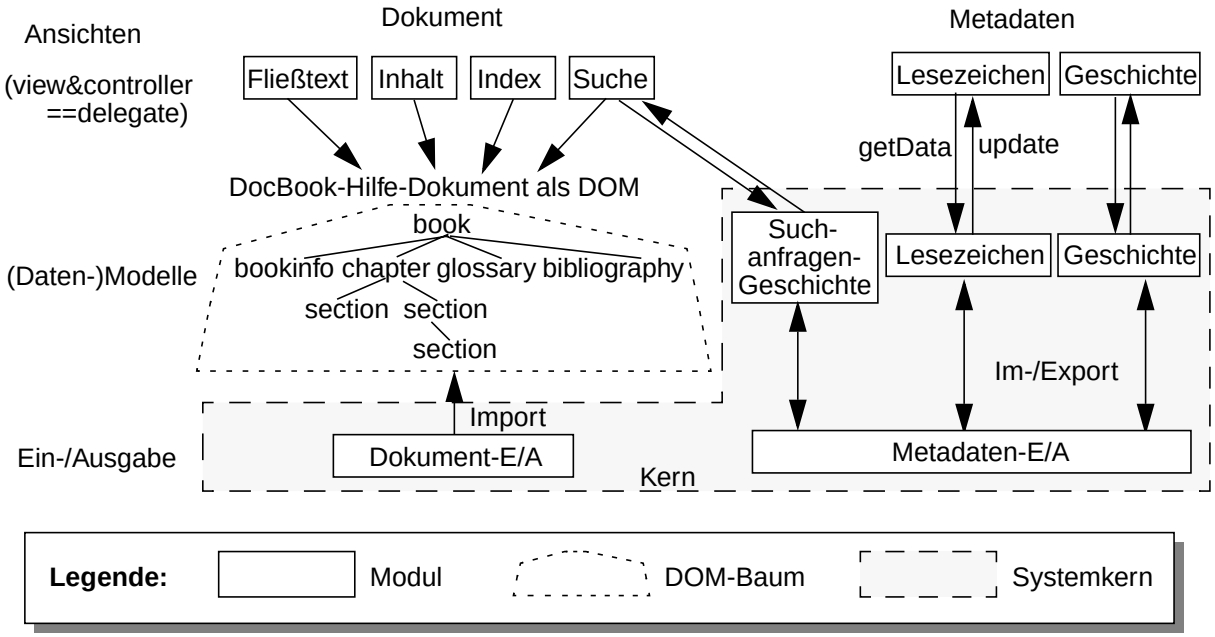
# 6 Systemaufbau

Das Thema dieses Kapitels ist der Entwurf des SESAMDoc-Browsers. Zunächst wird im ersten Abschnitt ein grober Überblick über das gesamte System gegeben. Da hier das System auf einen Blick ersichtlich wird, kann dieser Abschnitt jederzeit als Referenz zu Rate gezogen werden, um globale Zusammenhänge innerhalb des Systems nachzuvollziehen. Anschließend werden die Systemteile Systemkern, Datenmodelle und Ein-/Ausgabemodule näher beschrieben. Der Aufbau der graphischen Benutzungsoberfläche wird separat behandelt, da der Entwurf die Oberfläche soweit wie möglich von oberflächenunabhängigen Teilen trennt. Zum Verständnis der dort verwendeten Begrifflichkeiten geht der Behandlung eine kurze Einführung in die eingesetzten Entwurfsmuster voraus. Zuletzt wird das Modul für die Fließtext-Ansicht im SESAMDoc-Browser aufgrund seiner Komplexität separat behandelt.

## 6.1 Übersicht

Das System ist modular aufgebaut. Die zu verwendende Implementierungs-Programmiersprache Java bietet hierzu ein Paketkonzept. Darüberhinaus ist das System weitgehend aus wiederverwendbaren Komponenten aufgebaut. Dies wird durch sogenannte JavaBeans ermöglicht. Detailliertere Angaben zu JavaBeans folgen in Abschnitt 6.5.3 „JavaBeans“ auf Seite 50. Abbildung 6–1 zeigt einen Überblick über den Systementwurf.

**Abbildung 6–1: Systementwurf**




Dem Entwurf liegt das MVC-Entwurfsmuster (model view controller) zugrunde. Die Aufgaben der drei Teile des Musters sind wie folgt definiert: Das Modell enthält Daten und besitzt einen bestimmten Zustand, über dessen Änderung es interessierte Objekte unterrichtet. Die Ansicht (view) hat die Aufgabe, die Daten aus dem Modell anzuzeigen und greift hierzu ausschließlich lesend auf das Modell zu. Die Steuerung (controller) verarbeitet Ereignisse, die z.B. durch Benutzerinteraktion in der Ansicht erzeugt werden, und verändert daraufhin Modelldaten.

Das zentrale Datenmodell des Systems ist das DocBook-Hilfe-Dokument. Es wird durch die Baum-Datenstruktur des DOM (Document Object Model) repräsentiert. Für die Metadaten des Systems wie Lesezeichen, Geschichte und Suchanfragen-Geschichte existieren spezialisierte abstrakte Datentypen. In Analogie zu DOM, welches ausschließlich durch seine Schnittstellenbeschreibung spezifiziert ist, sind auch die Metadatenmodelle durch ihre Schnittstellen beschrieben. Dadurch bleibt der Zugriff auf die Daten unabhängig von einer bestimmten Implementierung.

Den verschiedenen Modellen sind entsprechende Ansichten zugeordnet. Entgegen dem reinen MVC-Prinzip findet keine Trennung von Ansicht und Steuerung statt. In Konsistenz mit dem Swing-Toolkit sind diese beiden Entwurfsmuster-Teile stellvertretend zu einem sogenannten „delegate“ zusammengefaßt. Swing unternimmt diese Zusammenfassung hauptsächlich aus Performance-Gründen. Da die Teile des Musters ausschließlich über Ereignisse miteinander kommunizieren spart der Zusammenschluß von Ansicht und Steuerung einen Großteil dieses Kommunikationsaufwands.

Ein Hilfe-Dokument wird in verschiedenen spezialisierten Ansichten dargestellt. Dazu gehören der formatierte Fließtext, das Inhaltsverzeichnis, der Index sowie ein Dialog zur Ausführung von Suchanfragen, dessen Suchergebnisse eine Teilmenge des Dokuments darstellen. All diese Ansichten greifen ausschließlich lesend auf das Datenmodell zu, um ihre Darstellungen aufzubauen (angedeutet durch einseitige Pfeile in Abbildung 6–1). Die Hilfe-Dokumente können sich während einer Browser-Sitzung nicht verändern, so daß eine Benachrichtigung an die Ansichten bei Zustandsänderungen im Modell nicht erforderlich ist.

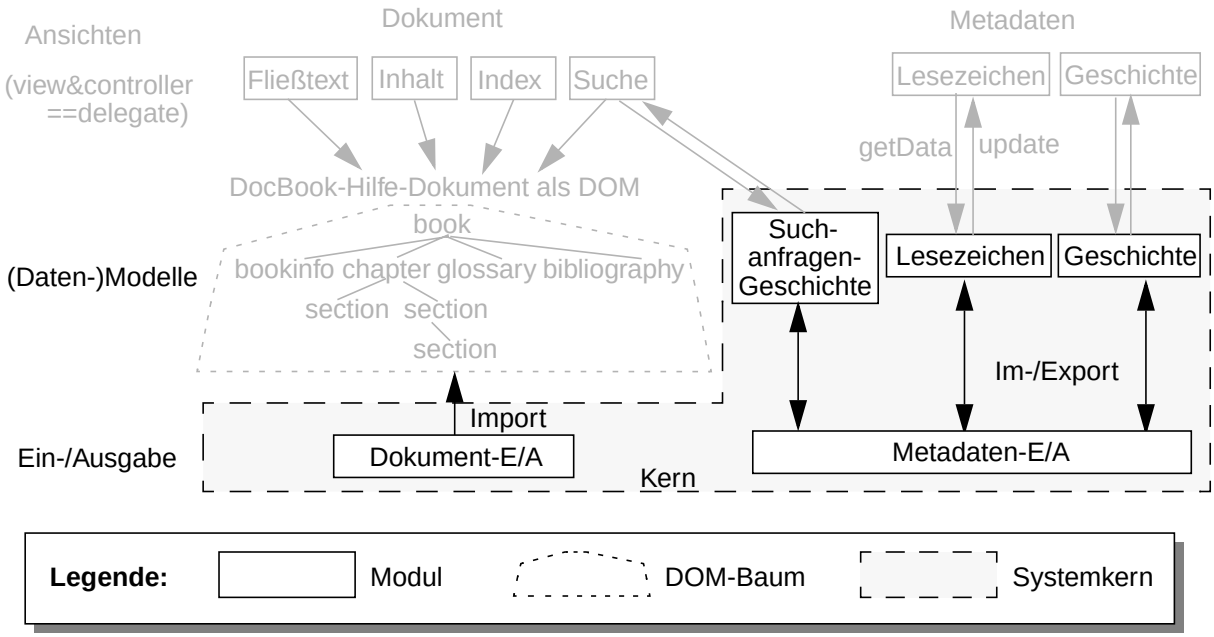
Die Metadaten des Systems werden ebenfalls in verschiedenen Ansichten dargestellt. Dazu gehören die Lesezeichen, die Geschichte sowie die Suchanfragen-Geschichte. Letztere teilt sich den Suchanfrage-Dialog mit der im vorangehenden Absatz beschriebenen Suchergebnis-Ansicht. Zum Aufbau der Darstellungen greifen die Metadaten-Ansichten lesend auf ihre Datenmodelle zu (angedeutet durch die Pfeile mit der Beschriftung „getData“ in Abbildung 6–1). Die Modelle können jederzeit ihren Zustand ändern, indem der Benutzer beispielsweise die Beschriftung eines Lesezeichens bzw. deren Sortierung ändert oder durch Blättern im Dokument indirekt einen Geschichtseintrag erzeugt. Damit die Ansichten ihre Darstellung in solchen Fällen aktualisieren können, werden sie von ihrem Modell benachrichtigt sobald sich dessen interner Zustand ändert (angedeutet durch die Pfeile mit der Beschriftung „update“ in Abbildung 6–1).

Die oben aufgeführten Datenmodelle werden über Ein-/Ausgabe-Module eingelesen bzw. dauerhaft auf ein Speichermedium zurückgeschrieben. Letztendlich wird für das Einlesen eines Hilfe-Dokuments ein XML-Parser verwendet, der die DOM-Repräsentation im Speicher aufbaut. Da auch die Metadaten im XML-Format vorliegen, werden auch hier XML-Parser verwendet, die allerdings statt dem DOM die spezialisierten abstrakten Datentypen für Lesezeichen usw. aufbauen. Außerdem müssen die Metadaten persistent gespeichert werden können. Zu diesem Zweck werden sogenannte XML-Serialisierer eingesetzt, die quasi den Parse-Vorgang umkehren und aus den Datenmodellen wieder lesbare XML-Dokumente generieren.

## 6.2 Systemkern

Der Systemkern ist der zentrale Teil der Browser-Anwendung. Er weist keine Abhängigkeiten von anderen Programmteilen auf und ist unabhängig von der graphischen Oberfläche. Der Kern enthält programmglobale Datenstrukturen, die er zusammen mit verschiedenen Diensten anderen interessierten Programmteilen zur Verfügung stellt. Zu den Datenstrukturen gehören

**Abbildung 6–2: Systemkern**




die abstrakten Datentypen für Lesezeichen, Geschichte und Suchanfragen-Geschichte. Außerdem hält der Kern Objekte aus den Ein-/Ausgabe-Modulen für den Import von Hilfe-Dokumenten sowie den Im- und Export der Metadaten vor. Abbildung 6–2 zeigt noch einmal den Systementwurf in der Übersicht, wobei alles außer dem Systemkern grau dargestellt ist.

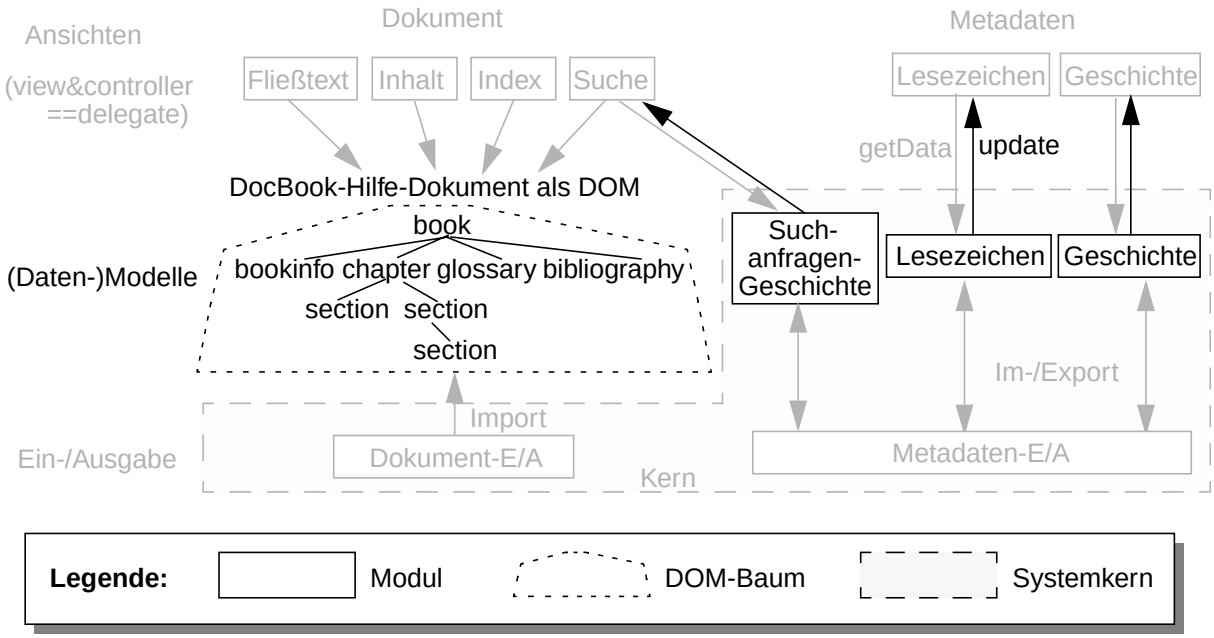
Um die Zusammenhänge zwischen der Anwendung, dem Systemkern und der graphischen Benutzungsoberfläche zu verdeutlichen, folgt ein kurzer Überblick über die Vorgänge beim Programmstart. Zunächst wird ein Objekt der Browser-Anwendung einmalig instanziert. Dieses Anwendungsobjekt generiert wiederum den Systemkern. Anschließend erzeugt das Anwendungsobjekt eine anwendungsspezifische Instanz der graphischen Oberfläche, ist aber ansonsten unabhängig von dieser. Zum Abschluß des Programmstarts tritt die Anwendung in die Ereignisverarbeitungs-Schleife und läuft fortan gesteuert durch Ereignisse aus der graphischen Oberfläche.

Der Zugriff auf den Systemkern erfolgt an beliebiger Stelle über eine statische Klassenmethode des Anwendungsobjekts. Die Methode liefert eine Referenz auf den Kern zurück. Diese Art des Zugriffs ist sehr einfach zu vollziehen. Im Gegenzug entstehen dadurch allerdings stillschweigend Abhängigkeiten von der Existenz jener Anwendungsklasse mit ihrer statischen Methode und zwar überall dort, wo durch Zugriffe Gebrauch von dieser Methode gemacht wird. Eine schönere Lösung wäre die Trennung von Schnittstelle und Implementierung und der ausschließliche Zugriff auf den Kern über eine Referenz dieser Schnittstelle. Dies erfordert einen erheblichen Aufwand, da für den Zugriff die Referenz auf den Kern an allen benötigten Programmstellen bekannt gemacht werden muß. Zwangsläufig führt dies zum Durchreichen der Referenz durch verschiedenste Programmteile, selbst wenn vermittelnde Teile die Referenz gar nicht selbst benötigen. Mit der Verwendung einer Klassenmethode bin ich einen Kompromiß gegen die aufwendige, saubere Lösung zugunsten einer effektiv benutzbaren eingegangen. Dies scheint mir insofern gerechtfertigt, als die oben beschriebenen Abhängigkeiten an nur wenigen Programmstellen zum Vorschein treten.

## 6.3 Datenmodelle

Die im System verwendeten Datenmodelle sind ausschließlich spezifiziert durch ihre Schnittstellen. Auf diese Weise bleiben sie unabhängig von ihrer Implementierung. Abbildung 6–3 zeigt noch einmal den Systementwurf in der Übersicht, wobei alles außer den Datenmodellen grau dargestellt ist.

**Abbildung 6–3: Datenmodelle**




Das Datenmodell für ein *Hilfe-Dokument* realisiert einen Kontainer, der die DOM-Repräsentation sowie weitere benötigte Daten enthält und Dienste zur Verfügung stellt. Eine andere Lösungsalternative bestünde darin, die DOM-Schnittstelle durch Ableiten zu erweitern und die DOM-Implementierung ebenfalls so zu erweitern, daß die neue Schnittstelle vollständig implementiert wird. Dieser Ansatz wurde verworfen, da er weitaus komplexer ist und zu viele Abhängigkeiten von der konkret eingesetzten DOM-Implementierung mit sich bringt. Dies steht gewissermaßen im Widerspruch zur Trennung von Schnittstelle und Implementierung.

Neben dem Zugriff auf die DOM-Repräsentation eines Dokuments bietet dieses Modell den Zugriff auf den Index des Dokuments sowie den Dokument-Titel. Außerdem enthält der Kontainer eine Zeichenkette, welche die Ressource, aus der das Dokument eingelesen wurde, eindeutig unter den geöffneten Dokumenten identifiziert. Darüber ist es möglich, festzustellen, ob ein bestimmtes Dokument bereits geöffnet wurde. Weiterhin stellt das Modell eine Methode zur Verfügung mit deren Hilfe anhand eines ID-Attributwertes der zugehörige DOM-Knoten aufgefunden werden kann. Dies wird zur Realisierung von Indexeinträgen und Hyperlinks benötigt, um das Sprungziel in Form eines DOM-Knotens zu ermitteln.

Im Falle der *Lesezeichen* spezifiziert die Modellschnittstelle Methoden für den Zugriff auf eine Baum-Datenstruktur. Objekte, die an einer Änderung des Zustands interessiert sind, können sich als Ereignisempfänger registrieren. Um die Umorganisation der hierarchischen Lesezeichensortierung durch Ausschneiden und Einfügen von Teilbäumen zu ermöglichen, existieren Methoden für den Zugriff auf eine Modell-interne Zwischenablage, die einen Teilbaum aufnimmt. Außerdem wird eine Methode definiert, mit deren Hilfe neu angelegte Lesezeichen hinzugefügt werden. Die Baumknoten der Datenstruktur sind in einer eigenen Schnittstelle für

Lesezeichen-Einträge definiert. Da Knoten sowohl Kontainerordner (innere Knoten) als auch Lesezeichen (Blattknoten) sein können, findet eine Unterscheidung mit Hilfe einer Typ-Eigenschaft statt. Jeder Knoten enthält als Information einen Beschriftungstext. Knoten vom Typ Lesezeichen enthalten zusätzlich die Zielposition, die sich aus einem Dokument-Identifizierer und der Position innerhalb dieses Dokuments zusammensetzt.

Die Schnittstelle der *Geschichte* spezifiziert Methoden für den Zugriff auf eine Liste von Geschichtseinträgen. Analog zum Lesezeichen-Modell können sich auch hier interessierte Objekte als Ereignisempfänger registrieren, um Modell-Zustandsänderungen zu erfahren. Weiterhin wird eine Methode zum Hinzufügen neuer Geschichtseinträge spezifiziert. Die Einträge der Liste sind in einer eigenen Schnittstelle definiert. Sie enthalten als Information einen Beschriftungstext, die Zielposition, zusammengesetzt aus Dokument-Identifizierer und Position, und schließlich das Datum, zu dem die Zielposition zuletzt besucht wurde.

Die Schnittstelle der *Suchanfragen-Geschichte* unterscheidet sich nur unwesentlich von derjenigen der Geschichte. Jedoch enthalten die Einträge in diesem Fall neben einer Beschriftung die Information für die Suchanfrage.

## 6.4 Ein-/Ausgabe-Module

Ebenso wie die Datenmodelle des Systems sind die Ein-/Ausgabe-Module ausschließlich durch ihre Schnittstellen definiert und somit unabhängig von einer konkreten Implementierung. Dies ist hier insofern wichtig, als verschiedene Arten der Ein-/Ausgabe vorgesehen werden müssen. Ein Hilfe-Dokument läßt sich nicht nur aus einer einfachen Datei in einem Dateisystem importieren. Vielmehr ist es sinnvoll die Dokumente in Paketen oder von einem zentralen Dokumentrepositorium verwalten zu lassen und sie direkt von dort aus zu importieren. Derartige Möglichkeiten eröffnen sich bei feststehender Schnittstelle durch eine spätere Veränderung der Implementierung. Auf diese Weise ist die problemlose Erweiterung des Systems jederzeit gegeben.

Die Schnittstelle des *Dokument*-Import-Moduls stellt sich recht kurz dar. Sie besteht im wesentlichen aus nur einer Methode zur Ausführung des Import-Vorgangs, die bei Erfolg ein Datenmodell eines Hilfe-Dokuments zurückliefert. Diese Methode ist in zwei überladenen Versionen vorgesehen. Zum einen kann ein Eingabestrom[1] als Import-Quelle angegeben werden. Zum anderen genügt ein URL als Quellenangabe.

