Diplomarbeit
Visualisierung von digitalen Bibliotheken

ausgeführt am
Institut für Softwaretechnik
der Technischen Universität Wien

unter der Anleitung von

O.Univ.Prof. Dipl.-Ing. Dr.techn. A Min Tjoa

und der Betreuung von

Univ. Ass. Dipl.-Ing. Andreas Rauber

durch

Harald Bina
Neulerchenfelderstraße 57/17
1160 Wien

29. März 2000 Harald Bina

Inhaltsverzeichnis

Kapitel 1

Einleitung

Das libViewer-System ist eine Client-Server Applikation mit grafischem Frontend zur Visualisierung von digitalen Bibliotheken. Das in JAVA entwickelte Paket besteht aus einem Applet (libViewer) und zwei Servern (AVServer und DCServer), die in einer Zeitspanne von etwa einem Jahr am Institut für Softwaretechnik enstanden sind. Dieses Kapitel beschreibt die Idee bzw. Motivation hinter diesem Projekt und zeigt mögliche Benefits für den Forschungsbereich „Web-basierende Informationsvisualisierung“ auf. Abschließend wird noch auf den kreativen Prozeß bzw. die Entwicklungsgeschichte näher eingegangen.

1.1 Idee und Benefit

Wenn man sich heute die Repräsentation von Ergebnissen einer Suchmaschine des World Wide Webs genauer ansieht, so wird man eines sofort feststellen: Die Metainformation der zurückgelieferten Dokumente wird vorwiegend textuell dargestellt. Beispielsweise werden Größe und Relevanz des Dokuments sowie Information über die verwendete Sprache fast ausschließlich aus Textbausteinen zusammengesetzt.

Denselben Eindruck gewinnt man auch bei gebräuchlichen Bibliotheksuchsystemen wie z.B. dem OPAC-System. Auch hier basiert die Visualisierung auf Text - graphische Metaphern werden selten bis gar nicht verwendet.

Menschen, die schon lange mit dem Medium Computer umgehen, also mit diesem schon sehr vertraut sind, werden diese Repräsentation weniger als störend empfinden, wenn nicht sogar bevorzugen. Durch die fortschreitende Verbreitung des Internets auf alle gesellschaftlichen Schichten entsteht aber eine neue Situation: Sehr viele Benutzer von computerbasierten Informationssystemen wie z.B. dem Internet sind mit textuellen Metainformationen

nicht vertraut. Dieser Anwenderschicht ist es oft nicht möglich, Textbausteine wie z.B. die Größenangabe eines Dokuments richtig zu interpretieren. Die Maßeinheit Bytes bzw. Kilobytes ist für einen Computerlaien völlig unverständlich. Aus diesem Grund ist eine Interpretation des Ergebnisses in Bezug auf dessen Größe schwer bis unmöglich [19]. Weitere Beispiele von textueller Metainformation, die keine intuitive Interpretation erlauben, sind die Relevanz, das Alter, die Sprache, die Art oder die Quelle der Information (z.B. Verlag).

Durch die Steigerung der graphischen Performance von Rechnern vor allem im Heimcomputerbereich und durch die Entwicklung von Client-basierenden Programmiermethoden wie z.B. der JAVA-Applet Technologie sind viele neue Möglichkeiten der graphischen Visualisierung entstanden. Mußte früher noch jede Graphik vom Server über das Netz zum Client übertragen werden, so kann jetzt nur noch die Metainformation gesendet und von der Client-Applikation graphisch umgesetzt werden.

Aus diesem Umstand heraus haben wir in diesem Projekt unter Zuhilfenahme der neuen Technologie an einer Verbesserung der aktuellen Visualisierungsmethoden von Metainformationen gearbeitet. Die Visualisierung sollte vor allem intuitiv interpretierbar werden, also auch für weniger versierte Anwender verständlich sein.