Für jeden einzelnen Typ von *Metadaten*, d.h. für Lesezeichen, Geschichte und Suchanfragen-Geschichte, definiert die Modulschnittstelle jeweils eine Methode für den Import sowie eine Methode für den Export. Die Import-Methoden liefern das jeweilige Datenmodell zurück, während die Export-Methoden ein entsprechendes Modell als Parameter entgegennehmen.

### 6.4.1 Automatische Erzeugung des Index

DocBook sieht vor, daß Indexbegriffe innerhalb des Dokumenttextes als solche ausgezeichnet werden. Anschließend läßt sich in einem unabhängigen Vorgang automatisch ein Index aus den Begriffsauszeichnungen erstellen [Walsh et al. 1999]. Bei Verwendung einer Import-Modul-Implementierung, die beispielsweise auf einem Hilfepaket oder einem zentralen Dokumentrepositorium basiert, kann ein vorgenerierter Index aus den Verwaltungssystemen übernommen werden. Die derzeitige Implementierung des Import-Moduls für Hilfe-Dokumente erlaubt das Öffnen von DocBook-Dokumenten in Form von einfachen Dateien. In diesem Fall übernimmt

1. Sequentielle Ein-/Ausgabeoperationen werden in Java grundsätzlich auf sogenannte „streams“ zurückgeführt.

das Import-Modul die Indexgenerierung. Der Generierungsvorgang läßt sich in drei Teilaufgaben untergliedern:

1. *Identifikation* aller Indexbegriffsdefinitionen innerhalb des Dokumenttextes
2. *Zusammentragen* der Indexbegriffsdefinitionen in einer geeigneten Datenstruktur
3. *Erzeugung* eines gültigen Index-DocBook-Dokuments

Noch vor Beginn der ersten Teilaufgabe wird eine neue DOM-Datenstruktur instanziert, die den Index als Dokument aufnehmen soll. Das Dokument besitzt als Wurzelknoten das DocBook-Element „index“ und wird im weiteren Verlauf mit einem gültigen Inhaltsmodell für DocBook-Indizes aufgefüllt werden. Beispiel 6–1 zeigt die Ausschnitte eines fiktiven DocBook-Dokuments, die Indexbegriffsdefinitionen enthalten.

**Beispiel 6–1: Begriffsdefinitionen für einen Index eines DocBook-Dokuments**

```
<indexterm id="idx.sesam">
  <primary>SESAM</primary>
</indexterm>
...
<indexterm>
  <primary>SESAM</primary>
  <secondary>SESAMDoc</secondary>
</indexterm>
...
<indexterm id="index.browser">
  <primary>SESAM</primary>
  <secondary>SESAMDoc</secondary>
  <tertiary>Browser<tertiary>
</indexterm>
...
<indexterm>
  <primary>foo</primary>
</indexterm>
...
<indexterm>
  <primary>bar</primary>
</indexterm>
...
<indexterm> <!-- weitere Begriffsdefinition für SESAM -->
  <primary>SESAM</primary>
</indexterm>
```

Zu den Definitionen in Beispiel 6–1 würde man in einem gedruckten Buch beispielsweise eine Index-Form wie im folgenden Beispiel 6–2 erwarten. Die angegebenen Seitenzahlen sind natürlich frei erfunden und hängen von der Position der Begriffsdefinitionen im Dokument und vom Seitenumbruch ab.

**Beispiel 6–2: Mögliche Darstellungsform für die Indexdefinitionen aus Beispiel 6–1**

bar ..... 12
foo ..... 31
SESAM ..... 1, 23
SESAMDoc ..... 3
Browser ..... 5

**Identifikation der Indexbegriffsdefinitionen**

1. Zur Vorbereitung wird über sämtliche „indexterm“-Elemente innerhalb des bereits geparsten und als DOM vorliegenden Hilfe-Dokuments iteriert. Jedes dieser Elemente bekommt, sofern es noch kein ID-Attribut besitzt, ein solches Attribut mit einem eindeutigen Wert zugewiesen. Dies ist notwendig, um später aus einem Indexeintrag zur entsprechenden Dokumentposition, an welcher der Begriff definiert wurde, springen zu können. Das Sprungziel ist durch ein ID-Attribut eindeutig innerhalb des Dokuments identifiziert. Prinzipiell verändert dieser Vorgang zwar das Dokument, was jedoch keine negativen Auswirkungen auf andere Programmteile hat. Außerdem existiert das veränderte Dokument ausschließlich im Speicher und wird nicht wieder zurückgespeichert, so daß die Veränderungen nur temporärer Natur sind. Nach der Identifizierung aller Begriffsdefinitionen aus Beispiel 6–1 würden die Dokumentausschnitte wie im folgenden Beispiel 6–3 aussehen. Tatsächlich wird die Veränderung natürlich innerhalb der DOM-Datenstruktur vorgenommen und nicht in einem XML-Dokument in lesbarer Form.

**Beispiel 6–3: Identifizierte Begriffsdefinitionen aus Beispiel 6–1**

```
<indexterm id="idx.sesam">
  <primary>SESAM</primary>
</indexterm>
<indexterm id="sesamdoc.browser.1">
  <primary>SESAM</primary>
  <secondary>SESAMDoc</secondary>
</indexterm>
<indexterm id="index.browser">
  <primary>SESAM</primary>
  <secondary>SESAMDoc</secondary>
  <tertiary>Browser<tertiary>
</indexterm>
<indexterm id="sesamdoc.browser.2">
  <primary>foo</primary>
</indexterm>
<indexterm id="sesamdoc.browser.3">
  <primary>bar</primary>
</indexterm>
<indexterm id="sesamdoc.browser.4">
  <primary>SESAM</primary>
</indexterm>
```

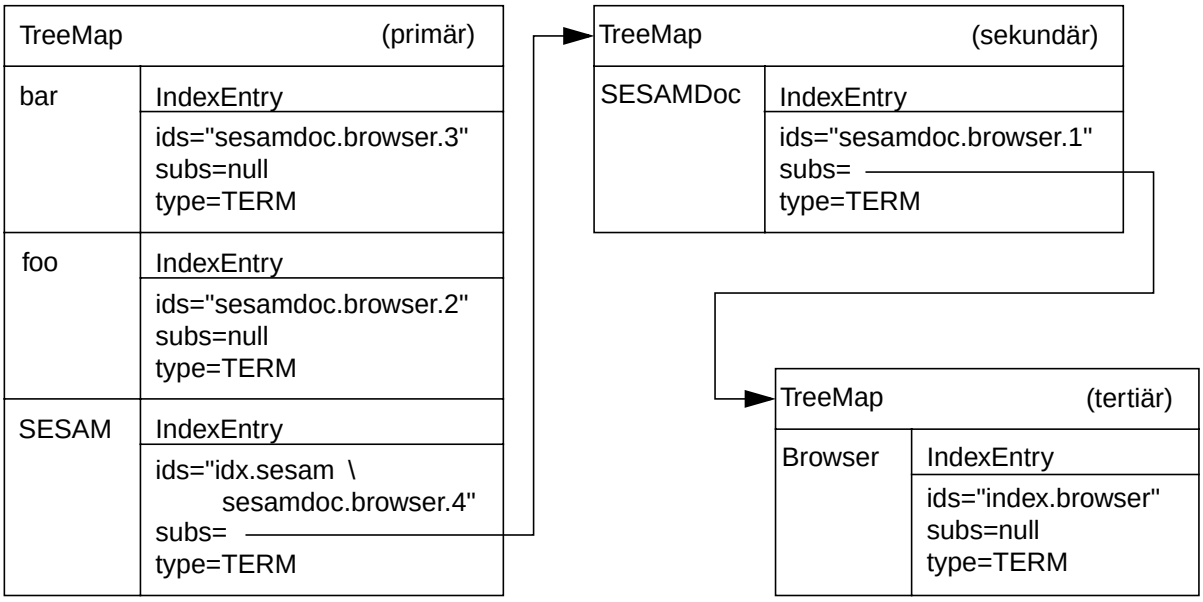
**Zusammentragen der Indexbegriffsdefinitionen**

2. Anschließend wird der Index Begriff für Begriff zunächst in einer geeigneten Datenstruktur aufgebaut. Für jeden Indexbegriff existiert in der Datenstruktur ein Objekt der Klasse „IndexEntry“. Die Klasse enthält eine Zeichenkette, die durch Leerzeichen getrennt sämtliche ID-Attributwerte der zugehörigen Begriffsdefinitionen aufnimmt. Weiterhin wird der Typ des Begriffs festgehalten. Es kann sich um einen Begriff handeln, für den ein Sprungziel in Form einer Begriffsdefinition im Dokumenttext existiert oder um einen Indexeintrag der Form „Begriff-1 siehe Begriff-2“ bzw. „Begriff-1 siehe auch Begriff-2“. Zur Unterstützung verschachtelter Indexeinträge speichert die Klasse „IndexEntry“ außerdem eine Referenz auf die in der Hierarchie untergeordnete Datenstruktur. DocBook erlaubt mit primären, sekundären und tertiären Indexbegriffen bis zu drei Hierarchiestufen in einem Index.

Nachdem nun die Einträge der Datenstruktur bekannt sind, möchte ich die Besonderheit der Struktur erläutern. Es handelt sich um den abstrakten Datentyp „TreeMap“ aus der JDK-Klassenbibliothek. TreeMap ist eine Wörterbuch-Datenstruktur – auch Hash genannt –, die zu einem Schlüssel mit möglichst geringem Aufwand einen zugeordneten Wert liefert. Hash-Tabellen berechnen aus dem Schlüssel einen Tabellenindex, um an den Wert zu gelangen. Im Gegensatz dazu basiert TreeMap, wie der Name bereits andeutet, auf einem Baum als Wertbehälter. Außerdem stehen die Werte in TreeMap jederzeit in alphabetischer Reihenfolge ihrer zugehörigen Schlüssel zu Verfügung. Der Aufwand für Einfüge- und Ausleseoperationen ist logarithmisch. Die Sortierung nach Schlüsseln ermöglicht nach Aufbau der Datenstruktur ohne weiteren Aufwand die Erstellung eines alphabetisch nach Begriffen sortierten Index.

Der Aufbau der Datenstruktur erfolgt dynamisch während der Iteration über die im Dokument befindlichen Begriffsdefinitionen. Ist ein Begriff noch nicht in der Struktur enthalten, so wird ein neues IndexEntry-Objekt eingefügt. Bei bereits in der Struktur vorhandenen Begriffen wird der ID-Attributwert als Sprungziel der erneuten Definition an die im IndexEntry gespeicherten Werte angehängt. Auf diese Weise ergibt sich für jeden Indexbegriff exakt ein Eintrag in der Datenstruktur. Für die Begriffsdefinitionen aus Beispiel 6–3 wird beispielsweise die Datenstruktur in Beispiel 6–4 aufgebaut. Zur Übersichtlichkeit wird auf eine explizite Darstellung der internen Baumstruktur von TreeMap verzichtet und stattdessen eine einfache Tabelle verwendet, wenngleich dies natürlich nicht der tatsächlichen Struktur entspricht.

**Abbildung 6–4: Aufgebaute Datenstruktur nach der Indexgenerierung**




**Erzeugung eines gültigen Index-DocBook-Dokuments**

3. Nachdem die Index-Datenstruktur vollständig aufgebaut worden ist, wird über all ihre Einträge iteriert und der eigentliche Index in gültiger DocBook-Form erzeugt. Zu diesem Zweck wird für jeden Eintrag das entsprechende DocBook-Element mit evtl. zugehörigen Attributen im Index-Dokument erzeugt und in dessen DOM-Baum eingehängt. Beispiel 6–4 zeigt die lesbare Darstellung des letztendlich aus den Begriffen von Beispiel 6–1 generierten Index.

**Beispiel 6–4: Generierter Index nach Iteration über die aufgebaute Datenstruktur**

```
<indexentry>
  <primaryie linkends="sesamdoc.browser.3">bar</primaryie>
</indexentry>
<indexentry>
  <primaryie linkends="sesamdoc.browser.2">foo</primaryie>
</indexentry>
<indexentry>
  <primaryie linkends="idx.sesam sesamdoc.browser.4">
    SESAM
  </primaryie>
  <secondaryie linkends="sesamdoc.browser.1">
    SESAMDoc
  </secondaryie>
  <tertiaryie linkends="index.browser">Browser</tertiaryie>
</indexentry>
```

**Beurteilung des Algorithmus**

Es stellt sich die berechtigte Frage, ob die Erzeugung des Index auf die eben beschriebene Art eine gute Lösung darstellt. Als Alternative würde sich beispielsweise der Einsatz der XSL-Transformationssprache anbieten. Diese ist eigens für die Erzeugung von XML-Dokumenten aus XML-Dokumenten entworfen worden. Nachteilig wäre an einer solchen Lösung allerdings, daß zunächst die zusätzliche Einarbeitung in die Programmierparadigmen der Regelbasierten XSLT-Sprache notwendig würde. Außerdem benötigte man dann zur Ausführung des Browsers neben dem bereits eingesetzten XML-Parser auch ein XSLT-Transformationswerkzeug für Java. Prinzipiell wäre letzteres ein geringes, lösbares Problem. Ich habe mich aber trotzdem für eine explizite, imperative Implementierung in Java entschieden. Zum einen, weil die erforderliche intensive Auseinandersetzung mit der DOM-Schnittstelle den Umgang mit ihr in anderen Programmteilen durch den erzielten Lerneffekt vereinfacht. Zum anderen, weil sich die Implementierung in Java nahtlos in den restlichen Browser-Quelltext eingliedert.

Es bleibt noch zu erwähnen, daß der Algorithmus in seiner derzeitigen Form nicht umfassend sämtliche, umfangreichen Möglichkeiten von DocBook unterstützt, wenngleich doch ein Großteil funktionsfähig ist. Nicht erkannt werden fehlerhafte Begriffsdefinitionen im Dokument, die sich nach [Walsh et al. 1999] nicht grundlegend durch die XML-DTD, die DocBook definiert, verhindern lassen. Ich bin der Meinung, daß dieser Umstand vernachlässigt werden kann, da die Erzeugung und Überprüfung auf gültige Begriffsdefinitionen bei der Erstellung eines Hilfe-Dokuments vollzogen werden sollte und nicht bei der späteren Darstellung in einem Browser.

## 6.5 Eingesetzte Entwurfsmuster

Um den Entwurf der graphischen Oberfläche erläutern zu können, ist es notwendig vorab eine kurze Einführung zu den dort verwendeten Entwurfsmustern zu geben. Die folgenden Darstellungen sind von der Implementierungssprache Java und dem zugehörigen Oberflächen-Toolkit Swing beeinflußt und folglich in diesem Zusammenhang zu betrachten. Prinzipiell existieren die Muster jedoch unabhängig von einer bestimmten Programmiersprache und Toolkit.

### 6.5.1 Java-Interface (Schnittstelle)

Voraussetzung für die Umsetzung vieler der folgenden Muster in Java ist das Konzept der Schnittstelle, in Java „Interface“ genannt. Es handelt sich dabei neben Klassen um ein Sprach-

konstrukt zur objektorientierten Programmierung. Im Gegensatz zu Klassen in Java erlauben Schnittstellen Mehrfachvererbung. Bei Schnittstellen können keine Mehrdeutigkeiten bei polymorpher Typauflösung entstehen. Dadurch wird die Behandlung sowohl für den Übersetzer und die Laufzeitumgebung als auch für den Programmierer wesentlich vereinfacht. Schnittstellen spezifizieren ausschließlich eine Menge von Konstanten sowie Methoden mit deren Signatur jedoch ohne eine bestimmte Implementierung. Insofern sind Schnittstellen vergleichbar mit abstrakten Klassen beispielsweise in der Sprache C++. In anderen objektorientierten Sprachen wird eine vergleichbare Funktionalität teilweise als Protokoll bezeichnet. Jede definierte Schnittstelle hat einen dazu passenden Typ. Klassen können nun solche Typen mit Leben erfüllen, indem sie Schnittstellen implementieren. Durch eine Implementierung übernimmt die Klasse zusätzlich zu ihrem eigenen Klassentyp auch den Schnittstellentyp. Weiterführende Informationen und eine detailliertere Beschreibung befinden sich beispielsweise in [Friendly et al. 1999], [Niemayer et al. 1997] und [Wilhelms et al. 1999].

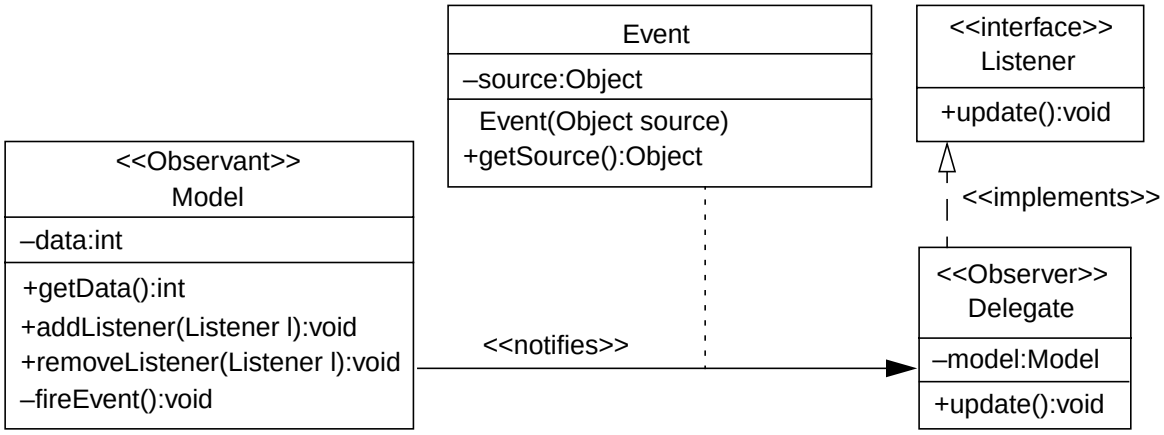
### 6.5.2 Observer-Muster und Kommunikation mit Ereignissen

Die Kommunikation zwischen unabhängigen Objekten kann mit Hilfe sogenannter Ereignisse (*events*) erfolgen. Ereignisse sind selbst Objekte, die Information tragen. Dabei ist die Referenz des Ereignissenders die Minimalinformation, die ein Ereignis tragen muß. Nun stellt sich die Frage, welche Arten von Verbindung zwischen Ereignissender und -empfänger möglich sind und wie diese hergestellt wird. Es kann einerseits höchstens einen Empfänger geben. In diesem Fall spricht man von einem „unicast“-Szenario. Da es in dieser Studienarbeit keine Verwendung findet und auch sonst eher selten eingesetzt wird, wird an dieser Stelle nicht näher darauf eingegangen. Als zweite Verbindungsart kann es andererseits im „multicast“-Szenario beliebig viele Empfänger geben. Die Verbindung geschieht nach dem *Observer*-Entwurfsmuster. Dabei registriert sich ein an einem Ereignis interessiertes Objekt – der Observer bzw. Empfänger – bei einem Objekt, das ein Ereignis vom gewünschten Typ aussendet – der Observant bzw. Sender. Zur Registrierung muß der Empfänger eine spezifische Schnittstelle implementieren. Mit Hilfe dieses Schnittstellentyps registriert er seine eigene Referenz beim Sender. Sobald der Sender ein Ereignis aussenden möchte, verteilt er es an alle Empfänger, die sich zuvor bei ihm registriert haben. Die Verteilung erfolgt durch Aufruf einer Benachrichtigungsmethode des Schnittstellentyps eines jeden Empfängers, wobei das Ereignisobjekt als Parameter übergeben wird. Die graphische Darstellung dieser Zusammenhänge ist mit Hilfe eines statischen Klassendiagramms in Abbildung 6–5 erfolgt. Weiterführende Informationen und eine detailliertere Beschreibung befinden sich beispielsweise in [Friendly et al. 1999], [Morrison 1997], [Niemayer et al. 1997] und [Wilhelms et al. 1999].

### 6.5.3 JavaBeans

Wiederverwendbare Komponenten sind in Java in Form sogenannter JavaBeans spezifiziert. Beans sind nicht beschränkt auf die Menge der graphischen Komponenten. Beliebige wiederverwendbare Programmteile, die keine graphische Repräsentation besitzen müssen, können in Form einer Bean Dienstleistungen zur Verfügung stellen. Die Bean-Spezifikation sieht vor, daß eine Klasse ausschließlich durch Verwendung von Mustern ihre Bean-Eigenschaft erlangt. Es werden also gegenüber einer Klasse keine weiteren Werkzeuge zur Erstellung und Verwendung einer Bean benötigt. Für den Benutzer einer Bean ist nur deren öffentliche Schnittstelle maßgebend – wobei als Schnittstelle hier nicht zwangsläufig ein Java-Interface vorausgesetzt wird. Nach außen hin besteht eine Bean aus Eigenschaften (properties) und Methoden. Eigenschaften sind grundsätzlich nur über Zugriffsmethoden zugänglich. Neben Methoden zum Lesen von Eigenschaftswerten – die sogenannten „getter“ – existieren optional Methoden zum Setzen von Eigenschaftswerten – die sogenannten „setter“. Ihre Spezialbezeichnung verdanken

**Abbildung 6–5: Observer-Entwurfsmuster in UML-Notation**




diese Methoden dem Muster, nach dem ihre Bezeichner aufgebaut sein müssen: Sie enthalten als Präfix entweder „get“ oder „set“ und anschließend den Eigenschaftsnamen, dessen erster Buchstabe groß zu schreiben ist. Beispielsweise lautet die Lese- bzw. Schreibmethode für eine Eigenschaft mit Namen „foo“ vom Typ Integer „`int getFoo()`“ und „`void setFoo(int wert)`“. Darüberhinaus können Eigenschaften gebunden sein (bound property). Bei Wertänderung einer solchen Eigenschaft benachrichtigt die Bean alle registrierten Empfänger mit Hilfe eines Ereignisses. Natürlich ist die JavaBean-Spezifikation wesentlich umfangreicher. An dieser Stelle soll es genügen, nur die im System verwendeten Besonderheiten von Beans darzustellen. Weiterführende Informationen und eine detailliertere Beschreibung befinden sich beispielsweise in [Friendly et al. 1999], [Morrison 1997] und [Wilhelms et al. 1999].

Das folgende Beispiel 6–5 zeigt einen Ausschnitt aus der Programmierschnittstelle der Knopf-JavaBean aus dem Swing-Toolkit. Der Quelltext ist in Pseudo-Java notiert, da kein eigenes Java-Interface für den Knopf existiert und Java keine Trennung von Deklaration und Implementierung kennt. Zu diesem Zweck wurden einfach die Initialisierungen der Klassenvariablen sowie die Implementierungs-Körper der Methoden weggelassen.

**Beispiel 6–5: Schnittstelle einer Knopf-JavaBean in Pseudo-Java**

```
// Paket, in dessen Namensraum die folgende Bean deklariert wird
package javax.swing;

// verwendete Klassen und Schnittstellen aus anderen Paketen
import javax.swing.event.ChangeListener;
import java.awt.event.ActionListener;

// Knopf-JavaBean (ist tatsächlich von anderer Klasse abgeleitet)
public class JButton {

  // Namen für die gebundenen Eigenschaften
  public static final String TEXT_CHANGED_PROPERTY;
  public static final String ICON_CHANGED_PROPERTY;

  // Standard-Konstruktor für JavaBean
  public JButton()

  // gebundene Eigenschaft "text" repräsentiert Knopf-Beschriftung
  public String getText()
```

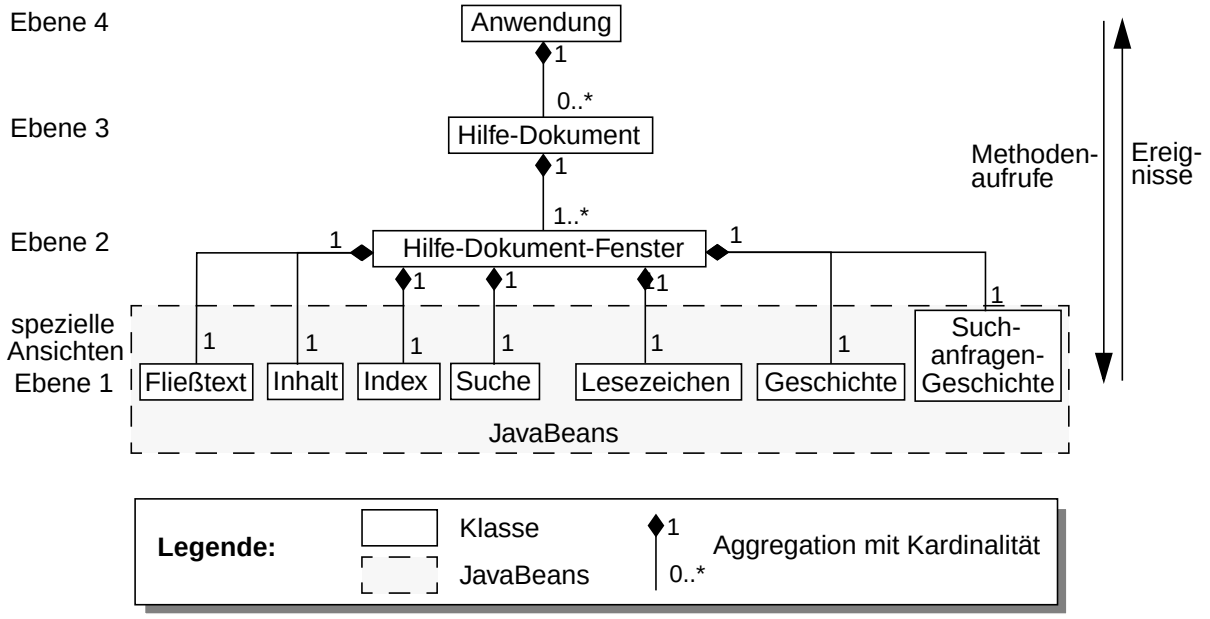
```
    public void setText(String text)

    // gebundene Eigenschaft "icon" repräsentiert Knopf-Sinnbild
    public Icon getIcon()
    public void setIcon(Icon defaultIcon)

    // (De-)Registrierung für gebundene Eigenschaften
    public void addChangeListener(ChangeListener l)
    public void removeChangeListener(ChangeListener l)

    // Eigenschaft "actionCommand",
    // eine Art Knopf-Funktionsbeschreibung, verwendet in ActionEvent
    public String getActionCommand()
    public void setActionCommand(String actionCommand)

    // (De-)Registrierung für Knopf-Aktivierungs-Ereignis
    // (ActionEvent wird z.B. ausgelöst durch Mausklick auf Knopf)
    public void addActionListener(ActionListener l)
    public void removeActionListener(ActionListener l)
}
```

## 6.6 Einteilung und Organisation der graphischen Oberfläche

Die Anforderungen an die graphische Benutzungsoberfläche ergeben sich aus der Spezifikation: Es muß möglich sein, mehrere Hilfe-Dokumente zu öffnen. Von jedem geöffneten Dokument sollen mehrere Darstellungen existieren können. In jeder Darstellung müssen wiederum beliebige, verschiedene spezialisierte Ansichten eines Dokuments sowie Ansichten von Metadaten des Hilfesystems angezeigt werden. Darüber hinaus soll der Entwurf der graphischen Oberfläche ein so flexibles Design aufweisen, daß eine spätere Änderung der erwähnten Anforderungen möglich ist und gleichzeitig möglichst viele Komponenten wiederverwendet werden können.

Zur Erfüllung der Anforderungen wird die Oberfläche in Ebenen aufgeteilt. Eine Übersicht über die bestehenden Ebenen zeigt Abbildung 6–6 auf Seite 53. Das Grundprinzip besteht darin, daß ein Objekt der Ebene n ein Kontainer für beliebig viele Objekte aus der darunterliegendenen Ebene (n–1) ist. Es handelt sich hierbei um die konsequente Weiterführung des Kontainerprinzips wie es im Swing-Toolkit Verwendung findet. Da ein Kontainer seine direkt in ihm enthaltenen Komponenten „kennt“, erfolgt die Kommunikation von Kontainer zu Komponente mit Hilfe von Methodenaufrufen. In der umgekehrten Richtung ist dies nicht möglich. Der Versuch auch hier Methodenaufrufe zu verwenden, würde zu nicht auflösbaren zyklischen Abhängigkeiten zwischen Objekten benachbarter Ebenen führen. Stattdessen werden für diese Kommunikationsrichtung Ereignisse eingesetzt. Auf diese Weise bleiben die Objekte einer Ebene n völlig unabhängig von den Kontainern der darüberliegenden Ebenen n+1, n+2,…, in denen sie enthalten sein können. Mit anderen Worten sind sich die Objekte nicht „bewußt“, was „um sie herum“ geschieht. Ihre Mitteilungen versenden sie in Form von Ereignissen, die von registrierten Interessenten der darüberliegenden Ebene empfangen werden. Kann eine Ebene n ein Ereignis nicht selbst behandeln, da ihr der benötigte Kontext fehlt, wird es an die nächste Ebene (n+1) weitergereicht. Das Durchreichen wiederholt sich unter Umständen solange, bis eine Behandlung möglich ist. Dies ist spätestens in der obersten Ebene 4 der Fall, ansonsten bleibt das Ereignis wirkungslos.

**Abbildung 6–6: Schematisches, statisches Klassendiagramm der graphischen Oberfläche**




Im folgenden sollen die Aufgaben der einzelnen Ebenen vorgestellt werden. Zu beachten ist, daß die Klasse „Hilfe-Dokument“ auf Ebene 3 nicht mit dem Datenmodell für Hilfe-Dokumente aus Abschnitt 6.3 „Datenmodelle“ auf Seite 44 zu verwechseln ist. Möglicherweise ist die Namenswahl etwas unglücklich. Anhand der Modulaufteilung ist jedoch eine strenge Unterscheidung möglich, da die Datenmodelle in anderen Paketen definiert werden als die Teile der graphischen Oberfläche.

In der obersten *Ebene 4* existiert genau ein Objekt, das die Anwendung in Bezug auf die graphische Oberfläche repräsentiert. Das Objekt verwaltet eine Menge von geöffneten Hilfe-Dokumenten – inklusive des Benutzerhandbuchs, das selbst auch ein Hilfe-Dokument ist – und ist zuständig für das Öffnen neuer Dokumente. Auch wenn das Auswerten von Kommandozeilenparameter nicht unbedingt zur Oberfläche hinzuzurechnen ist, so ist dieses Anwendungs-Objekt trotzdem auch dafür zuständig. Für jeden Kommandozeilenparameter wird versucht, ihn als Dateiname oder URL auszuwerten und das referenzierte Dokument zu öffnen. Auf diese Weise konnten die Programmstellen, die den Öffnungsvorgang für ein neues Dokument auslösen, auf eine Stelle konzentriert werden. Das Objekt der Ebene 4 hat keine direkte graphische Repräsentation sondern verrichtet unsichtbar seine Verwaltungsdienste.

Für jedes geöffnete Hilfe-Dokument existiert in *Ebene 3* ein Objekt der gleichnamigen Klasse. Ein Objekt verwaltet zum einen das Dokument, das es repräsentiert und zum anderen eine nichtleere Menge von Fenstern, in denen jeweils ein Dokument dargestellt wird. Existiert von einem Dokument kein Fenster mehr wird auch das Objekt in dieser Ebene verworfen. Um den Bezug zum Dokument-Inhalt herzustellen, enthält das Objekt eine Referenz auf das Dokument-Datenmodell. Objekte der Ebene 3 haben keine direkte graphische Repräsentation sondern verrichten unsichtbar ihre Verwaltungsdienste.

Für jedes geöffnete Hilfe-Dokument – und somit für jedes Objekt der Ebene 3 – existiert in *Ebene 2* jeweils mindestens ein Objekt der Klasse „Hilfe-Dokument-Fenster“. Ein solches Objekt tritt in der graphischen Oberfläche als Hauptfenster in Erscheinung. Das Fenster beherbergt eine Menüleiste, eine Werkzeugleiste und eine Menge von Komponenten aus Ebene 1.

Die spezialisierten Ansichten in *Ebene 1* übernehmen zum einen die Darstellung von Hilfe-Dokumenten und zum anderen die Darstellung von Metadaten des Systems. Alle Klassen dieser Ebene sind JavaBeans. Ihre Programmierschnittstellen sind in Anhang A auf Seite 77 wiedergegeben. Durch die Behandlung der verschiedenen Ansichten von diversen Beans wird die Darstellung einer einzelnen Bean sehr spezifisch. Dies führt dazu, daß eine solche Bean im allgemeinen nicht mehr sinnvoll weiter zerteilbar ist, da sie eine nur ganz bestimmte Aufgabe erfüllt. Auf diese Weise wird dem Ziel entsprochen, daß zumindest die Beans aus Ebene 1 in praktisch jeder beliebig organisierten graphischen Oberfläche wiederverwendet werden können.

Im Gegensatz dazu sind die Klassen aus Ebene 2 bis 4 nur in einer bestimmten Oberflächenorganisation verwendbar. Jedoch übernehmen sie ausschließlich verwaltende Aufgaben, die organisationsspezifisch sind und wohl nur in wenigen Fällen überhaupt auf andere Szenarien übertragbar sind. Insofern denke ich, daß durchaus ein Großteil der Oberfläche wiederverwendbar ist.

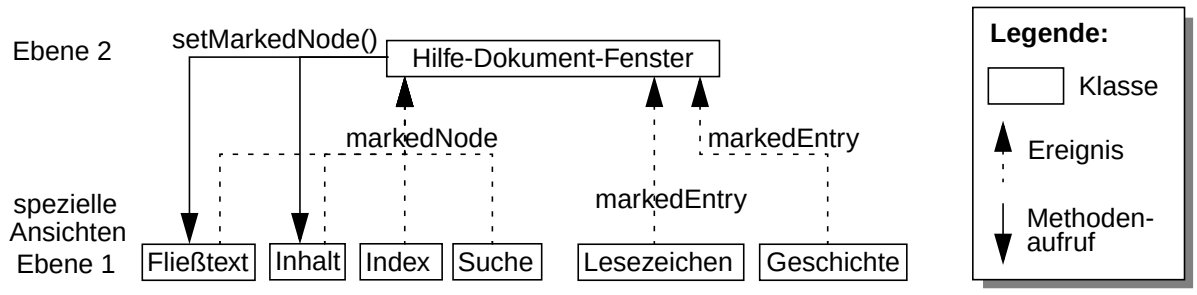
Nachdem nun die Ebenen der graphischen Oberfläche vorgestellt wurden, möchte ich die Hauptmerkmale der Kommunikation zwischen Ebenen erläutern. Wichtig sind hier vor allem zwei verschiedene Ereignisse, die in den zwei folgenden Abschnitten näher erläutert werden.

### 6.6.1 Die gebundene Eigenschaften „markedNode“ und „markedEntry“

Das erste Ereignis, das in der graphischen Oberfläche eine wichtige Rolle spielt, ist das Ereignis, das bei der Wertänderung zweier gebundener Eigenschaften ausgesandt wird.

Es wird davon ausgegangen, daß innerhalb eines Hilfe-Dokument-Fensters höchstens ein spezifischer Knoten aus dem DOM-Baum des Dokuments selektiert ist. Diese Selektion äußert sich im Inhaltsverzeichnis durch eine Markierung auf dem entsprechenden Gliederungsknoten. Die Fließtext-Ansicht zeigt den Inhalt des umgebenden Kontext eines selektierten DOM-Knotens. Im Inhaltsverzeichnis, Index, den Lesezeichen, der Geschichte oder innerhalb der Suchergebnisliste kann der Benutzer die Selektion eines spezifischen Knotens veranlassen, wodurch sich die Markierung im Inhaltsverzeichnis und die Fließtext-Ansicht ändern.

**Abbildung 6–7: Die gebundenen Eigenschaften „markedNode“ und „markedEntry“**




Die eben beschriebene Funktionalität wird durch eine gebundene Eigenschaft realisiert. Die vier Beans Fließtext, Inhalt, Index und Suche, die spezialisierte Ansichten eines Hilfe-Dokuments erzeugen, besitzen zu diesem Zweck die gebundene Eigenschaft *„markedNode“*. Bei Veränderung des Eigenschaftswertes innerhalb einer Bean, sei es durch Benutzerinteraktion in der Bean-Ansicht oder auch, indem ein Objekt einer übergeordneten Ebene den Wert von außen verändert, wird ein Ereignis ausgesandt. Das umgebende Kontainer-Objekt in Ebene 2 empfängt dieses Ereignis und aktualisiert daraufhin seine eigene „markedNode“-Eigenschaft. Eine Veränderung deren Wertes bewirkt anschließend die Benachrichtigung der Ansicht für

Inhalt und Fließtext. Durch Methodenaufrufe wird deren „markedNode“-Eigenschaft auf den neuen Wert gesetzt und die Ansichten aktualisieren ihre Darstellung. Dieser Mechanismus propagiert die Wertänderung der gebundenen Eigenschaft zu allen Oberflächenteilen, die darauf reagieren müssen. Abbildung 6–7 zeigt die verschiedenen Informationsflüsse zwischen einem Hilfe-Dokument-Fenster und seinen enthaltenen Ansichts-Beans der Ebene 1.

In analoger Weise verhalten sich die Beans für Lesezeichen und für die Geschichte, die Ansichten der Metadaten erzeugen. Ihre gebundenen Eigenschaften sind allerdings nicht vom Typ eines DOM-Knotens sondern vom jeweiligen Eintragstyp des zugehörigen Datenmodells. Folglich trägt die gebundene Eigenschaft in diesen Fällen den Namen „*markedEntry*“.

### 6.6.2 Das Ereignis „LayerEvent“

Das zweite Ereignis, das in der graphischen Oberfläche eine wichtige Rolle spielt, ist das „LayerEvent“. Es dient zur Kommunikation zwischen Ebenen von unten nach oben. Das Ereignis transportiert eine Aktion, die in der Oberfläche ausgelöst wurde, durch die Ebenen bis ein Kontext erreicht wird, in dem die Aktion behandelt werden kann. Im Einzelnen handelt es sich bei den Aktionen um Benutzerauswahlen aus den Menüs und Werkzeugleisten der Hauptfenster sowie die Auswahl von Menüeinträgen aus Kontextmenüs der einzelnen Beans in Ebene 1.

## 6.7 Modul für die Fließtextansicht

Die Fließtext-Darstellung eines Hilfe-Dokuments ist sicherlich die wichtigste Ansicht innerhalb des Hilfesystems. Gleichzeitig bringt diese Ansicht durch ihre umfangreichen Formatierungsaufgaben und Unterstützung für Hyperlinks ein hohes Maß an Schwierigkeiten mit sich. Aus diesem Grund habe ich während des Entwurfs besonderes Augenmerk auf mögliche Umsetzungsvarianten für die Fließtext-Darstellung gerichtet. Im folgenden möchte ich zunächst kurz die Anforderungen an das Modul für die Fließtext-Ansicht charakterisieren. Anschließend folgt eine Übersicht über die Eigenschaften der graphischen Text-Komponenten, die Javas Swing-Toolkit mitbringt – vor allem im Hinblick auf deren Unterschiede zu DOM als zentralem Hilfe-Dokument-Modell im System. Mit den nun zur Verfügung stehenden Grundlagen erfolgt eine Diskussion über verschiedene Lösungsansätze. Das Hauptziel einer optimal zu bezeichnenden Lösung ist dabei die nahtlose Integration in die Text-Komponenten von Swing. Der Hintergrund dafür ist der Versuch, möglichst große Teile der bestehenden Komponenten und ihrer durchdachten Architektur weiternutzen zu können.

Wenn im folgenden spezifische Schnittstellen-, Klassen- und Methoden-Namen verwendet werden, so geschieht dies deshalb, weil durch einen solchen Namen leichter ein eindeutiger Bezug hergestellt werden kann, als würde ein künstlich erfundener Begriff verwendet. Kenntnis der Klassenbibliotheks-Dokumentation wird nicht vorausgesetzt, da die Erläuterungen in sich abgeschlossen und unabhängig sind.

### 6.7.1 Anforderungen an den Entwurf des Moduls

Grundsätzlich ist das verwendete Datenmodell die Repräsentation eines – evtl. auch mehrerer – DocBook-XML Dokumente in Form einer DOM-Datenstruktur. Die DOM-Programmierschnittstelle ist vom W3C standardisiert. Diese Standardisierung und eine Vielzahl von Implementierungen dieser Schnittstelle, sowie Implementierungen, die darauf aufbauen, bekräftigen den Entschluß, DOM als zentralen Ausgangspunkt für das Dokument-Modell zu verwenden.

DOM bietet dem Programmierer den Zugriff auf ein im Speicher befindliches XML-Dokument, das bereits geparst wurde. Der Zugriff über Methoden auf das Dokument entspricht dem

Zugriff auf eine Datenstruktur in Baumform. Dabei sind die Knoten in diesem Baum die Bausteine, aus denen sich ein XML-Dokument zusammensetzt. Zu diesen Bausteinen zählen unter anderem Zeichendaten, Elemente und CDATA-Abschnitte. In jedem dieser Baumknoten sind die lokalen Informationen, wie beispielsweise Textinhalt, Attribute oder Kindelemente eines Elements, verfügbar.

Aufgabe dieses Moduls ist nun die Visualisierung des, in der DOM-Struktur gespeicherten, XML-Dokuments als formatierten Fließtext, ähnlich wie dies bei Webbrowsern für HTML-Web-Seiten geschieht. Außerdem muß Benutzerinteraktion mit Textteilen möglich sein, um Hyperlinks oder Kontextmenüs zu realisieren. Die Implementierung soll in Java auf Basis des JDK 1.2 (Java Development Kit) und dem darin enthaltenen graphischen Oberflächen-Toolkit Swing erfolgen. Die in Swing enthaltenen, sehr flexiblen Text-Komponenten bereiten die Grundlage zur Anzeige von formatiertem Fließtext. Durch deren Einsatz ergeben sich Richtlinien für den Entwurf, um eine möglichst glatte Integration mit dem vorgegebenen Toolkit zu erreichen. Im folgenden Abschnitt werden die wichtigsten Eigenschaften der Swing Text-Komponenten, in Bezug auf das zu entwerfende Modul, aufgezeigt.