Wenn man sich in einer Bibliothek oder in einer großen Buchhandlung aufhält, so findet man meistens durch einen sehr intuitiven Prozeß automatisch zum passenden Regal. So haben z.B. Regale mit Wörterbüchern ein anderes Aussehen zu solchen mit Diplomarbeiten. Durch jahrelanges Training hat der Mensch Verhaltensweisen entwickelt, die ihm dieses effiziente, intuitive Auffinden der gesuchten Information in Bibliotheken ermöglicht. Unsere Idee war es, dem Benutzer durch Visualisierung dieses Real-World-Szenarios am Computer die Möglichkeit zu geben, diesen intuitiven Prozeß auch hier anwenden zu können.

1.2 Der kreative Prozeß

Wir stellten uns am Anfang dieses Projekts die Aufgabe, eine Verbesserung der Darstellung von Metadaten zu erreichen. Metadaten wie z.B. der Titel, Autor, Sprache, Art des Dokuments usw. sollten durch geeignete Mapping-Verfahren auf bestimmte Metaphern abgebildet werden. Dabei war es am Anfang nicht klar, wie diese Metaphern aussehen werden, d.h. ob also „Real-World Metaphern“ oder abstrakte Darstellungsformen zum Einsatz kommen.

Nach der ersten Brain-Storming Phase wurde die Entscheidung getroffen, sich vorerst auf die Idee der Visualisierung von Real-World Metaphern