### 6.7.2 Eigenschaften der Swing Text-Komponenten

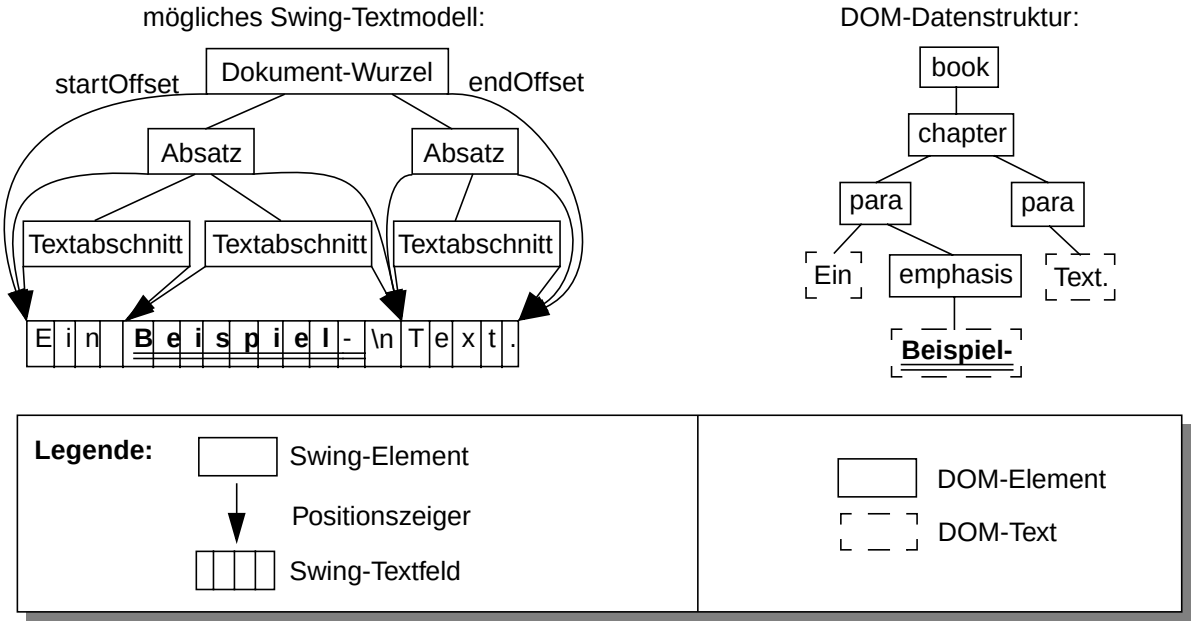
Für die im folgenden beschriebenen Swing Text-Komponenten wurde die folgende Literatur herangezogen: [Friendly et al. 1999], [Eckstein et al. 1998], [Prinzing 2000], [Violet-1 2000], [Violet-2 2000], [Violet-3 2000] sowie die JDK-Klassenbibliotheks-Dokumentation und JDK-Quelltexte.

Swing verwendet als durchgehendes Entwurfsmuster eine leicht abgewandelte Form des Model-View-Controller (MVC) Prinzips. Der Unterschied zum reinen MVC besteht in der Zusammenfassung von View und Controller zum sogenannten „delegate“. Dieser Unterschied ändert jedoch nichts an der wichtigen Tatsache, daß eine strikte Trennung der Daten im Modell von den jeweiligen Ansichten (views) der Daten vorhanden ist.

Die Swing Text-Komponenten verwenden als Dokumentmodell ein Java-Interface (`javax.swing.text.Document`) bzw. davon abgeleitete Schnittstellen und die dazugehörigen Implementierungen in Java-Klassen. Das Modell repräsentiert formatierten Text, indem für verschieden formatierte Textteile Knoten (`javax.swing.text.Element`) in einer Baumstruktur eingesetzt werden, die in ihrer Attributierung jeweils ausschließlich Formatierungshinweise (`javax.swing.text.AttributeSet`) enthalten. Soweit ist das Textmodell dem DOM aus XML recht ähnlich, zumal dieses Design in Anlehnung an SGML-Strukturen entwickelt wurde, unter anderem wohl, um einfach HTML darstellen zu können. Der reine, unformatierte Textinhalt ist dagegen in einem separaten, eindimensionalen Feld (Zugriff über die Methode: `javax.swing.text.Document.getText(offset, length)`) abgelegt. Die Verbindung der Baumknoten mit diesem Textfeld erfolgt mit Hilfe von ganzzahligen Positionszeigern, die in jedem Baumknoten den Anfang (`javax.swing.text.Element.getStartOffset`) und das Ende (`javax.swing.text.Element.getEndOffset`) eines gleichartig formatierten Textteils im Textfeld spezifizieren. Dies ist der Hauptunterschied zur DOM-Datenstruktur, die Textteile direkt innerhalb von Blattknoten vom Typ Text ablegt (`org.w3c.dom.Text.getData()`). Abbildung 6–8 auf Seite 57 zeigt das Swing Textmodell und DOM in der direkten Gegenüberstellung.

Ihre Flexibilität erreichen die Swing Text-Komponenten durch gezielte Arbeitsteilung. Die im folgenden kurz skizzierten Klassen bzw. Schnittstellen werden in jeder Text-Komponente, die Swing zur Verfügung stellt, verwendet. Durch ihr Zusammenspiel ergibt sich eine umfangreiche Textdarstellungs- und Textbearbeitungsumgebung.

**Abbildung 6–8: Swing Textmodell und DOM in der Gegenüberstellung**




- `javax.swing.text.View` ist eine Schnittstelle, die leichtgewichtige Teilansichten einer Text-Komponente implementieren müssen. Es handelt sich bei diesen Teilansichten nicht um vollwertige Swing-Komponenten. Aus Performance-Gründen teilen sich alle Teilansichten die ihnen gemeinsame, sie umgebende Text-Komponente. Jede Teilansicht ist dafür zuständig, für einen bestimmten Textabschnitt eine graphische Darstellung zu zeichnen.
- `javax.swing.plaf.TextUI` repräsentiert die Benutzungsschnittstelle einer Text-Komponente. Sie ist zuständig für das Erscheinungsbild bezüglich der Farben für Text, Hintergrund, Markierungen, Cursor usw. Diese Klasse richtet eine Implementierung eines Cursors, sowie eine Tastaturbelegungstabelle ein. Außerdem baut sie in den einfacheren Text-Komponenten die Ansicht (view) des Dokumentmodells auf. In den komplexeren Komponenten wird diese Aufgabe an die Klasse EditorKit delegiert.
- `javax.swing.text.EditorKit` stellt Mechanismen zur Verfügung, um mit einer bestimmten Art von Textinhalt (content-type) umgehen zu können. Das EditorKit kreiert ein Textmodell und bietet eine Erzeuger-Klasse (`javax.swing.text.ViewFactory`), die zu den jeweiligen Formatierungs-Knoten im Textmodell die entsprechende Teilansichten generiert. Weiterhin sind Kommandos (`javax.swing.Action`) als innere Klassen definiert, die beispielweise über die Tastaturbelegungstabellen ausgelöst werden und auf diese Weise den Text per Tastatur bearbeitbar machen. Schließlich speichert das EditorKit die Daten aus dem Textmodell in einen Zeichenstrom (stream) bzw. liest Daten aus einem Zeichenstrom und baut daraus das Textmodell auf.

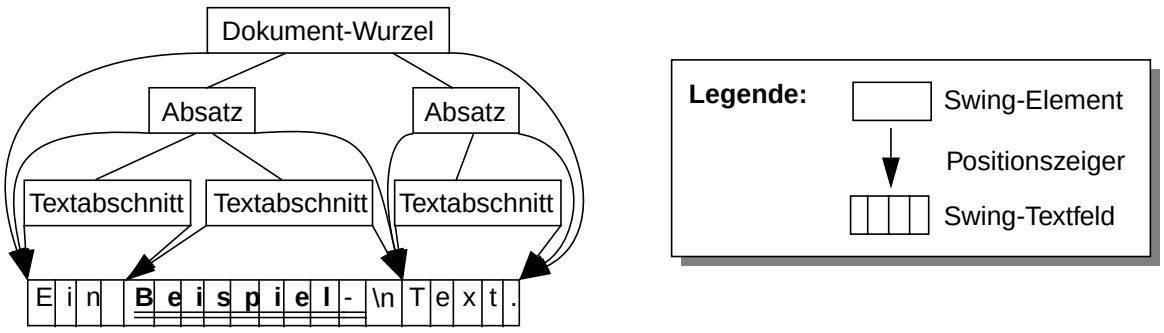
Konkrete Implementierungen der Text-Komponenten in JDK 1.2 sind die Klassen `JTextField`, `JPasswordField`, `JTextArea`, `JEditorPane` und `JTextPane`. Die ersten drei stellen ausschließlich einheitlich formatierten Text dar und sind somit in diesem Entwurf nicht verwendbar. `JEditorPane` ermittelt anhand der Dateiendung bzw. des MIME-Types eines Textes, um welches Textformat es sich handelt und verwendet daraufhin eine passende Implementierung der Text-Komponente. Swing unterstützt von Haus aus die folgenden Formate: Klartext (text/plain), Hypertext-Markup-Language (text/html) und Rich-Text-Format (text/rtf).

Das HTML-Format erschien zu Beginn der Überlegungen eine mögliche Lösung. In Verbindung mit Norman Walsh's modularen XSLT-Stylesheets für DocBook[1] läßt sich direkt aus der

DOM-Struktur mit Hilfe eines XSLT-Prozessors wie beispielsweise Xalan-J[1] ein HTML-Dokument bzw. eine Menge von Teildokumenten generieren. Für die Benutzerinteraktion und erweiterte Suchfunktionalität ist es aber notwendig, den Bezug von der Ansicht zum zugehörigen DOM-Modell aufrechtzuerhalten. In diesem Fall würde dies bedeuten, daß die Stylesheets verändert werden müßten, um redundante Information in die generierten HTML-Dokumente zu übertragen. Auf diese Weise sollen möglichst alle Informationen des ursprünglichen DOM-Modells erhalten werden bzw. durch Rückverweise zur DOM-Struktur verfügbar gemacht werden. Diese Lösung erscheint mir zwar möglich aber nicht sehr sauber. Auch würde durch die nötige Einarbeitung in die sehr umfangreichen Stylesheets, um gezielt erwünschte Veränderungen vornehmen zu können, die Zeit zur Umsetzung sehr knapp werden. Darüberhinaus ist es sehr ungewiß, ob die Veränderungen an den Stylesheets mit Hilfe benutzerdefinierter Anpassungen (customizing) derart durchgeführt werden können, daß sie auch bei neu erscheinenden Stylesheet-Versionen anwendbar bleiben. Ansonsten wären die Veränderungen für immer an eine bestimmte Version gebunden.

Die zuletzt genannte funktionsreichste Text-Komponente `JTextPane` erlaubt dem Programmierer, beliebig formatierten Text darzustellen. Allerdings gibt sie ein Textmodell vor, bei dem der Formatierungsbaum exakt drei Ebenen tief ist. Unter der Wurzel befinden sich Knoten, die Formatierungen für einen Absatz enthalten, d.h. eine Block-Formatierung vornehmen. Unter diesen liegen Knoten mit Formatierungshinweisen für einen Textteil innerhalb eines Absatzes, d.h. sie übernehmen Inline-Formatierung. Dies ist für eine WYSIWYG-Textverarbeitung ausreichend. Abbildung 6–9 zeigt beispielhaft die Struktur des Textmodells, wobei die Zeichensequenz „\n“ für eine neue Zeile und somit einen neuen Absatz steht. DocBook-XML

**Abbildung 6–9: Datenmodell der Swing-Text-Komponente JTextPane**




Dokumente können aber durchaus mehr Verschachtelungsebenen enthalten. Bei direkter Verwendung der `JTextPane` müßte also die DOM-Struktur auf maximal zwei Ebenen „flachgedrückt“ werden. Hier ergibt sich dann ähnlich wie bei der Lösung über die HTML-Ansicht das Problem, daß redundante Information aus dem DOM-Modell mitgeführt werden müßte, um die Beziehung zwischen Ansicht und Modell aufrechtzuerhalten. Im Gegensatz zur Lösung mit HTML ist dieser Ansatz möglicherweise besser zu realisieren, da keine vorhandenen Stylesheets verändert werden müssen, sondern die Redundanzinformation in Hash-Datenstrukturen abgelegt werden könnte.

Aus den bisherigen Überlegungen läßt sich schließen, daß keine der Text-Komponenten direkt verwendet werden kann, um eine DOM-Struktur als formatierten Fließtext anzuzeigen. Statt-

1.http://nwalsh.com/docbook/xsl/
1.http://xml.apache.org/xalan/

dessen ist es notwendig ein Adapter-Modul zu entwerfen, das zwischen dem DOM als Modell und der Ansicht vermittelt. Im folgenden Abschnitt werden verschiedene Lösungsansätze hierfür aufgezeigt und bewertet.

### 6.7.3 Lösungsansätze für ein Adapter-Modul zwischen DOM-Modell und Text-Komponenten-Ansicht

Auf den ersten Blick erscheint mir die idealste Lösung, die Ansicht so zu verändern, daß DOM als Modell direkt verwendet werden kann. Ergebnis einer solchen Vorgehensweise wäre ein allgemein verwendbares Darstellungs- und evtl. Bearbeitungs-Modul für XML-Dokumente, basierend auf der DOM-Datenstruktur. Dies kann auf zwei Arten erreicht werden.

Zum einen könnten die vorhandenen Ansichts-Klassen (`javax.swing.text.…View`) durch Ableiten so wiederverwendet werden, daß sie statt auf das von Swing vorgegebene Textmodell (`javax.swing.text.Document`) auf DOM zugreifen, um die zur Darstellung benötigten Informationen aus dem Modell zu beziehen. Nach eingehendem Studium des Swing Java-Quelltextes hat sich allerdings herausgestellt, daß die Verbindung der Ansichten zum Swing-Dokument-Modell durch fest programmierte Zugriffe über Modell-Methoden zu eng ist, um durch Ableiten noch verändert werden zu können.

Der zweite Ansatz folgt nun direkt aus der eben genannten Tatsache. Die Ansichts-Klassen müßten auf Quelltext-Ebene an das DOM-Modell angepaßt werden, indem soviel Quelltext wie möglich direkt übernommen wird. Alle Programmstellen, die auf das Modell zugreifen, werden auf DOM-Zugriffe umgestellt. In der vorgegebenen Zeit der Studienarbeit ist dieses Verfahren nicht durchführbar. Die Zusammenhänge in den Ansichts-Klassen in Verbindung mit dem Dokumentmodell und EditorKits sind hochgradig komplex. Es müßten große Teile der Swing Text-Komponenten neu implementiert werden, was im Rahmen dieser Arbeit nicht möglich ist. Außerdem ist nicht klar, inwiefern derartige Veränderungen am JDK-Quelltext vorgenommen und weiterverbreitet werden dürfen. Problematisch ist in diesem Zusammenhang vor allem, daß der SESAMDoc-Browser unter dem Lizenzmodell der GPL veröffentlicht werden soll. Die direkte Veränderung von Quelltext und „Wiederverwendung“ von Quelltexten durch Kopieren und Ausschneiden ist sicherlich auch nicht die software-technische Methode der Wahl und insofern abzulehnen.

Die bisherigen Betrachtungsergebnisse führen zu dem Schluß, daß an den vorhandenen Swing Text-Komponenten möglichst wenig zu ändern ist. Stattdessen sollte sich das DOM-Modell an die vorhandene Swing-Umgebung anpassen.

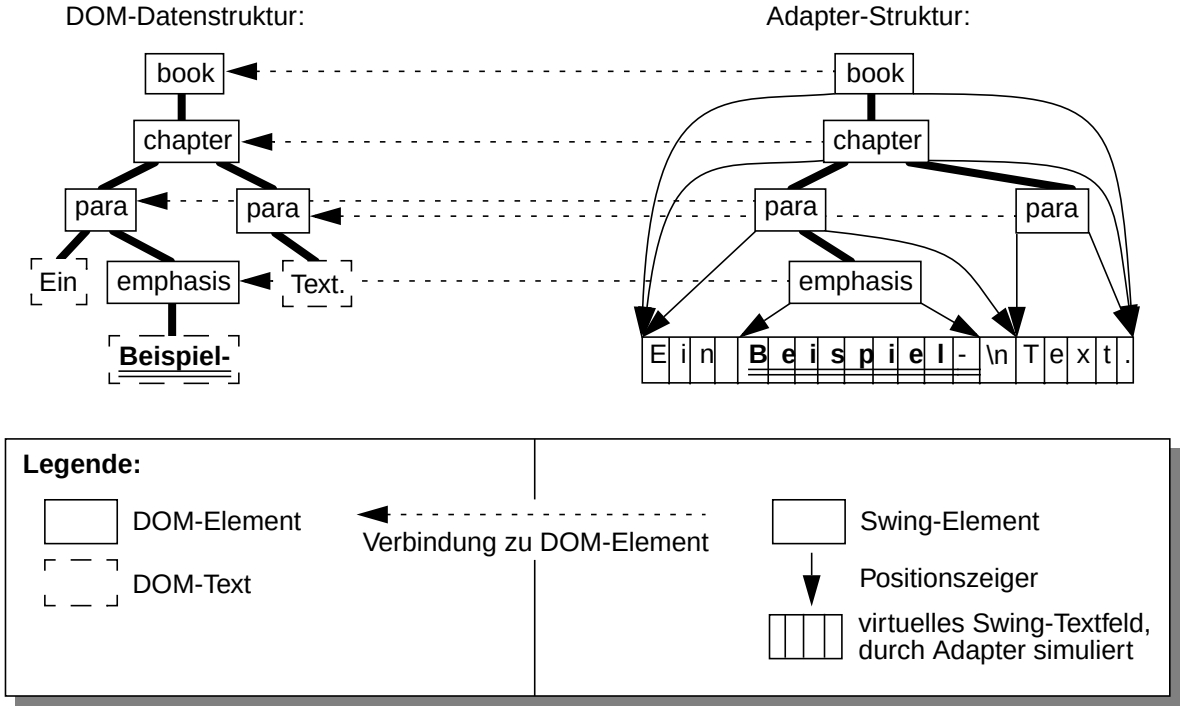
Eine Art, wie sich das DOM-Modell an die Swing Text-Komponenten anpassen läßt, wäre ein Adapter, der als Schicht zwischen DOM und Swings Textmodell vermittelt. Dies erspart Änderungen an DOM bzw. an Swings Text-Komponenten. Allerdings müßten dadurch zwei Modelle für ein und dieselbe Ansicht ständig mitgeführt werden. Für eine bearbeitbare Fließtext-Ansicht betrachte ich diese Lösung als am geeignetsten. Ist aber eine nur-lesbare Ansicht, wie in dieser Studienarbeit, erforderlich, war ich zunächst der Meinung, daß eine direktere Bindung zwischen DOM und Swing-Ansicht vorgenommen und somit auf die doppelte Modell-Führung verzichtet werden kann.

Der letzte Lösungsansatz verfolgt das Ziel, das DOM-Modell so zu kapseln, daß es für die vorhandenen Swing-Ansichten wie das Swing Textmodell aussieht. Auf diese Weise wird DOM so verpackt, daß es die unveränderten Text-Komponenten direkt als ihr eigenes Modell verwenden können.

Zur Kapselung von DOM sind die Java-Interfaces aus dem Swing-Modell in jedem DOM-Baumknoten zu implementieren. Dies ist möglich sofern der XML-Parser, der die DOM-Struktur aufbaut, sogenannte DOM-Beans erzeugen kann. Dabei handelt es sich um die Möglichkeit, selbst definierte, abgeleitete Klassen der Baumknoten-Implementierungen beim Aufbau der DOM-Struktur durch den Parser verwenden zu lassen. Bei Verwendung des Xerces-J[1] als XML-Parser bedeutet dies, daß ein eigener Parser definiert wird. Dieser ist vom DOM-Parser in Xerces-J (`org.apache.xerces.parser.DOMParser`) abgeleitet und registriert im Konstruktor über eine geschützte Methode (`protected setDocumentClassName(name)`) eine eigene Implementierung der DOM-Dokument-Schnittstelle (`org.w3c.dom.Document`). Diese Schnittstelle wird am besten implementiert, indem von der bestehenden Implementierung in Xerces-J (`org.apache.xerces.dom.DocumentImpl`) abgeleitet wird und die benötigten „`create…()`“ Methoden überschrieben werden. Die überschriebenen Methoden liefern beim Aufbau des DOM-Baums eigene DOM-Beans an die Parser-Maschinerie, die diese Beans statt den Standard-Implementierungen in den Baum einhängt.

Eine Probe-Implementierung der eben beschriebenen Vorgehensweise ergab, daß dabei durch eine redefinierte Methode eine Kollision erzeugt wird. Die Methode „`NamedNodeMap org.w3c.dom.Node.getAttributes()`“, die Bestandteil jedes DOM-Knotens ist, ist mit derselben Signatur auch im Swing Textmodell als „`AttributeSet javax.swing.text.Element.getAttributes()`“ enthalten. Da sich die Methoden nur in ihrem Rückgabewert unterscheiden, ist es nicht möglich, beide Interfaces in derselben Klasse zu implementieren. Somit ist die Realisierung mittels DOM-Beans nicht möglich.

**Abbildung 6–10: Eingehüllte DOM-Struktur gibt sich als Swing-Textmodell aus.**




Ein Ausweg besteht nun darin, eine eigene Implementierung der Swing Textmodell-Schnittstelle vorzunehmen. Diese soll dasselbe Baumgerüst wie die zugehörige DOM-Struktur repräsentieren, erhält ihre Modell-Daten aber aus dem darunterliegenden, eingehüllten DOM-

1.http://xml.apache.org/xerces-j/

Baum. Gegenüber der Verwendung von DOM-Beans muß nun die Verwaltung der Baumstruktur, d.h. beispielsweise Listen von Kindknoten, nochmals implementiert werden. Abbildung 6–10 zeigt die Zusammenhänge zwischen dem Adapter, der die Swing Textmodell-Schnittstelle implementiert und dem verbundenen DOM-Baum, aus dessen Textknoten ein virtuelles Textfeld simuliert wird.

Für die Ansicht der oben beschriebenen Implementierung des Textmodells mit beliebig großer Baumtiefe muß eine spezielle ViewFactory zur Verfügung gestellt werden. Diese Factory liefert für die Dokumentbaum-Elemente spezielle Ansichten (views), die von den in Swing vorhandenen abgeleitet sind. Im einzelnen sind diese speziellen Ansichten einmal für DocBook-Elemente die blockweise formatiert werden und einmal für DocBook-Elemente die inline formatiert werden zu erstellen. Die konkrete Formatierung muß so im Quelltext vorgegeben werden, daß jederzeit eine Erweiterung zu flexiblen konfigurierbaren Formatierungshinweisen möglich ist. Für Ansichten mit Blockformatierung wird `javax.swing.text.ParagraphView` abgeleitet und für Ansichten mit Inline-Formatierung `javax.swing.text.LabelView`. DocBook-Elemente, die Hyperlink-Funktionen übernehmen, müssen evtl. mit einer eigenen Ansichten-Klasse behandelt werden. Dies ist notwendig, um sowohl das optische Erscheinungsbild eines Links zu erzeugen als auch, um die Link-Verfolgung durch Mausklick oder ähnliches zu realisieren.

Nach Abschluß der Entwurfsphase war ich mir trotz intensiver Auseinandersetzung mit der Materie nach wie vor unsicher, welche Lösungsart überhaupt im Endeffekt umsetzbar sein würde. Die Präferenz lag sicherlich bei der oben zuletzt vorgestellten Möglichkeit, DOM durch Umhüllung zu kapseln und somit wenigstens annähernd eine doppelte Datenhaltung zu vermeiden. Die letztendliche Entscheidung habe ich schließlich erst während der Implementierung getroffen.

### 6.7.4 Umsetzung der Fließtext-Ansicht

Nachdem ein Grundgerüst der Gesamtsystem-Funktionalität umgesetzt war und sich als tragfähig erwiesen hatte, mußte ich mich für eine konkrete Umsetzung der Fließtext-Ansicht entscheiden. Während der, seit dem Entwurf, vergangenen Zeit haben sich leider wenig neue Erkenntnisse, die eine Entscheidung vereinfacht hätten, ergeben. Ein erneutes Studium der Swing-Quelltexte ergab zumindest, daß die Entwickler der Text-Komponenten durchaus vorgesehen hatten, in der HTML-Ansicht auch anders strukturierte SGML- und somit evtl. auch XML-Daten zu speichern und darzustellen. Implementierungsversuche mit einem Aufwand von einigen Tagen blieben angesichts der Komplexität einer solchen Anpassung erfolglos. Schlußendlich mußte ich mir eingestehen, daß vorerst nur eine *einfache Transformation des* DOM-*Baums* auf das Textmodell der `JTextPane`-Komponente zum Erfolg führen würde.

Das *Grundprinzip* der Implementierung der Fließtext-Ansicht ist eine Preorder-*Traversierung* des DOM-Dokumentmodells. Während der Baumtraversierung werden für gleichartig zu formatierende Textabschnitte diese Abschnitte jeweils in die JTextPane-Komponente eingefügt. Das Einfügen geschieht mit Hilfe einer Methode, die eine beliebige Zeichenkette einheitlicher Formatierung an einer bestimmten Stelle im JTextPane-Dokumentmodell einfügt (`void javax.swing.text.Document.insertString(int offset, String str, AttributeSet a)`). Die Formatierung wird festgelegt durch eine Attributmenge, die als Parameter mitübergeben wird. Wie in Abschnitt 5.2.5 „Fließtext-Ansicht“ auf Seite 35 beschrieben, wird die Formatierung anhand von DocBook-Elementnamen entschieden. Eine eigene Klasse bestimmt die Zuordnung von Elementnamen zur zugehörigen Formatierung in Form einer Attributmenge.

Entsprechend der Spezifikation wurde zunächst beim Aufbau eines Dokumentfensters im SESAMDoc-Browser das gesamte Hilfe-Dokument transformiert und dargestellt. Bedingt durch die Vielzahl von Auszeichnungsmöglichkeiten in DocBook-Dokumenten werden während der Transformation meist eine Ummenge kleiner Textabschnitte wie oben beschrieben in die Fließtext-Ansicht eingefügt. Für jeden dieser Abschnitte sind wiederum eine Menge von Methodenaufrufen notwendig, so daß die Zeit zum Aufbau der Ansicht für den Benutzer inakzeptabel lang wurde. Aus diesem Grund habe ich eine *Aufteilung in Dokumentteile* vorgenommen. Jeder Teil beginnt bei einem Gliederungspunkt wie er im Inhaltsverzeichnis angezeigt wird und endet vor dem logisch nächsten Gliederungspunkt. Auf diese Weise ergibt sich zu jedem Knoten im Inhaltsbaum ein bestimmter Dokumentteil, der als Fließtext angezeigt wird.

Die entstandene Umsetzung ist bedingt durch die vergleichweise kurze Zeit, die zur Verfügung stand, leider teilweise fehlerhaft und unvollständig. Beispielsweise ist die korrekte Behandlung von Leerraum zwischen Wörtern noch unvollständig. Dies liegt daran, daß der DOM-Baum sämtliche Leeräume widerspiegelt, die in der Textform des zugrundeliegenden XML-Dokuments enthalten waren – dazu gehören auch neue Zeilen und Leerzeichen bzw. Tabulatoren zur Einrückung. Nun sind aber nur bestimmte Leerräume signifikant, z.B. soll zwischen zwei Wörtern nur ein Leerzeichen dargestellt werden, auch wenn im XML-Dokument eine neue Zeile und einige Tabulatoren zur Einrückung verwendet wurden. Bei Beispielen, die Programm-Quelltext enthalten, ist dagegen wiederum jeder einzelne Leeraum signifikant und muß in der Darstellung erhalten bleiben. Eine saubere, algorithmische Behandlung dieser Fälle war in der Kürze der Zeit nicht mehr möglich.

Weiterhin existiert durch die Vielzahl von DocBook-Elementen und Attributen eine große Menge an Formatierungsregeln. All diese Regeln können nicht auf einen Schlag realisiert werden. Einige wichtige, ausgewählte werden bereits gesondert behandelt. Für eine zukünftige Erweiterbarkeit wäre es sinnvoll, einen Mechanismus vorzusehen, über den für jeden Elementtyp eine eigene Klasse dessen Sonderfälle bei der Transformation berücksichtigt. Auf diese Weise lassen sich jederzeit benötigte Elementformatierungen hinzufügen und somit Schritt für Schritt eine vollständige Sammlung aufbauen. Im Prinzip übernehmen diese Elementformatierungs-Klassen die Aufgaben eines Stylesheets. Zwar erscheint es so, als ob Stylesheet-Funktionalität dadurch neu erfunden wird, jedoch sehe ich keine bessere Möglichkeit, eine Transformation vorzunehmen und dabei gleichzeitig Verbindungen zwischen den Dokumentbaum-Knoten vor und nach der Transformation aufrechtzuerhalten.

Schließlich ist es noch überlegenswert, ob nicht eine Menge von transformierten Textabschnitten in einem Cache zwischengespeichert wird. Es ist sicherlich nicht ungewöhnlich, daß der Benutzer mehrmals denselben Dokumentteil betrachtet, indem er z.B. in der Geschichte zurückspringt. Zur Zeit bedeutet der Sprung zu einem anderen Dokumentteil eine erneute Transformation, die Zeit kostet. Durch einen Zwischenspeicher könnte diese Zeit teilweise eingespart und dem Benutzer ein schnellerer Ansichtswechsel ermöglicht werden.

# 7 Realisierung

## 7.1 Verwendete Werkzeuge

### 7.1.1 Programmiersprache Java-2

Grundlegendes Entwicklungswerkzeug war das Java Development Kit (JDK) von Sun in der Version 1.2.2. Das JDK liefert neben dem Übersetzer, der aus Java-Quelltext Bytecode generiert, weitere Hilfswerkzeuge, beispielsweise einen einfachen Debugger für die Fehlersuche. Ebenso wichtig ist die mitgelieferte Klassenbibliothek, die Java quasi erst zu einer brauchbaren Sprache macht. In der Bibliothek enthalten ist neben einer Vielzahl von Datenstrukturen und Ein-/Ausgabe-Klassen auch das Swing-Toolkit zur Erstellung graphischer Benutzungsoberflächen.

Als teilweise schwer nachvollziehbares Problem hat sich Javas fehlende Sprachunterstützung für typsichere Aufzählungen herausgestellt. Ein einfacher Entwurfsmuster-Ansatz wird in Büchern zu Java vorgestellt, unter anderem in [Niemayer et al. 1997]. Dieser einfache Ansatz stützt sich auf eine willkürliche Ansammlung von ganzzahligen, statischen Konstanten, die innerhalb einer Klasse als deren Attribute implementiert werden. Für einen einfachen Aufzählungstyp, beispielsweise für die drei Farben einer Verkehrsampel, ergibt sich entsprechend dem Muster etwa ein Java-Quelltext wie er in Beispiel 7–1 dargestellt ist.

**Beispiel 7–1: Einfacher Aufzählungstyp**

```
public class Ampel {
  public final static int ROT = 0;
  public final static int GELB = 1;
  public final static int GRUEN = 2;
}
```

Auf den ersten Blick ist gegen diese Vorgehensweise nichts einzuwenden. Probleme ergeben sich aber zwangsläufig dann, wenn die Attribute der Klasse Ampel als Aufzählungselemente in anderen Klassen verwendet werden. Nachträgliche Veränderungen an der Ampel-Klasse erkennen die meisten Projektverwaltungen für Java nur unzureichend. Sowohl die IDE Forté als auch der im JDK mitgelieferte Übersetzer erkennen keine Abhängigkeiten[1] zwischen der Ampel-Klasse und den Klassen, in der die Aufzählung verwendet wird. So kommt es, daß Klassen weiterhin alte Definitionen der Aufzählung verwenden, da der Übersetzer die symbolischen Literale bereits bei einer früheren Übersetzung in ihre ganzzahligen Werte aufgelöst hat. Zudem ist der grundlegende Typ der Aufzählung nach wie vor eine Ganzzahl „int“, so daß ungewollt alle Aufzählungen dieser Art zueinander typkompatibel sind.

Nachdem ich selbst einem solchen Verhalten zum Opfer gefallen bin, habe ich – leider zu spät – ein besseres Entwurfsmuster auffinden können, das zumindest für jede Aufzählung einen eigenen Typ erlaubt. Dieses Muster wird beispielsweise in den Java-Sprachbindungen für Corba verwendet. Zur Realisierung der Typsicherheit wird nun statt ganzzahligen Werten ein eigener Klassentyp kreiert. Für jedes Aufzählungselement wird innerhalb dieser Klasse eine statische Instanz derselben angelegt. Diese Instanzen werden dann als Werte verwendet. Um weiterhin schnelle Verzweigungsabfragen gestalten zu können, kommen zusätzlich wiederum für jedes Element ganzzahlige Äquivalente zum Einsatz. Diese können innerhalb eines switch-

1. Offenbar erfolgt eine erneute Übersetzung einer einzelnen Java-Klasse nur, falls ihr eigener Quelltext ein jüngeres Datum als der zugehörige, evtl. von einem früheren Übersetzungsvorgang vorhandene, Bytecode aufweist.

case-Konstrukts verwendet werden. Für den einfachen Aufzählungstyp einer Verkehrsampel ergibt sich nun entsprechend dem neuen Muster etwa ein Java-Quelltext wie in Beispiel 7–2.

**Beispiel 7–2: Typsicherer Aufzählungstyp**

```
public final class Ampel {
  public final static int _ROT = 0;
  public final static int _GELB = 1;
  public final static int _GRUEN = 2;
  public final static Ampel ROT = new Ampel(_ROT);
  public final static Ampel GELB = new Ampel(_GELB);
  public final static Ampel GRUEN = new Ampel(_GRUEN);

  private int farbe; // Wertbehälter für Element-Instanzen

  private Ampel(int farbe) { // versteckter Konstruktor
    // Erzeugen einer Instanz mittels fromInt() !
    this.farbe = farbe;
  }

  public int farbe() { return farbe; }

  public static Ampel fromInt(int farbe) {
    switch(farbe) {
      case _ROT: return ROT; break;
      case _GELB: return GELB; break;
      case _GRUEN: return GRUEN; break;
      default: return null; // Fehlerfall weiter behandeln
    }
  }

}
```

Trotz des offensichtlichen Mehraufwands ist der Einsatz dieses Aufzählungsmusters in jedem Fall dem fehleranfälligen, weiter oben vorgestellten vorzuziehen.

Abgesehen von der fehlenden Sprachunterstützung für Aufzählungen ist Java meiner Meinung nach sehr gut geeignet für die Implementierung innerhalb eines Projekts wie diesem. Strenge Typsicherheit, durchgehende Unterstützung für Ausnahmebehandlung und ein Modulkonzept unterstützen die Erstellung von wartbarem, wiederverwendbarem Programmcode. Das größtenteils ausgereift erscheinende Swing-Toolkit tut durch die ausnahmslose Verwendung des MVC-Prinzips sein Übriges. Swing ist sehr flexibel und deshalb recht umfangreich, bleibt aber durch Einsatz der JavaBean-Muster stets konsistent bedienbar. Einzig die Text-Komponenten sind durch ihre schier unglaubliche Flexibilität äußerst schwierig zu verstehen und zu handhaben.

### 7.1.2 Integrierte Entwicklungsumgebung Sun Forté

Die „Richtlinien für das Projekt SESAMDoc“ empfehlen den Einsatz der integrierten Entwicklungsumgebung (IDE, Integrated Development Environment) „Forté for Java Community Edition“ der Firma Sun. Es handelt sich dabei um eine frei verfügbare Entwicklungsumgebung für Java. Sie ist selbst zu 100% in Java implementiert und entsprechend plattformunabhängig. Tatsächlich war die Ausführung ein und derselben Implementierung unter Linux 2.2.x, Sun Solaris 2.7 und Windows NT 4.0 problemlos möglich. Einzig der damit einhergehende erhöhte

Ressourcenverbrauch – vor allem an Hauptspeicher – schlägt leicht negativ zu Buche, läßt sich aber durchaus verschmerzen. Die Arbeit mit dieser IDE erleichtert die Implementierung vor allem durch die Automatisierung alltäglicher Arbeiten ungemein. Insofern kann ich die Empfehlung nur weitergeben. Eines der herausragenden Merkmale ist sicherlich die Quelltext-Komplettierung. Anhand des aktuellen Java-Quelltext-Kontextes bietet sie dem Programmierer sämtliche verfügbare Klassen bzw. anwendbare Methoden oder Attribute zur Auswahl aus einer eingeblendeten Liste. Die Methoden werden dabei inklusive ihrer Signatur angezeigt, so daß sich oftmals der Blick in die Klassenbibliotheks-Dokumentation erübrigt und syntaktische Fehler von vornherein vermieden werden können.

Nicht verschweigen möchte ich einige Probleme was die Funktionalität von Forté betrifft. Trotz zeitweisem Mitlesen der Benutzer-Mailingsliste zu Forté und intensiver Suche auf den Forté-Web-Seiten, vor allem auch in den Listen bekannter Fehler, ist es mir nicht gelungen, ein Projekt, das auf einem Rechnersystem angelegt wurde, auf einen anderen – sogar identischen – Rechner zu übertragen. Dies ist sehr ärgerlich, da sämtliche Projekteinstellungen bei einem alleinigen Transport der Quelltexte verloren gehen und mühsam per Maus in der IDE neu eingestellt werden müssen. Nicht unbeteiligt an diesem Problem ist sicherlich die Tatsache, daß nur ein Teil der Einstellungen in XML-Dateien abgelegt wird. Die restlichen Konfigurationsdaten werden serialisiert. Unter der Serialisierung ist in Java das persistente Speichern von Klasseninstanzen – also Objekten – auf einem Speichermedium zu verstehen. Leider ist diese Speicherung äußerst kritisch beim Austausch zwischen verschiedenen Versionen der Klassen, die die Serialisierung durchführen, beispielsweise der Klassenbibliothek des JDK.

## 7.2 Geplante, noch nicht realisierte Funktionen

Aufgrund der beschränkt zur Verfügung stehenden Zeit im Rahmen der Studienarbeit konnten ein paar Funktionen aus der Spezifikation des SESAMDoc-Browsers nicht bzw. nicht vollständig implementiert werden. Dies betrifft Teile der Fließtext-Ansicht sowie der Lesezeichen und der Geschichtseinträge. Außerdem fehlt die Implementierung der Suchfunktionalität. Im folgenden werden die fehlenden Teilfunktionalitäten geschildert und die vorgesehenen Lösungswege beschrieben.

- In der Fließtext-Ansicht des Browsers fehlt bislang die beidseitige Zuordnung zwischen DOM-Knoten und den zugehörigen Fließtextabschnitten. Benötigt wird diese Information, um an eine bestimmte Positionen innerhalb des angezeigten Dokumentteils springen zu können, beispielweise als Ziel von Hyperlinks, Lesezeichen oder Suchanfrage-Ergebnissen. Außerdem dient die Information dazu, herauszufinden, welchem DOM-Knoten des Dokuments die aktuell angezeigten Fließtext-Position entspricht. Die fehlende Zuordnung muß während des Transformationsvorgangs zum Aufbau der Ansicht in Hash-Datenstrukturen aufgebaut werden.
- Die korrekte Erzeugung von Lesezeichen- und Geschichtseintrag-Zielpositionen innerhalb eines Dokuments konnte aus Zeitmangel leider nicht mehr vollständig implementiert werden. Das Problem bei der Erzeugung von Lesezeichen und Geschichtseinträgen ist, ein Verfahren zu finden, mit dessen Hilfe die Zielpositionen im schreibgeschützten Dokument separat von diesem gespeichert werden können. Es ist nicht möglich, ein Sprungziel mit einem sogenannten Anker im Zieldokument dauerhaft anzulegen, da das Dokument ausschließlich lesbar ist.

  In HTML-Dokumenten beispielsweise ist die Verwendung von Ankern die einzige Möglichkeit an eine bestimmte Position in einem Dokument springen zu können. Innerhalb von

HTML wird ein Anker wie folgt definiert: <A NAME="MeinAnker42">. Ein Hyperlink kann dann beispielsweise jenes Ziel durch <A HREF="#MeinAnker42"> adressieren.

Für XML sieht das W3C eine Adressierungssprache für Dokumentteile namens XPointer vor. Diese Sprache erlaubt es, beinahe beliebige Positionen bzw. Bereiche in schreibgeschützten Dokumenten zu adressieren. Da meiner Meinung nach zur Zeit des Entwurfs noch keine freie Implementierung von XPointer in Java existierte, mußte eine Alternative aufgesucht werden. Nun ist XPointer die Erweiterung einer anderen XML-Sprache genannt XPath. Da die Erweiterungen für das hier zu lösende Problem nicht benötigt werden, ist XPath in seiner Funktionalität ausreichend. Hinzu kommt, daß auch XSLT XPath verwendet und mit Xalan-J ein freier XSLT-Transformator in Java verfügbar ist. Dessen XPath-Maschinerie kann zur Auswertung solcher Adressierungsausdrücke eingesetzt werden.

Problematisch ist die Tatsache, daß ein Benutzer meist keinen Einfluß auf Änderungen am Zieldokument hat. Werden hier keine Vorkehrungen getroffen, ist der Benutzer solchen Änderungen schutzlos ausgeliefert und seine Lesezeichen bzw. Geschichtseinträge können unbrauchbar werden. Zur Behebung der Problematik soll das Ziel auf möglichst langlebige, stabile Art adressiert werden. Die Idee zur Erzeugung eines möglichst langlebigen, stabilen XPath-Ausdrucks als Zielposition basiert auf der Verwendung von ID-Attributen.

ID-Attribute definiert der XML-Standard als Attribute, die unter allen ID-Attributen innerhalb eines Dokuments einen eindeutigen Wert besitzen müssen. Sie lassen sich mit Primärschlüsseln in einer Relation eines relationalen Datenbanksystems vergleichen. Ähnlich den Primärschlüsseln, kann man davon ausgehen, daß die Werte von ID-Attributen und deren Position, d.h. das Element zu dem sie gehören, in vielen Fällen stabil gegenüber Dokumentänderungen sind. Insofern bieten diese Attribute einen guten Startpunkt, um ein Ziel zu adressieren.

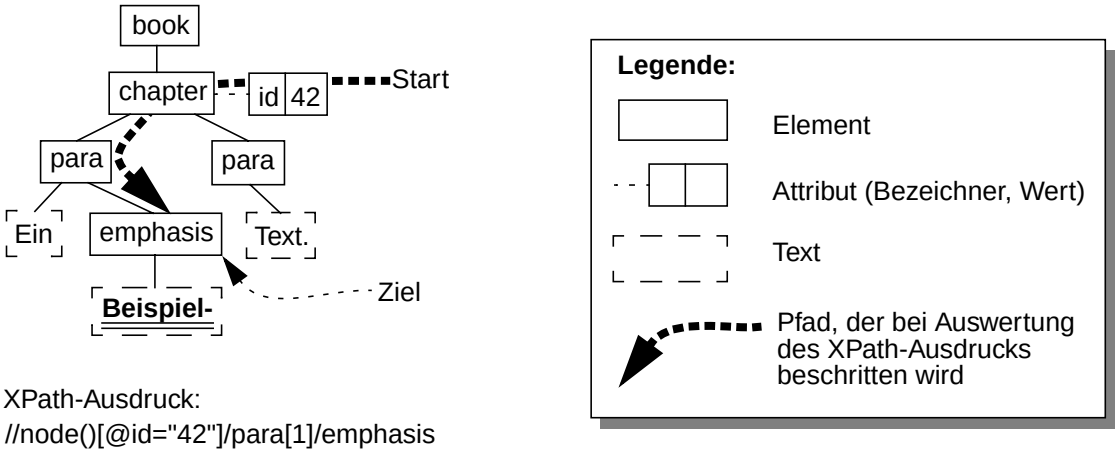
Im allgemeinen sind jedoch nicht alle Elemente eines DocBook-Dokuments mit ID-Attributen versehen. Um nun trotzdem jedes beliebige Element mit Hilfe eines XPath-Ausdrucks angeben zu können, bietet es sich an, ausgehend von einem dem Ziel naheliegenden Element mit ID-Attribut relativ bis zum tatsächlichen Zielelement durch den DOM-Baum hinabzusteigen.