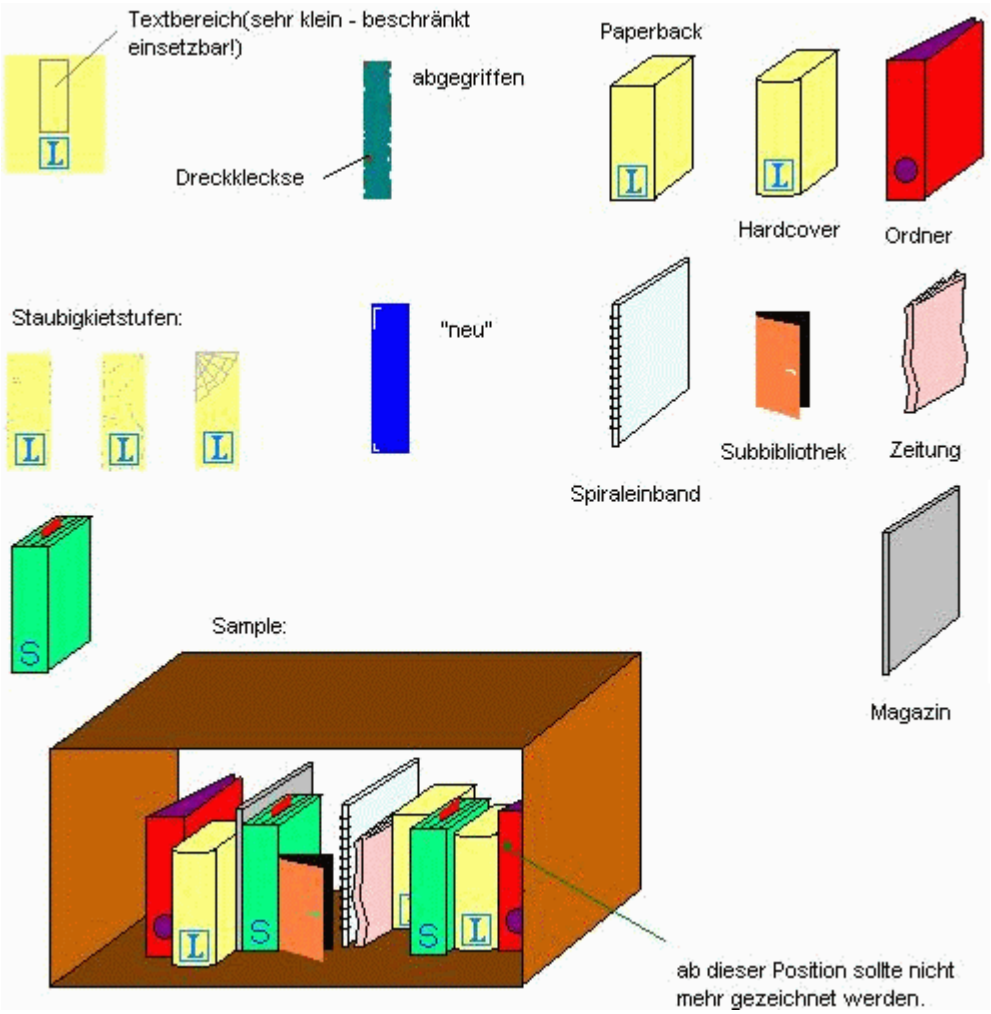

Abbildung 1.1: Erster Metapher-Entwurf

zu beschränken. Wir untersuchten die Merkmale von Büchern und wie man Metadaten auf diese Merkmale abbilden kann. Insbesondere waren uns verschiedene Ausprägungen von Dokumentmetaphern wichtig, wie Abbildung 1.1 des ersten Entwurfes zeigt.

Da die Umsetzung dieses Ansatzes zumindest als Prototyp ein wichtiges Ziel war, untersuchten wir die Machbarkeit der Gestaltungsmerkmale (Titel am Buchrücken, Logos für den Herausgeber, verschiedene Buchtypen). Es entstand daraus ein Entwurf, bei dem die Minimalgröße eines Buches (in Pixeln) definiert wurde, wo die geplanten Merkmale noch sinnvoll visualisiert werden können.

Parallel zur Entwicklung des ersten Prototyps, der Bücher in einer Reihe

zeichnen kann, suchten wir nach einem geeigneten Format für die dazustellenden Metadaten. Es wurde uns klar, daß nur eine schon jetzt standardisierte Form von Metadaten unserem Prototypen eine Zukunft geben kann. Daher entschieden wir uns zur Integration des Dublin Core Element Sets (siehe Kapitel 3).

Der Reihe nach wurden viele Arten von Mappings konzipiert und umgesetzt, sowie ein völlig flexibles Mappingschema angedacht. Bei diesen Mapping-Verfahren wird es möglich, mit Hilfe einer Steuer-Datei eine bestimmte Auswahl an Metadaten (z.B. die Sprache oder den Verlag) auf das Aussehen der Metaphern (Farbe, Logo für den Verlag, etc.) überzuleiten. Ein völlig flexibles System würde dabei die Möglichkeit bieten, jedes zur Verfügung stehende Metadaten-Element jedem existierenden Metapher-Merkmal zuzuordnen. Eine detaillierte Beschreibung dieser Mapping-Verfahren finden Sie in Kapitel 5.

Neben der Implementierung einer Übersicht, in der alle darzustellenden Dokumente verkleinert gezeichnet werden, kreierten wir auch die Anbindung an die Internet-Suchmaschine AltaVista und teilten das bisherige stand-alone Applet in einen Client für die Visualisierung und zwei Server, den Alta-Vista und den Dublin-Core Server, auf (siehe Kapitel 5).

Sehr interessant ist auch, daß uns manche Metaphern sowie deren Umsetzung erst während der Realisierungsphase eingefallen sind. Die gesamte Entwicklungsperiode war ein stetes Wechselwirkungsspiel zwischen Kreativität und Umsetzung, Machbarkeit und Phantasie. Ähnlich wie beim Brain Storming wurde kein Ansatz als übertrieben angesehen sondern als Vision notiert. So sind heute einige dieser Ideen umgesetzt und manch andere leben vorerst noch in unseren Köpfen.

1.3 Aufbau

Dieses Kapitel, die Einleitung, hat neben der zugrundeliegenden Idee mögliche Benefits dieses Projekts für die Visualisierung digitaler Bibliotheken beschrieben und den kreativen Prozeß skizziert.

Kapitel 2 vermittelt Grundlagen zu digitalen Bibliotheken. Neben der Geschichte von „Digital Libraries“ werden Probleme bei der Visualisierung von digitalen Dokumentsammlungen aufgezeigt. Wir haben in diesem Kapitel analysiert, warum die Verbesserung der grafischen Repräsentation im Bereich Bibliotheksvisualierung notwendig und sinnvoll ist.