Abbildung 7–1 auf Seite 67 zeigt, wie ein Zielelement ohne eigenes ID-Attribut mit Hilfe eines XPath-Ausdrucks adressiert werden kann. Der Ausdruck wird von links nach rechts ausgewertet. Er beginnt mit der Auswahl einer Menge von sämtlichen im DOM-Baum enthaltenen „//“ Knoten „nodes()“. Diese Menge wird verkleinert, indem nur der Knoten, der ein ID-Attribut „@id“ mit dem Wert „42“ besitzt, herausgefiltert wird. Dies ist ein eindeutiger Startknoten zum weiteren relativen Abstieg bis zum Ziel. Anschließend wird das erste „[1]“ Kindelement mit Namen „para“ ausgewählt. Zuletzt wird wiederum dessen einziges Kindelement mit Namen „emphasis“ ausgewählt. Der dabei übrig bleibende DOM-Knoten ist das adressierte Ziel des gesamten Ausdrucks.

- Im Zusammenhang mit der noch fehlenden Zieladressierung bei Lesezeichen und Geschichtseinträgen, konnte das Springen zu den entsprechenden Zielen ebenfalls nicht realisiert werden.

  Schwierig ist hierbei vor allem, eine möglichst intelligente Wiederverwendung der bereits geöffneten Hilfe-Dokumente und der zugehörigen Fenster. Wie in Abschnitt 6.2 „Systemkern“ auf Seite 42 beschrieben sind Lesezeichen und Geschichtseinträge global zu einer laufenden SESAMDoc-Browser-Instanz. Dies bedeutet, daß Einträge aus

**Abbildung 7–1: XPath zur persistenten Adressierung beliebiger DOM-Elemente.**




unterschiedlichsten Hilfe-Dokumenten in beliebiger Mischung in den Meta-Datenmodellen enthalten sind. Es ist nicht sinnvoll für jeden ausgewählten Eintrag einfach – falls notwendig – das zugehörige Dokument zu öffnen und die Zielposition in einem neuen Fenster anzuzeigen. Eine solche Vorgehensweise würde beispielsweise beim Zurückwandern durch die Geschichte eine Unmenge von neuen Fenstern erzeugen.

Um bereits geöffnete Dokumente wiederverwenden zu können, sieht die Spezifikation vor, daß bei Zielpositionen innerhalb solcher Dokumente deren ältestes Fenster benutzt wird, um das Ziel anzuzeigen. Dies erspart ein erneutes Öffnen des Dokuments sowie das Öffnen eines zusätzlichen Fensters zur Anzeige. Das älteste Fenster wird deshalb verwendet, damit in den meisten Fällen nicht das vom Benutzer aktuell betrachtete Fenster seinen Inhalt ändert. Eine Änderung im aktuellen Fenster beschränkt sich also auf den Fall, daß nur ein Fenster zu einem Dokument geöffnet ist.

Falls ein Dokument aus einem Lesezeichen oder Geschichtseintrag nicht bereits geöffnet ist, wird die Öffnung automatisch vorgenommen und das Ziel in dem, durch die Öffnung neu erscheinenden, Fenster angezeigt. Für alle weiteren Ziele, die dieses Dokument betreffen, bewirkt die bereits beschriebene Vorgehensweise, daß stets dessen vorhandenes Fenster zur Zielanzeige wiederverwendet wird. Im schlimmsten Fall wird jedes adressierte Hilfe-Dokument höchstens einmal geöffnet und in jeweils einem Fenster dargestellt. Einzig das Schließen nicht mehr benötigter Dokumentfenster muß der Benutzer selbst vornehmen, da das System nicht entscheiden kann, ob sie der Benutzer noch benötigt.

- Die Oberfläche zur Ausführung von Suchanfragen sowie die zugehörige Logik, die aus den Benutzereingaben einen XPath-Ausdruck als Suchausdruck generiert, konnte nicht mehr implementiert werden.

## 7.3 Nicht geplante, bereits realisierte Funktion

- Während des Probebetriebs des SESAMDoc-Browser hat sich recht schnell herausgestellt, daß die ausschließliche Auswertung von „System Identifier“ (SI) zur Quellenangabe von DTDs in XML-Dokumenten äußerst ungeschickt ist. Der Zugriff auf die DTD wird vom XML-Parser benötigt, um Hilfe-Dokumente wie auch Metadaten des Systems während des Parsings auf Gültigkeit zu überprüfen. Auf diese Weise wird von vornherein sichergestellt, daß nur Daten mit gültiger Struktur im System verarbeitet werden. Problematisch ist, daß

„System Identifier“ mit Hilfe eines URI (Uniform Resource Identifier) auf eine Ressource verweisen. Dadurch treten unweigerlich Fälle auf, bei denen auf verschiedenen Rechnersystemen, die verwiesene Ressource an unterschiedlichen Positionen gespeichert ist und somit nicht immer gefunden werden kann. Norman Walsh beschreibt diesen Mißstand im Artikel [Walsh 2000].

Zwar sieht der XML-Standard vor, neben „System Identifier“ auch sogenannte „Public Identifier“ (PI) zu benutzen, jedoch gibt es keinen Standard, der beschreibt wie der formale Namensbezeichner eines „Public Identifier“ in einen konkreten Verweis auf eine Ressource aufgelöst werden soll. Die Firma Arbortext hat nun die Implementierung der OASIS-Catalog- sowie der XML-Catalog-Funktionalität für XML-Parser in Java als OpenSource-Software veröffentlicht (siehe hierzu auch [Walsh 2000]). Man hofft, daß durch die vielfache Nutzung der Implementierung in anderen Software-Paketen die Notwendigkeit für einen Standard offensichtlich wird.

Die Katalog-Funktion besteht darin, daß auf jedem Rechnersystem ein spezifischer Katalog existiert, der „Public Identifier“ in Verweise auf tatsächlich auf *diesem* System auffindbare Ressourcen auflöst. Dies ist insofern sinnvoll, als auf den meisten Systemen bereits SGML-Kataloge existieren und gepflegt werden und diese somit für XML transparent mitbenutzt werden können. Im Prinzip sind also Kataloge nichts anderes, als eine Abstraktionsstufe zur Namensauflösung, vergleichbar mit dem Domain Name System (DNS), welches Rechnernamen in Internet-Adressen auflöst.

Dank des äußerst flexibel konfigurierbaren Standards SAX (Simple API for XML) für XML-Parser in Java ist die Integration der Katalog-Implementierung für jeden SAX-kompatiblen Parser einfach und sauber möglich. Meiner Meinung nach wird ein XML-basiertes System erst mit einer solchen Abstraktionsstufe wirklich sinnvoll benutzbar. Sämtliche Probleme mit nicht auffindbaren Ressourcen sind seit der Verwendung von Katalogen im SESAMDoc-Browser behoben, vorausgesetzt die Kataloge sind ordnungsgemäß gepflegt.

# 8 Projektinformationen

In diesem Kapitel soll ein Überblick über den Projektverlauf dieser Studienarbeit gegeben werden. Nach einer Darstellung der verschiedenen, geplanten Arbeitspakete erfolgt eine persönliche Bewertung des Projekts. Zum Abschluß möchte ich mögliche Erweiterungen für den SESAMDoc-Browser aufzeigen und einen Ausblick geben.

## 8.1 Verlauf

Die Projektdauer überspannt den Zeitraum vom 31.05.2000 bis 30.11.2000. Während der Erstellung des Projektplans zu Beginn des Projekts wurde eine Aufteilung in 10 Arbeitspakete vorgenommen. Tabelle 8–1 zeigt die Arbeitspakete inklusive eingeplanter Reservezeit in der Übersicht. Dargestellt werden geplanter sowie tatsächlicher Aufwand in Arbeitsstunden, weiterhin die geplanten sowie tatsächlichen Meilenstein-Termine. Außerdem zeigt die letzte Spalte die direkt greifbaren Ergebnisse in Form von Dokument- bzw. Folien-Seiten oder Quelltextzeilen (LOC, lines of code) der jeweiligen Arbeitspakete.

**Tabelle 8–1: Arbeitspakete und entstandene Ergebnisse**

| Arbeitspakete | Aufwand | | Meilensteine | | Ergebnisse |
|---|---|---|---|---|---|
| | Soll | Ist | Soll | Ist | |
| Projektplan | 20 | 20 | 06.06. | 05.06. | 7 Seiten |
| Systemvergleich | 40 | 60 | 20.06. | 29.06. | 19 Seiten |
| Anforderungen | 20 | 15 | 27.06. | 06.07. | 8 Seiten |
| Spezifikation | 40 | 55 | 11.07. | 20.07. | 38 Seiten |
| Entwurf | 40 | 56 | 25.07. | 31.07. | 37 Seiten |
| Implementierung | 120 | > 120 | 19.09. | 27.09. | 12800 LOC |
| Test | 40 | ~ 24 | 03.10. | 03.10. | 17 Seiten |
| Benutzerhandbuch | 20 | ~ 10 | 10.10. | 10.10. | 21 Seiten |
| Vortrag | 20 | ~ 20 | – | 30.10. | 9 Folien |
| Ausarbeitung | 120 | ~ 160 | 23.11. | 29.11. | 92 Seiten |
| Reserve | 20 | ~ 0 | – | – | – |
| Summe | 500 | > 540 | – | – | 239 Seiten<br>9 Folien<br>12800 LOC |

In Tabelle 8–2 ist der gesamte Projektverlauf anhand einer Kalenderwochen-Übersicht dargestellt. Da das Beginn-Datum des Projekts Mittwoch, der 31.05.2000 in Kalenderwoche 22 ist, reichen die eingetragenen Wochen nicht von Montag bis Sonntag, sondern von Mittwoch bis Dienstag der darauffolgenden Woche. In Woche 24 wurde in Übereinkunft mit dem Betreuer ein Teil des Entwurfs vorgezogen. Dies erschien sinnvoll, da sich der Bearbeiter einer parallel laufenden Diplomarbeit mit ähnlichem Thema zu diesem Zeitpunkt in der Entwurfsphase befand und bezüglich eines Fließtext-Anzeige-Moduls gemeinsam an einer Lösung gearbeitet

werden konnte. Dadurch verschoben sich die folgenden Arbeitspakete jeweils um eine Woche nach hinten. Schlußendlich benötigte ich für den gesamten Entwurf doch mehr Zeit als veranschlagt, so daß sich die Verzögerung um eine Woche vorerst fortsetzen sollte. Ab Woche 31 beginnt die vorlesungsfreie Zeit. Während der Projektplanung waren die Prüfungstermine noch nicht bekannt, weshalb prophylaktisch ab diesem Zeitpunkt drei Wochen für Prüfungsvorbereitung eingeplant wurden. Da die Prüfungen erst später stattfanden, habe ich ab Woche 31 parallel zur Prüfungsvorbereitung in Teilzeit die Implementierung vorgenommen. Erst in Woche 36 wurde die Implementierungsphase zugunsten einer Woche Prüfungsvorbereitung in Vollzeit unterbrochen. Nach der Prüfung habe ich die Implementierung in Vollzeit wieder aufgenommen. Um möglichst viel Funktionalität aus der Spezifikation implementieren zu können, wurde die erste Woche des Test-Zeitraums umdisponiert. Mit der Erstellung des Testplans und Durchführung der Testfälle befand sich der Projektfortgang wieder in der geplanten Zeit. Der Vortrag zur Studienarbeit fand später als geplant statt und wurde parallel zur bereits begonnen Ausarbeitung vorbereitet.

**Tabelle 8–2: Kalenderwochen-Übersicht**

| Arbeitspakete | | Kalenderwochen | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| Projekt-plan | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| System-vergleich | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Anforde-rungen | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Spezifi-kation | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Entwurf | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Implemen-tierung | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Test | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Benutzer-handbuch | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Vortrag | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Ausar-beitung | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reserve | Soll | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | Ist | | | | | | | | | | | | | | | | | | | | | | | | | | |

## 8.2 Bewertung

Meine Entscheidung zur Durchführung dieser Studienarbeit wurde beeinflußt durch mein großes Interesse an XML und DocBook. Dies rührt unter anderem daher, daß ich bereits die Ausarbeitung zu meinem Seminar über XML und später zu meinem Hauptseminar mit Hilfe von DocBook verfaßt habe. Daher waren bereits entsprechende Kenntnisse sowie Begeisterung für generische Auszeichnung mit ihren schier unendlichen Möglichkeiten vorhanden. Im Gegensatz dazu waren meine Kenntnisse bezüglich Software-Technik vor Beginn der Studienarbeit eher mager. Durch diese Arbeit konnte ich mein Wissensspektrum durch praktische Erfahrungen in Richtung der Software-Technik erweitern. Natürlich hat dieses parallele Hinzulernen von software-technischen Verfahrensweisen die Studienarbeit nicht unbedingt erleichtert. Allerdings sah ich es als Herausforderung und denke rückblickend, daß die Erfahrungen äußerst hilfreich sind und sich später sicherlich auszahlen werden.

Bezüglich den Arbeitspaketen wie Anforderungen, Spezifikation und Entwurf mußte ich feststellen, daß ich mich mit meiner kaum vorhandenen Praxis im Verfassen der dazugehörigen Dokumente teilweise schwer getan habe. Einem Student der Software-Technik wäre dies bestimmt leichter gefallen, da er die Erstellung solcher Dokumente wiederholt während des Studiums praktiziert. Gerade im Hinblick auf diese Problematik wäre es sehr hilfreich, gäbe es exemplarische Vorlagen solcher Dokumente, die Bearbeiter von Studien- und Diplomarbeit als Richtlinie verwenden könnten.

An den tatsächlich aufgelaufenen Aufwänden in Tabelle 8–1 auf Seite 69 läßt sich ablesen, daß diese im Durchschnitt über den geplanten Aufwänden liegen. Meinen bisherigen Erfahrungen in Praktika, Seminar und Hauptseminar nach zu urteilen ist dies normal und bewegt sich im Rahmen des Üblichen. Prinzipiell ist gegen Mehraufwände nichts einzuwenden. Ich denke die These, daß ein Mehr an Arbeit zu einem besseren Ergebnis führen würde, hat in gewisser Weise ihre Richtigkeit. Nun ist die Ausdehnung der Aufwände für die Studienarbeit aber nicht beliebig möglich, belegt man – wie vom Studienplan vorgesehen – parallel dazu Vorlesungen und legt Prüfungen ab. Insofern mußte ich mich stellenweise damit abfinden, Arbeitspakete abzuschließen und zum nächsten überzugehen, auch wenn ich am Ergebnis gerne noch etwas länger gearbeitet hätte.

Im Vergleich verschiedener zur Verfügung stehender Programmiersprachen und Toolkits für graphische Benutzungsoberflächen scheint mir Java die richtige Wahl gewesen zu sein. Notwendige Merkmale wie strenge Typsicherheit, Paketkonzept, ausschließlich Referenzen statt Zeigern, Schnittstellen sowie die grundlegende Plattformunabhängigkeit sind allesamt Bestandteile der Sprache Java und nicht nur mehr oder weniger gut passende Zusätze. Insgesamt halte ich Java mit der Sprachversion 2 und dem zugehörigen Toolkit Swing für relativ ausgereift und konsistent. Unter den gängigen Toolkits ist mir keines bekannt, welches das MVC-Prinzip derart konsequent umsetzt. Läßt man sich auf die in Java und Swing verwendeten Entwurfsmuster ein, kann der Entwurf und vor allem die Implementierung größtenteils sehr motivierend sein. Die vorhandenen Muster und Klassenbibliotheken ersparen zeitraubende und fehleranfällige Neuerfindungen, so daß sich der Programmierer auf die Umsetzung der Ideen konzentrieren kann und von den sich einstellenden Erfolgserlebnissen profitiert.

Auch die Wahl von DocBook in seiner XML-Ausprägung halte ich für sehr sinnvoll. Die Auszeichnungssprache ist sehr flexibel einsetzbar und ihre aktive Weiterentwicklung verspricht auch in Zukunft die Umsetzbarkeit neuer Anforderungen. Spätestens seit dem einsetzenden XML-Trend meine ich, auch anhand der Aktivitäten um DocBook – beispielsweise in den Mailinglisten[1] – vermehrtes Interesse an dieser Auszeichnungssprache festzustellen. Dies hat wohl auch zur Veröffentlichung von [Walsh et al. 1999] geführt und wird möglicherweise

durch die recht umfassende Dokumentation, die nun in Form des Buches sowie im Web erhältlich ist, im Gegenzug mehr Benutzer zum Einsatz von DocBook bewegen.

## 8.3 Mögliche Erweiterungen

Zum Abschluß möchte ich noch einige mögliche, interessante Erweiterungen für das SESAM-Doc-Browser-System vorstellen.

**Inhaltsverzeichnis**

Die Spezifikation beschreibt exakt, welche DocBook-Elemente als Gliederung zu zählen sind und deshalb im Baum des Inhaltsverzeichnisses erscheinen müssen. Es ist durchaus möglich, daß ein Benutzer Hilfe-Dokumente anzeigen lassen möchte, deren Gliederung sich nur schwer aus den festgelegten Elementen erschließen läßt. Sei es, daß die angezeigte Gliederung zu grob oder zu fein ist und somit der Überblick verloren geht. Es wäre für den Benutzer wünschenswert, könnte er die Auswahl, welche Elemente als Gliederung dargestellt werden und welche nicht, selbst vornehmen. Um eine solche Auswahl zu erleichtern, kann eine Liste der im Dokument verwendeten Elemente angeboten werden. In dieser Liste markiert der Benutzer dann die Elemente, die er im Inhaltsverzeichnis angezeigt haben möchte. Möglicherweise benutzerfreundlicher könnte die Markierung von Elementen in einer vollständigen Baumdarstellung des Hilfe-Dokuments sein. Auf diese Weise erschließen sich nämlich dem Benutzer die hierarchischen Zusammenhänge zwischen den Elementen. Die Zusammenhänge bleiben in einer Liste verborgen, denn es kann nicht vom Benutzer erwartet werden, daß er die Struktur von DocBook-Dokumenten kennt. Ganz allgemein könnte auch nur vorgegeben werden, bis zu welcher Hierarchietiefe Elemente im Inhaltsbaum angezeigt werden.

**Fließtext-Ansicht**

Die Darstellung der Hilfe-Texte am Bildschirm erfolgt in Teilen, die eine bestimmte Größe nicht überschreiten sollten. Dies bedeutet, daß ein Hilfe-Dokument in einzelne, logisch aufeinanderfolgende Teile zerlegt wird und die Fließtext-Ansicht eines dieser Teile anzeigt. Die sequentielle Navigation erfolgt mit Hilfe von Navigations-Knöpfen in der Werkzeugleiste. Diese aufgeteilte Darstellung vermittelt mehr Übersicht, da idealerweise nur die zum aktuellen Kontext passende Hilfe-Information angeboten wird. Somit wird der Benutzer nicht durch vorhergehenden und nachfolgenden Text, der bei der durchgehenden Darstellung eines Dokuments sichtbar ist, irritiert. Außerdem kommt es dem Lesen in Papierform eher näher. Dort findet die Aufteilung in Seiten statt. Es wird üblicherweise kein Text auf Endlospapier gedruckt, was einer Anzeige von fortlaufendem Fließtext entsprechen würde. Wie diese Aufteilung im System vorgenommen wird, könnte durch den Benutzer bestimmbar auf der Basis auswählbarer Gliederungsstufen erfolgen. Die verschiedenen Stufenmodelle teilen ein Dokument auf verschieden hohen Gliederungsstufen und ermöglichen so eine individuelle, zum Inhalt eines Dokuments passende Aufteilung. Nun kann aber der Umfang von Text in verschiedenen Abschnitten gleicher Hierarchiestufe sehr unterschiedlich ausfallen, so daß eine dynamische Aufteilung Sinn macht. Ein „intelligenter“ Algorithmus könnte eine Aufteilung vornehmen, dessen entstehenden Teildokumente eine optimale Größe zur Betrachtung aufweisen.

**Index**

Wie in Abschnitt 6.4.1 „Automatische Erzeugung des Index“ auf Seite 45 beschrieben, wird beim Öffnen von Hilfe-Dokumenten aus einfachen Dateien der Index automatisch vom System

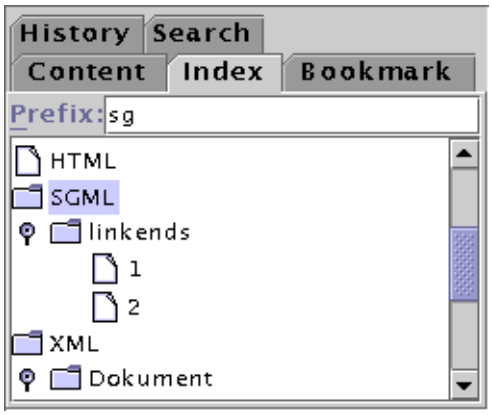
1.http://www.oasis-open.org/docbook/mailinglist/index.html

erzeugt. Je mehr Indexbegriffe im Dokument ausgezeichnet werden, desto länger kann diese Indexgenerierung dauern. Zur Lösung dieses Problems könnte die Erzeugung und Aufbereitung ausgelagert werden und beispielsweise von einer Paketverwaltung bereits zum Abschluß der Dokumenterstellung erfolgen.

Derzeit erlaubt das System eine inkrementelle Suche von Suchbegriff-Präfixen innerhalb der Menge der primären Indexbegriffe. Erweiterte Suchfunktionen im Index könnten beliebige Teilzeichenketten in allen Begriffstufen finden. Dies ist deshalb nicht in SESAMDoc-Browser umgesetzt worden, da bei einer derartig erweiterten Suche nun über den gesamten Index hinweg verstreute Treffer gefunden werden könnten. Es ist dabei schwierig dem Benutzer zu vermitteln, wieviele Treffer gefunden wurden und wo sich diese im Index befinden. Beinahe unvermeidlich ist dabei das Springen zum nächsten Suchtreffer im Indexbaum. Bei einer Vielzahl von Fundstellen kann dies sehr verwirrend sein. Zwar liese sich das Springen vermeiden, indem die Suchtreffer in einer eigenen Liste angezeigt würden, dann wäre aber der Zusammenhang zum Index verloren. Der Benutzer sieht dann keine neben dem gefundenen Begriff liegenden, anderen Begriffe mehr, in denen er vielleicht selbst schnell weitere passende Begriffe entdecken kann.

Wird ein Indexbegriff mehrmals im Dokument als solcher ausgezeichnet, bedeutet dies im Index, daß diesem Begriff mehrere Stellen im Dokument zugeordnet sind. Um eine mehrfache Darstellung desselben Begriffs im Indexbaum zu vermeiden und gleichzeitig den Sprung zu einem Ziel aus mehreren möglichen zu erlauben, versammelt die derzeitige Implementierung der Index-Ansicht sämtliche Sprungziele als Blattknoten unterhalb eines „Sprungziele“-Knotens, welcher wiederum ein Kind des Knotens für den Indexbegriff selbst ist. Abbildung 8–1 zeigt einen Bildschirmabzug eines Index, dessen Eintrag „SGML“ zwei Sprungziele „1“ und „2“ unterhalb des „Sprungziele“-Knotens (linkends) enthält. Indexeinträge der Form „Begriff1

**Abbildung 8–1: Mehrfache Sprungziele eines Indexeintrags**




siehe Begriff2“ bzw. „Begriff1 siehe auch Begriff2“ sind dadurch auf derselben Hierarchie-Ebene wie der „Sprungziele“-Knoten, d.h. als Kinder eines Begriffs, realisierbar ohne mit den Blattknoten für Sprungziele vermischt zu werden. Die Vermischung wird verhindert, da Einträge der Form „siehe“ und „siehe auch“ kein Sprungziel im Text des Hilfe-Dokuments besitzen, sondern einfach nur informell auf einen anderen Indexeintrag verweisen. Prinzipiell wäre es möglich diesen informellen Verweis automatisch in eine Art Sprungziel zu wandeln, so daß bei einer Auswahl der verwiesene Begriff im Indexbaum markiert wird. Es spricht nichts dagegen, auch eine vollkommen anders geartete, als die hier beschriebene Darstellung des Index vorzunehmen.

### Glossar

Die Assoziation zwischen Glossarbegriffen im Dokumenttext und den zugehörigen Glossareinträgen mit Begriffsdefinition läßt sich sowohl implizit als auch explizit vornehmen. Die Spezifikation des SESAMDoc-Browsers sieht bislang eine Unterstützung expliziter Hyperlinks vor. Diese Links sind im Dokument realisiert durch ein ID-Attribut mit eindeutigem Wert am Element der Zielposition, d.h. der Begriffsdefinition, sowie einem „linkend“-Attribut am Element der Begriffsverwendung im Text. Dabei trägt das „linkend“-Attribut den Wert des Ziel-ID-Attributs. Auf diese Weise ergibt sich ein einfacher, einseitiger Hyperlink-Mechanismus, der zur Standardfunktionalität von XML und somit auch DocBook gehört.

Allein durch das Vorhandensein der Auszeichnung einer Begriffsverwendung und der dazugehörigen Definition läßt sich durch einen Algorithmus eine Zuordnung automatisch vornehmen. Dadurch würden Verweise realisiert, die implizit durch das Gleichlauten von Begriffen im Text sowie im Glossar vorhanden sind. Dabei stellt sich mir allerdings die Frage, ob es nicht Aufgabe der Dokument-Autoren oder einer Paketverwaltung wäre, mit einem Werkzeug, das zur Unterstützung beispielsweise über den eben beschriebenen Algorithmus verfügt, explizite Verweise in ein Dokument einzubringen.

Unabhängig davon könnte der Browser auf Wunsch des Benutzers eine implizite Zuordnung für jede beliebige – auch nicht-ausgezeichnete – Verwendung eines im Glossar definierten Begriffs vornehmen. Diese Idee entspricht der Vorgehensweise der früheren Studienarbeit [Joos 1997].

Anstelle bzw. zusätzlich zur Verwendung von Verweisen wie eben beschrieben könnte die Definition eines Begriffs mit Hilfe eines Tooltips angezeigt werden. Um eine evtl. vorhandene Definition anzuzeigen, positioniert der Benutzer einfach den Mauszeiger über dem gewünschten Wort in der Fließtext-Ansicht. Eine erkennbare, spezielle Formatierung von Wörtern im Text, für die ein Glossareintrag existiert, sollte dafür sorgen, daß der Benutzer im Voraus weiß, ob er überhaupt eine Begriffsdefinition zu erwarten hat.

### Zweiseitige Hyperlinks

Explizit in einem Hilfe-Dokument vorhandenen, einseitigen Hyperlinks könnte durch Aufbau entsprechender, interner Datenstrukturen, eine zweiseitige Funktionalität verliehen werden. Dadurch würde für jeden Verweis automatisch die Verweisverfolgung in die andere als die vorgesehene Richtung ermöglicht. Dabei muß allerdings beachtet werden, daß hierdurch mehrfache Verweise entstehen können. Sobald mehrere einseitige Links auf dieselbe Position verweisen, existieren für die umgekehrte Verweisrichtung mehrere Zielmöglichkeiten. Hier muß dem Benutzer dann die Auswahl eines Ziels ermöglicht werden, beispielsweise durch ein Kontextmenü oder einen eigens eingeblendeten Auswahldialog.

### Zusammenführung von Hilfepaketen

In Abschnitt 4.4.1 auf Seite 28 wird die Anforderung formuliert, daß mehrere Hilfepakete automatisch zusammengeführt werden sollten, um die Suche in Dokumenten über mehrere Pakete gleichzeitig ausführen zu können. Hierbei stellt sich neben der technischen Umsetzung einer solchen Funktionalität auch die Frage, wie die Benutzungsschnittstelle dafür aussehen könnte. Bietet man dem Benutzer die Möglichkeit verschiedenste Hilfepakete in einem konfigurierbaren Inhaltsbaum einhängen zu können, kann er sich seine eigene Hilfe-Umgebung aufbauen. Dies wäre vergleichbar mit dem Einhängen verschiedener Dateisysteme („mount“) in ein Wurzeldateisystem unter Unix-Betriebssystemen.

Gerade im Hinblick auf die Zusammenführung mehrere Hilfepakete ergibt sich früher oder später die Problematik, daß Hilfe-Dokumente zu groß werden, um als DOM-Datenstruktur vollständig im Hauptspeicher gehalten werden zu können. Dies passiert vor allem auch dann, wenn Dokument-Autoren mehrere Bücher und Artikel zu einem großen DocBook-Set zusammenfassen, weil beispielsweise eine automatische Zusammenführung noch gar nicht existiert. Die Frage ist, ob sich eine Lösung finden läßt, die möglichst wenig Veränderungen am bestehenden SESAMDoc-Browser-System erfordert und somit eine idealerweise komplette Wiederverwendung ermöglicht.

Große Chancen bestehen meiner Meinung nach in einer Lösung, die eine andere Implementierung der DOM-Schnittstelle einsetzt. Da nach wie vor dieselbe Schnittstelle verwendet wird, sollten am System kaum Änderungen notwendig sein. Stattdessen übernimmt die neue DOM-Implementierung die Aufgabe, nur die aktuell benötigten Dokument-Fragmente in einer Art Cache im Hauptspeicher zu halten und bei Bedarf gegen andere nachzuladende Fragmente auszutauschen. Das Nachladen, könnte beispielsweise aus einer Paketverwaltung oder einem zentralen Dokumentrepositorium in einer Datenbank geschehen. Dabei kann sogar auf das Parsing von Dokumenten verzichtet werden, wenn diese bereits in einer Baumform in den Verwaltungssystemen vorliegen.

Ein nicht zu unterschätzendes Problem bei einer solchen Lösung stellt die Hauptspeicherverwaltung dar. In Java übernimmt dies die Laufzeitumgebung, die über einen Garbage Collector verfügt. Bis zur JDK Version 1.2.x handelt es sich um einen Mark-and-Sweep Garbage Collector, der unbenutzte zyklische Datenstrukturen nicht erkennen und aufräumen kann. Gerade bei DOM-Implementierungen existieren aber solche Zyklen durch die beidseitige Verknüpfung von Vater- und Kind-Knoten. In der Cache-Verwaltung der oben beschriebenen DOM-Implementierung muß im Sinne der Speicherausnutzung explizit darauf geachtet werden, daß nicht mehr benutzte Dokument-Fragmente tatsächlich freigegeben werden. Dies läßt sich durch Aufbrechen der Zyklen, indem die entsprechenden Objektreferenzen zu „null“ gesetzt werden, sicherstellen.

Bei Verwendung einer DOM-Implementierung, die auf einer Datenbank operiert, ist es nicht mehr sinnvoll, Suchanfragen mit Hilfe von XPath durchzuführen. Die Natur der XPath-Ausdrücke besteht in der Bildung und Filterung von Knotenmengen aus einem XML-Dokument. Dabei ist es nicht ungewöhnlich, daß während der Ausdrucksauswertung kurzzeitig große Knotenmengen entstehen. Statt nun eine XPath-Maschinerie im Hauptspeicher des Browser-Clients Ausrücke auswerten zu lassen, bietet es sich an, Suchanfragen effizient mittels Suchfunktionen der Datenbank durchzuführen.

Grundlegende Basisdienste für die Verwendung beliebiger DOM-Implementierungen, die auf verschiedenste Arten ihre Datenhaltung übernehmen, existierten bereits in einer früheren Version des OpenXML-Projekts[1]. In der derzeit aktuellen Version scheinen diese Ansätze aber wieder verschwunden zu sein.

**Suchanfrage-Typen**

Zum Abschluß der Erweiterungsmöglichkeiten möchte ich noch auf die vordefinierten Suchanfrage-Typen für die strukturierte Suche eingehen. Derzeit sieht der Entwurf vor, daß die Anfragetypen direkt in Form verschiedener, fest codierter Java-Klassen umgesetzt werden. Im Sinne einer optimaleren Suchfunktionalität für den Benutzer wäre es sinnvoll, Suchanfrage-Typen flexibel konfigurierbar zu machen. In Anlehnung an das Online-Hilfesystem DynaText (siehe Abschnitt 3.3.1 „DynaText“ auf Seite 9) könnten Anfragetypen in XML definiert werden. Sol-

1. http://www.openxml.org/

che Definitionen ließen sich dann auf die spezifischen Strukturmerkmale von Hilfe-Dokumenten zuschneiden und innerhalb von Hilfepaketen zusammen mit den Dokumenten ausliefern. Zur Definition eines Suchanfrage-Typs sollte es ausreichend sein, mit Hilfe von XML-Elementen und Attributen die benötigten Eingabe-Komponenten sowie Regeln festzulegen, die beschreiben wie aus den Benutzereingaben ein Suchanfrageausdruck generiert wird. Aus der Komponenten-Beschreibung kann das Browser-System dynamisch die entsprechenden Swing-Komponenten am Bildschirm anzeigen und unter Anwendung der Bildungsregeln aus den Eingaben eine Suchanfrage erstellen.

## 8.4 Ausblick

Abschließend stelle ich fest, daß mein Interesse für die Software-Technik trotz – oder gerade wegen – der Mühen, die sie dem Ungeübten bereiten kann, durch diese Studienarbeit gewachsen ist. Der Gedanke, die Erstellung von Software in naher Zukunft mit ingenieursmäßigen Mitteln durchführen zu können, weckt bei mir die Hoffnung, daß auch die damit verbundenen Schwierigkeiten besser beherrschbar werden und somit bessere Software entstehen wird. Bis es soweit sein wird, sollte man nicht auf die Erleuchtung warten, sondern aktiv darauf hinarbeiten. Hilfreich wird hierbei hoffentlich der Einsatz von DocBook-XML zur Erstellung der anfallenden Dokumentation sein. Und so hoffe ich, daß der in dieser Studienarbeit entstandene SESAMDoc-Browser einen Grundstein legt, auf dem die Abteilung Software Engineering und andere Interessierte aufbauen können.

# Anhang A Schnittstellen der JavaBeans

Dieser Anhang enthält die Programmierschnittstellen der JavaBeans aus Ebene 1 der graphischen Benutzungsoberfläche des SESAMDoc-Browser.

Da die relevanten Methoden und evtl. vorhandene Klassenvariablen nicht in eigenen Interfaces deklariert sind und Java keine explizite Trennung von Deklaration und Implementierung kennt, sind die JavaBean-Klassen in Pseudo-Java notiert. Hierzu wurden einfach die Initialisierungen für Klassenvariablen sowie die Implementierungskörper der Methoden weggelassen.

## A.1 Schnittstelle der Dokument-Ansicht-JavaBeans

Es folgt die Schnittstelle der spezialisierten Dokument-Ansichten. Stellvertretend für die vier verschiedenen Ansichten ist die JavaBean, die für den formatierten Fließtext zuständig ist, angegeben. Die Schnittstelle der drei übrigen Beans ist identisch und läßt sich durch einfaches Austauschen der Zeichenkette „Styledtext“ durch

- „Content“ für die Inhaltsverzeichnis-Bean,
- „Index“ für die Index-Bean oder
- „Search“ für die Suchanfrage-Bean

aus der folgenden ableiten.

```
// Paket, in dem die folgende Klasse definiert wird
package sesamdoc.browser.gui.layer1.styledtext;

// verwendete Klassen und Schnittstellen aus anderen Paketen
import sesamdoc.browser.HelpDocument;
import org.w3c.dom.Node;
import java.beans.PropertyChangeListener;
import sesamdoc.browser.gui.event.LayerListener;

// Klasse ist abgeleitet von Swing-Layout-Kontainer (JavaBean)
public class Styledtext extends javax.swing.JPanel {

  // Name der gebundenen Eigenschaft markedNode
  public final static String MARKED_NODE_PROPERTY;

  // Standard-Konstruktor für JavaBean
  public Styledtext()
  // Konstruktor setzt zusätzlich gleich das Datenmodell
  public Styledtext(HelpDocument helpDoc)

  // eine Art expliziter Destruktor (kein Java-Standard)
  public void closeStyledtext()

  // Datenmodell-Eigenschaft
  public HelpDocument getHelpDocument()
  public void setHelpDocument(HelpDocument helpDoc)

  // gebundene Eigenschaft markedNode
  public Node getMarkedNode()
  public void setMarkedNode(Node node)

  // Registrierung für gebundene Eigenschaften der Klasse
```

```
    public void addPropertyChangeListener(
                  PropertyChangeListener listener)
    public void removePropertyChangeListener(
                  PropertyChangeListener listener)

    // Registrierung für Ereignisse zwischen Ebenen der Oberfläche
    public void addLayerListener(LayerListener ll)
    public void removeLayerListener(LayerListener ll)
  }
```

## A.2 Schnittstelle der Metadaten-JavaBeans

Es folgt die Schnittstelle der Metadaten-Ansichten. Stellvertretend für die drei verschiedenen Ansichten ist die JavaBean, die für die Darstellung der Lesezeichen zuständig ist, angegeben. Die Schnittstelle der zwei übrigen Beans ist identisch und läßt sich durch einfaches Austauschen der Zeichenkette „Bookmark“ durch

- „History“ für die Betrachtungs-Geschichte-Bean oder
- „SearchHistory“ für die Suchanfragen-Geschichte-Bean

aus der folgenden ableiten.

```
  // Paket, in dem die folgende Klasse definiert wird
  package sesamdoc.browser.gui.layer1.bookmark;

  // verwendete Klassen und Schnittstellen aus anderen Paketen
  import sesamdoc.browser.config.BookmarkEntry;
  import sesamdoc.browser.config.BookmarkListener;
  import sesamdoc.browser.config.BookmarkEvent;
  import sesamdoc.browser.gui.event.LayerListener;
  import java.beans.PropertyChangeListener;

  // Klasse ist abgeleitet von Swing-Layout-Kontainer (JavaBean)
  // und implementiert BookmarkListener-Schnittstelle
  public class Bookmark extends javax.swing.JPanel
      implements BookmarkListener {

    // Name der gebundenen Eigenschaft markedEntry
    public static final String MARKED_ENTRY_PROPERTY;

    // Standard-Konstruktor für JavaBean
    public Bookmark()
    // Konstruktor setzt zusätzlich gleich das Datenmodell
    public Bookmark(sesamdoc.browser.config.Bookmark bookmark)

    // eine Art expliziter Destruktor (kein Java-Standard)
    public void closeBookmark()

    // Datenmodell-Eigenschaft
    public sesamdoc.browser.config.Bookmark getBookmark()
    public void setBookmark(
                  sesamdoc.browser.config.Bookmark bookmark)

    // gebundene Eigenschaft markedEntry
    public BookmarkEntry getMarkedEntry()
```

```
    public void setMarkedEntry(BookmarkEntry entry)

    // Registrierung für gebundene Eigenschaften der Klasse
    public void addPropertyChangeListener(
                   PropertyChangeListener listener)
    public void removePropertyChangeListener(
                   PropertyChangeListener listener)

    // Registrierung für Ereignisse zwischen Ebenen der Oberfläche
    public void addLayerListener(LayerListener ll)
    public void removeLayerListener(LayerListener ll)

    // Update-Methode aus der BookmarkListener-Schnittstelle
    // wird gerufen bei Zustandsänderung im Bookmark-Datenmodell
    public void bookmarkEvent(BookmarkEvent be)
  }
```

# 9 Literatur

[Apparao et al. 1998] Apparao, V.; Byrne, S.; Champion, M.; Isaacs, S.; Jacobs, I.; Le Hors, A.; Nicol, G.; Robie, J.; Sutor, R.; Wilson, C.; Wood, L.: **Document Object Model (DOM) Level 1 Specification**. W3C Recommendation 01.10.1998, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001

[Bray et al. 1998] Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.: **Extensible Markup Language (XML) 1.0**. W3C Recommendation 10.02.1998, http://www.w3.org/TR/1998/REC-xml-19980210
Deutsche Übersetzung: Mintert, S.; Behme, H.: Extensible Markup Language (XML) 1.0, http://www.mintert.com/xml/trans/REC-xml-19980210-de.html

[Clark et al. 1999] Clark, J.; DeRose, S.: **XML Path Language (XPath) Version 1.0**. W3C Recommendation 16.11.1999, http://www.w3.org/TR/1999/REC-xpath-19991116

[Drappa et al. 1995] Drappa, A.; Deininger, M.; Ludewig, J.; Melchisedech, R.: **Modeling and Simulation of Software Projects**, Proceedings of the Twentieth Annual Software Engineering Workshop. 29.-30. November 1995, Greenbelt, MD (USA), Seite 269-275, 1995.

[Eckstein et al. 1998] Eckstein, R.; Loy, M.; Wood. D.: **Java Swing**. O'Reilly & Associates, 1998

[Friendly et al. 1999] Friendly, L.; Campione, M.; Walrath, K.; Huml, A.: **The Java Tutorial**: A practical guide for programmers. 1999, http://java.sun.com/docs/books/tutorial/

[Gutentag et al. 1997] Gutentag, E.; Suttor, J.: **The Evolution of Sun's Answerbooks, A Case Study**: Report From the Trenches. 1997, http://www.oasis-open.org/cover/bib-fh.html#gutentagSGMLEU97 bzw. http://www.oasis-open.org/cover/gutentag97-tar.gz

[Herczeg 1994] Herczeg M.: **Software-Ergonomie**: Grundlagen der Mensch-Computer-Kommunikation, Addison-Wesley, 1994

[Joos 1997] Joos, M.: **Verknüpfung der SESAM-Dokumente mittels Hypertext**. Universität Stuttgart, Fakultät Informatik, Studienarbeit Nr. 1601, 1997

[Morrison 1997] Morrison, M.: **JavaBeans**: Technik, Konzepte, Beispiele. Markt und Technik, 1997

[Niemayer et al. 1997] Niemeyer, P.; Peck, J.; Deutsche Übersetzung von Merz, T. und Karsunke K.: **Java**: Expedition ins Programmierreich. O'Reilly / International Thomson Verlag, Bonn, 1997

[Prinzing 2000] Prinzing, T.: **Using the Swing Text Package**: It's Author Explains Exactly How It's Designed. Sun Microsystems, 2000, http://java.sun.com/products/jfc/tsc/articles/text/overview/

[Violet-1 2000] Violet, S.: **How The Swing Text Package Handles Attributes**. Sun Microsystems, 2000, http://java.sun.com/products/jfc/tsc/articles/text/attributes/

[Violet-2 2000] Violet, S.: **The Element Interface**: Use It To Model Text in Swing Applications. Sun Microsystems, 2000, http://java.sun.com/products/jfc/tsc/articles/text/element_interface/

[Violet-3 2000] Violet, S.: **Understanding the ElementBuffer**: A Closer Look at Modeling Text in Documents. Sun Microsystems, 2000, http://java.sun.com/products/jfc/tsc/articles/text/element_buffer/

[Walsh et al. 1999] Walsh, N.; Muellner, L.: **DocBook**: The Definitive Guide. O'Reilly & Associates, Sebastopol, 1999. (Auch im WWW verfügbar unter: http://www.docbook.org)

[Walsh 2000] Walsh, N.: **Standard Deviations from Norm**: If You Can Name It, You Can Clame It!. Think Tank Artikel von Arbortext, 2000, http://www.arbortext.com/Think_Tank/Norm_s_Column/Issue_three/issue_three.html

[Wilhelms et al. 1999] Wilhelms, G.; Kopp, M.: **Java professionell**. MITP-Verlag, Bonn, 1999

# 10 Glossar

**Anker:** Zielposition in einem HTML-Dokument, die durch ein sogenanntes anchor-Element festgelegt wird. Das Element enthält hierzu ein name-Attribut, dessen Wert einen unter allen Ankern im Dokument eindeutigen Bezeichner enthalten muß.
Beispiel: `<A NAME="foobar">Dokument-Textteil zum Thema foobar</A>`.

**Anmerkung:** Benutzerdefinierter Text, der mit einer bestimmten Position in einem Hilfe-Dokument assoziiert ist.

**Applet:** Kleines Java-Programm, das innerhalb einer Sicherheitsumgebung – der sogenannten Sandbox – als aktiver Teil einer Web-Seite in einem Browser ausgeführt wird.

**Attribut:** Paar aus Bezeichner und Wert, das ein SGML-/XML-Element parametrisiert.

**Auswahl-Box:** combo box. Graphische Komponente zur Auswahl eines Eintrags aus einer vorgegebenen Menge von Einträgen. Im normalen Zustand zeigt eine Auswahl-Box nur den ausgewählten Eintrag. Bei der Aktivierung der Auswahl, z.B. durch Mausklick des Benutzers, wird eine Liste der vorgegebenen Einträge eingeblendet.

**Betrachtungs-Geschichte:** Siehe Geschichte.

**Corba:** Common Object Request Broker Architecture. Offener Industriestandard für verteiltes objektorientiertes Programmieren.

**DocBook:** XML- bzw. SGML-DTD zur Erstellung strukturell ausgezeichneter technischer Dokumentation. Standardisiert von OASIS (Organization for the Advancement of Structured Information Standards). Siehe auch `http://www.oasis-open.org/docbook`.

**Dokument:** Im Zusammenhang mit SGML/XML handelt es sich um ein Datenobjekt, das der vorgeschriebenen Syntax genügt.

**Dokument-Autor:** Autor, der ein Hilfe-Dokument meist mit Hilfe eines Autorenwerkzeugs erstellt und aufbereitet, so daß das Dokument in einem Hilfesystem eingesetzt werden kann.

**DOM:** Document Object Model. Vom W3C standardisierte, programmiersprachen-neutrale Schnittstelle zum Zugriff auf ein bereits geparstes, im Speicher befindliches XML-Dokument. Der Zugriff erfolgt analog zu Datenstrukturen in Baumform. Siehe auch [Apparao et al. 1998]

**DSSSL:** Document Style Semantics and Specification Language (ISO/IEC 10179:1996). Standardisierte Sprache zur Transformation und Formatierung von ursprünglich SGML-Dokumenten, die aber auch für XML-Dokumente funktioniert.

**DTD:** Document Type Definition. Eine Dokumenttyp-Definition spezifiziert, wie die Struktur eines SGML-/XML-Dokuments aufgebaut sein muß. Sie legt also eine konkrete Auszeichnungssprache fest. XML-DTDs werden durch einen Kellerautomaten modelliert.

**Element:** Struktureller Baustein eines SGML- bzw. XML-Dokuments. Besteht entweder aus Start-Tag, Inhalt und End-Tag oder aus einem speziellen Start-Tag, falls es sich um ein leeres Element ohne Inhalt handelt. Inhalt setzt sich aus einem Inhaltsmodell, bestehend aus Elementen, Text oder einer Mischung von beidem, zusammen.

**Feststellknopf:** toggle button. Knopf, der bei Aktivierung durch den Benutzer zwischen zwei Zuständen – gedrückt und nicht-gedrückt – wechselt. Im nicht-gedrückten Zustand wird er meist wie ein normaler Knopf dargestellt, im gedrückten Zustand wie ein dauerhaft eingedrückter Knopf.

**Geschichte:** In der Geschichte des Systems werden Positionen in Hilfe-Dokumenten, die der Benutzer liest, aufgezeichnet. Dadurch entsteht ein Protokoll, zu welcher Zeit bestimmte Positionen einer Hilfe gelesen wurden. Auf diese Weise kann dem Benutzer ermöglicht werden, jederzeit wieder an besuchte Stellen zurückzuspringen.

**Gliederungspunkt:** Bestimmte DocBook-Elemente, wie sie beispielsweise für Büchersammlung, Buch, Kapitel oder Abschnitt existieren, repräsentieren im Dokument ein Gerüst logischer Dokumentteile. Jedes dieser Elemente kann als Gliederungspunkt angesehen werden.

**GNU GPL:** GNU General Public License. Software-Lizenzmodell der FSF (Free Software Foundation), das die freie Verfügbarkeit von Software inklusive ihres Quelltextes schützt. Siehe auch `http://www.gnu.org/copyleft/gpl.html`.

**Hilfesystem:** „Unterstützungskomponente zur Erklärung der Konzepte und der Funktionalität des Anwendungssystems und der Bedieneroberfläche“ [Herczeg 1994].

**Historie:** Siehe Geschichte.

**Hyperlink:** Verweis, der eine oder evtl. mehrere bestimmte Ressourcen referenziert. Wird von Hypermediasystemen meist in einer bestimmten Formatierung dargestellt und eine Aktivierung durch den Benutzer – z.B. durch einen Mausklick – verfolgt den Verweis und bringt somit die Zielressource zur Darstellung.

**Hypermediasysteme:** „Verfahren zur computerbasierten Organisation von Informationen in einem vernetzten Informationsraum. Die Darstellungsformen in den Informationsknoten des Netzes können aus Text, Graphik, Standbildern, Bewegtbildern und akustischen Elementen bestehen“ [Herczeg 1994].

**Hypertextsysteme:** „*Hypermediasysteme* mit ausschließlich textuellen Informationsknoten“ [Herczeg 1994].

**Interface:** Objektorientiertes Sprachkonstrukt in Java, das einer Schnittstelle entspricht. Definiert eine Menge von Konstanten und Methoden mit deren Signatur jedoch ohne Implementierung.

**JAR:** Java Archive. Paketformat, das primär zur Zusammenfassung und Übertragung von Java-Programmen genutzt wird. Darin enthalten sind meist die zum Programm gehörigen Klassen in Form von übersetztem, ausführbarem Bytecode, Hilfsdateien wie beispielsweise Sinnbildern für die graphische Benutzungsoberfläche, eventuell

elektronische Unterschriften (Signaturen), aber auch beliebige andere Dateien, die mit einem Java-Programm ausgeliefert werden sollen.

**Java:** Objektorientierte, streng typisierte, plattformunabhängige Programmiersprache. Java-Programme werden vor der Ausführung in Bytecode übersetzt. Dieser Bytecode wird dann in einer Softwaremaschine – der sogenannten JVM (Java Virtual Machine) –, die ein virtuelles System simuliert, durch Interpretation ausgeführt.

**JavaBean:** Wiederverwendbare Komponente in Java, die durch ihre externe Schnittstelle bestehend aus Eigenschaften und Methoden charakterisiert ist.

**JDK:** Java Development Kit. Sammlung von Hilfsprogrammen (u.a. Übersetzer, Debugger) und Klassenbibliotheken zur Entwicklung von Java-Programmen.

**Knopf:** button. Graphische Komponente mit welcher der Benutzer ein bestimmtes Ereignis durch Aktivierung – beispielsweise ein Mausklick – auslösen kann. Meist enthält der Knopf ein Sinnbild oder eine Beschriftung bzw. beides.

**Komponente:** Im Zusammenhang mit graphischen Benutzungsoberflächen ist eine Komponente ein rechteckiger Bereich, in dem eine graphische Darstellung zum Zwecke der Benutzerinteraktion erfolgt. Standard-Komponenten sind z.B. Knöpfe oder Menüs.
Im Zusammenhang mit Wiederverwendbarkeit ist eine Komponente ein – nicht notwendigerweise graphischer – wiederverwendbarer Programmteil, der bestimmte Dienste zur Verfügung stellt. In Java werden wiederverwendbare Komponenten JavaBeans genannt.

**Leerraum:** Beliebige Folge von Leerzeichen, Tabulator und Zeilenumbruch, die in SGML- bzw. XML-Dokumenten zunächst ohne Bedeutung ist und nur zur besseren Lesbarkeit des Dokument-Quelltextes dient. Nur innerhalb von ausgewählten Elementen ist Leerraum signifikant für die Darstellung des Dokuments. Dies ist beispielsweise der Fall bei der Wiedergabe von Programm-Quelltexten.

**Lesezeichen:** Dienen dem Benutzer eines Online-Hilfesystems zur benutzerdefinierten, permanenten Speicherung von Positionen in Hilfe-Dokumenten. Mit Hilfe von Lesezeichen kann der Benutzer jederzeit unmittelbar zu den abgespeicherten Zielpositionen springen.

**MIME:** Multipurpose Internet Mail Extensions (RFC 1521). Mechanismus zur Spezifikation und Beschreibung des Formats, das im Inhalt von Internet-Nachrichten verwendet wird.

**MVC:** Model View Controller. Entwurfmuster, dessen drei Teile Modell (model), Ansicht (view) und Steuerung (controller) genau definierte Aufgaben übernehmen. Das Modell enthält Daten und meldet Veränderungen seines Zustands. Die Ansicht stellt die Daten aus dem Modell dar. Die Steuerung reagiert auf Benutzerinteraktion in der graphischen Darstellung der Ansicht und verändert daraufhin den Zustand der Daten im Modell.

**OpenSource-Software (OSS):** Software, deren Quelltext frei zugänglich ist. Siehe auch `http://www.opensource.org`.

**PDF:** Portable Document Format. Von Adobe entwickelte Seitenbeschreibungssprache zur layout-treuen elektronischen Publikation.

**SAX:** Simple API for XML. Von David Megginson entwickelte Programmierschnittstelle zum Zugriff auf XML-Parser in Java. SAX spezifiziert eine Reihe von Callback-Methoden, die während des Parse-Vorgangs gerufen werden und so eine ereignisgesteuerte Verarbeitung von XML-Dokumenten ermöglichen.

**Scrolling:** Das Verschieben des sichtbaren Ausschnittes aus einer Menge, die zu groß ist um komplett dargestellt zu werden. Meist wird das Scrolling ausgelöst durch das Verschieben der Eingabemarkierung in einem Textfeld oder Betätigen von Scroll-Leisten mit der Maus oder Tastatur.

**SGML:** Standard Generalized Markup Language (ISO-Standard 8879:1986). Erweiterbare Metasprache zur Definition von Auszeichnungssprachen. Die definierbaren Auszeichnungssprachen dienen zur strukturellen, semantischen Auszeichnung von Inhalt. Sie verwenden ein textbasiertes Format.

**Sinnbild:** icon. Graphische Komponente, repräsentiert durch ein rechteckiges Bild, mit dem der Benutzer eine bestimmte Funktion assoziiert.

**Stylesheet:** Formatierungsanweisungen, die in einer formalen Sprache definiert werden. Unter Verwendung eines Stylesheets transformiert – d.h. Veränderung der Struktur und evtl. des Ausgabeformats – ein Stylesheet-Prozessor ein Dokument einer Auszeichnungssprache wie beispielsweise XML oder SGML und weist Formatierung zu, um das Dokument in einer bestimmten Form darzustellen.

**Suchanfragen-Geschichte:** Zeichnet Suchanfragen, die der Benutzer im System ausgeführt hat, auf. Mit dem entstehenden Protokoll wird dem Benutzer ermöglicht, zu einem späteren Zeitpunkt nocheinmal dieselbe oder eine abgewandelte Suche durchzuführen.

**Toolkit:** Eine Bibliothek, die eine Sammlung von Komponenten zum Aufbau graphischer Benutzungsoberflächen enthält.

**Tooltip:** auch Bubble Help. Kurze Beschreibung zu einer graphischen Komponente, die in Form einer Beschriftung nach einer bestimmten Verweildauer des Mauszeigers über der Komponente eingeblendet wird.

**URI:** Uniform Resource Identifier (RFC 2396). Internet-Standard, der allgemein die Syntax zur Spezifikation von Verweisen wie beispielsweise von URLs beschreibt.

**URL:** Uniform Resource Locator (RFC 1738). Internet-Standard, mit dem im WWW Verweise auf Ressourcen bzw. auf Positionen in diesen Ressourcen spezifiziert werden können.
Beispiel: `http://www.w3.org/Addressing/`

**W3C:** World Wide Web Consortium. Konsortium, das den Betrieb und die Weiterentwicklung des WWW sicherstellt. Empfehlungen für Standards, sogenannte „Recommendations“, werden ausgearbeitet und dienen zur Wahrung der Interoperabilität. Siehe auch `http://www.w3.org/`

**Werkzeugleiste:** toolbar. Graphische Komponente, die eine Ansammlung von Knöpfen bestimmter Funktion beinhaltet und einer Menüleiste vergleichbar einem Fenster zuge-

ordnet ist. Sie dient dem Benutzer als Alternative zu Menüauswahlen zum schnellen Zugang zu oft benötigten Programmfunktionen.

**WWW:** World Wide Web. Teil des Internet, der auf dem Protokoll HTTP (hypertext transfer protocol) und vorwiegend der Dokumentenbeschreibungssprache HTML (hypertext markup language) basiert.

**WYSIWYG:** What You See Is What You Get. Prinzip zur Erstellung von Dokumenten, bei dem die Darstellung während der Bearbeitung am Bildschirm bereits weitgehend der Darstellung auf dem endgültigen Ausgabemedium entspricht.

**Xalan-J:** In Java geschriebener XSLT-Prozessor aus dem XML-Apache-Projekt. Xalan-J verwendet standardmäßig als Parser Xerces-J, arbeitet aber im Prinzip mit jedem SAX-kompatiblen XML-Parser in Java zusammen. Siehe auch `http://xml.apache.org/xalan/`

**Xerces-J:** In Java geschriebener XML-Parser aus dem XML-Apache-Projekt. Xerces-J unterstützt unter anderem die folgenden XML-Standards: XML 1.0 [Bray et al. 1998], XML Namespaces, DOM 1 [Apparao et al. 1998] und 2, SAX und SAX2. Siehe auch `http://xml.apache.org/xerces-j/`

**XLink:** XML Linking Language. Vom W3C standardisierte Sprache in XML-Syntax zur Definition von Verweisen zwischen Ressourcen. Neben den von HTML bekannten einfachen, einseitigen Links sind fortgeschrittene Verweisformen möglich. Siehe auch `http://www.w3.org/TR/xlink/`

**XML:** Extensible Markup Language. Vom W3C standardisierte erweiterbare Metasprache zur Definition von Auszeichnungssprachen. Es handelt sich um ein SGML-Anwendungsprofil, d.h. eine echte Untermenge von SGML. Siehe auch [Bray et al. 1998]

**XPath:** Extensible Path Language. Vom W3C standardisierte Sprache, die Teile von XML-Dokumenten adressieren und auswählen kann. XPath ist die Grundlage für weitere W3C-Standards wie beispielsweise XSL und XPointer. Siehe auch [Clark et al. 1999]

**XPointer:** XML Pointer Language. Vom W3C standardisierte Sprache, die Teile von XML-Dokumenten adressieren kann, um die adressierten Ziele in Verweisen wie beispielsweise URLs verwenden zu können. Siehe auch `http://www.w3.org/TR/xptr`

**XSL:** Extensible Stylesheet Language. Vom W3C standardisierte Stylesheet-Sprache in XML-Syntax bestehend aus den Teilen **XSLT** (XSL Transformations) zur Transformation von XML-Dokumenten und **XSL-FO** (XSL Formatting Objects) zur Zuweisung von Formatierung. Siehe auch `http://www.w3.org/Style/XSL/`

# 11 Index


## A

Adobe ..... 7, 16
Aktionen ..... 55
Anker ..... 65
    siehe auch HTML/Anker
AnswerBook2 ..... 7, 12
Applet, siehe Java-Applet
Arbortext ..... 68
Aufzählungstyp in Java ..... 63
    einfach ..... 63
    typsicherer ..... 63

## B

Bean, siehe JavaBean
Bosak, Jon ..... 14
Browser ..... 8
Browsing ..... 8

## C

C++ ..... 50
CDE ..... 7, 15
Common Desktop Environment, siehe CDE
Cover, Robin ..... 7, 15

## D

Datenmodelle ..... 44
    Geschichte ..... 45
    Hilfe-Dokument ..... 44
    Lesezeichen ..... 44
    Suchanfrage-Geschichte ..... 45
Datenstrom ..... 45
doc.sun, siehe AnswerBook
DocBook ..... 2, 5
    Version 2.4.1 ..... 12
Document Object Model, siehe DOM
document type definition, siehe DTD
Dokumenttyp-Definition, siehe DTD
DOM ..... 4, 55
    Baumknoten ..... 56
    Bean ..... 60
DSSSL ..... 12
DTD ..... 4
    Helptag ..... 15
    SDL ..... 15
    Solbook ..... 12
DynaText ..... 7, 9
    Publisher ..... 9

## E

Ebene ..... 52
    1 (spezielle Ansichten) ..... 54
    2 (Hilfe-Dokument-Fenster) ..... 53
    3 (Hilfe-Dokument) ..... 53
    4 (Anwendung) ..... 53
Eigenschaft
    markedEntry (gebunden) ..... 55
    markedNode (gebunden) ..... 54
Ein-/Ausgabe
    Module ..... 45
Electronic Book Technologies ..... 7, 9
Entwurf ..... 41
Entwurfsmuster ..... 49
    MVC ..... 41
    Observer ..... 50
Ereignis ..... 50
    multicast ..... 50
    unicast ..... 50
event, siehe Ereignis
Exploration ..... 8
Extensible Markup Language, siehe XML

## F

Forté for Java Community Edition ..... 64
FrameMaker ..... 1, 7, 16

## G

Ghostscript ..... 1

## H

Hash ..... 48
Help Viewer ..... 7, 15
Hilfesystem
    Online-Hilfesystem ..... 3
HTML ..... 66
    Anker ..... 66
    Version 3.2 ..... 13, 19, 21
HTML Help ..... 7, 21
Hyperlink ..... 8
    benutzerdefiniert ..... 11
    dangling links ..... 13
    DocBook
        FirstTerm ..... 36
        FootnoteRef ..... 36
        GlossSee ..... 36
        GlossSeeAlso ..... 36

GlossTerm ..... 36
Link ..... 36
OLink ..... 36
ULink ..... 36
XRef ..... 36
mehrfach ..... 11
zweiseitig ..... 11
Hypermedia ..... 3
Hypertext ..... 3

**I**

INSO ..... 10
Interface ..... 49
Implementierung ..... 50
Typ ..... 50

**J**

Java ..... 24
Applet ..... 21
Bean ..... 20, 41, 50
Eigenschaft ..... 50
gebunden ..... 51
Development Kit (JDK) ..... 63
JavaHelp ..... 7, 19
JavaSoft ..... 7, 19
Journal ..... 11

**L**

LaTeX ..... 1
LayerEvent ..... 55

**M**

Microsoft ..... 7, 21
model view controller, siehe Entwurfsmuster/MVC

**N**

Navigation ..... 8

**O**

Objectteam ..... 7, 10
Observant ..... 50
Observer ..... 50

**P**

Panorama Viewer ..... 7, 18
PDF ..... 13
Perl ..... 24
Postscript ..... 1
Public Identifier ..... 68
OASIS-Catalog ..... 68
XML-Catalog ..... 68
Python ..... 24

**S**

SAX ..... 68
Scheme ..... 24
Schnittstelle, siehe interface
Seitenbeschreibungssprache ..... 1
SESAM
SESAM-2 ..... 1
SESAMDoc ..... 2
Browser ..... 29, 31
Simple API for XML, siehe SAX
SoftQuad ..... 7, 18
stream, siehe Datenstrom
Sun Microsystems ..... 7, 12
Swing ..... 56
Text-Komponenten ..... 56
Dokument-Modell ..... 56
EditorKit ..... 57
JEditorPane ..... 57
JPasswordField ..... 57
JTextArea ..... 57
JTextField ..... 57
JTextPane ..... 57, 58
Teilansicht ..... 57
System Identifier ..... 67
Systemkern ..... 42

**T**

Tcl/Tk ..... 24
TreeMap ..... 48

**U**

Uniform Resource Identifier, siehe URI
Uniform Resource Locator, siehe URL
URI ..... 68
URL ..... 13

**W**

Walsh, Norman ..... 57, 68
WYSIWYG ..... 16

**X**

Xalan-J ..... 58, 66
Xerces-J ..... 60
XLink ..... 14, 31
XML ..... 2, 3
XPath ..... 27, 66

XPointer ..... 31, 66
XSLT
    Modulare Stylesheets
        für DocBook ..... 35, 57
    Prozessor ..... 58

Ich versichere, daß ich diese Arbeit selbständig verfaßt und nur die angegebenen Hilfsmittel verwendet habe.