Das dritte Kapitel beschäftigt sich mit Metadaten für digitale Bibliotheken. Es wird definiert, was Metadaten eigentlich sind und gängige Metadaten-Standards (BibTeX, MARC, Dublin Core, etc.) vorgestellt.

„Metaphern für Informationsressourcen“ ist der Titel für eines der Kernkapitel (Kapitel 4) dieses Werkes. Um Metaphergrafiken richtig für die Visualisierung von Informationsressourcen einsetzen zu können, haben wir die dafür notwendigen Designrichtlinien beschrieben. Teil 2 dieses Kapitels bietet eine detaillierte Beschreibung aller von uns identifizierten Metaphern zur Visualisierung von Metadaten in digitalen Bibliotheken. Abschnitt 3 dieses Kapitels beschreibt anhand von Beispielgrafiken die Umsetzung der identifizierten Metaphern im libViewer.

Kapitel 5 beinhaltet die technische Systembeschreibung in Form einer Klassenreferenz, damit das libViewer-System auch in Zukunft für Neueinsteiger wartbar bzw. erweiterbar ist. Dabei werden alle Java-Klassen des libViewers, DCServers und des AVServers detailliert beschrieben. Anbei findet sich auch das libViewer-libServer Protokoll, das die Kommunikation zwischen Client und Server regelt.

Das 6. Kapitel beschreibt das User Interface des libViewers. Nach einer detaillierten Auflistung und Erklärung der einzelnen grafischen Komponenten des libViewer-Applets wird anhand von 2 Beispielsessions auf die Interaktion des libViewers mit dem DCServer sowie mit dem AVServer anhand von Screenshots näher eingegangen. Hier werden die von uns kreierten Metaphern in der Praxis vorgestellt.

Die Ergebnisse der von uns durchgeführten „Usability Evaluation“ kann in Kapitel 7 nachvollzogen werden. Diese Erkenntnisse konnten durch die Befragung unterschiedlicher Anwendergruppen gewonnen werden. Zu diesen Personen zählen u.a. unversierte Computeranwender, Bibliothekare und Experten auf dem Gebiet „User Interface Design“.

Kapitel 8 gibt einen Überblick über verwandte Arbeiten im Bereich Informationsvisualisierung (Related Work). Projekte wie das „Bookhouse Project“, das SOMLib System oder das kommerzielle Produkt „Insight“ sind ebenso in diese Aufstellung enthalten wie die Forschungsarbeiten bei Xerox PARC.

In Kapitel 9, welches gleichsam den Abschluß dieser Diplomarbeit bildet, haben wir noch einmal die Motivation hinter diesem Projekt skizziert und alle gewonnenen Erkenntnisse zusammengefaßt. Ein Ausblick auf geplante Erweiterungen des libViewer-Systems beendet dieses Werk.

Kapitel 2

Digital Libraries

Bibliotheken gibt es in vielen Arten. So gehören in der Computerwelt Code Libraries genauso unzertrennbar zum Software Engineering wie Object bzw. Class Libraries. Die Multimedia Technologie bringt uns digitale Bilder- und Audiobibliotheken, ja sogar digitale Videobibliotheken. Im Prinzip könnte man jede Ansammlung von Information, welche z.B. in Wissensdatenbanken, Textdatenbanken bzw. im Gopherspace und natürlich auch im World Wide Web abgelegt ist als Bibliothek bezeichnen. [7].

Während einerseits die Artenvielfalt der Information und die Organisationsform ständig erweitert bzw. verändert wird, so wenden traditionelle Bibliotheken einen immer größer werdenden Teil ihrer finanziellen Mittel dafür auf, ihre elektronischen Dienstleistungen zu erweitern. Beispiele sind Bücher auf CD-ROM wie auch öffentlich zugängige Kataloge und Datenbanken im Internet. Diese Entwicklung wird sich fortsetzen, da die Kosten für digitalen Speicherplatz gegenüber herkömmlichen Speichermedien (z.B. großer Platzbedarf für Bücherregale) weiter sinken. Diese Services werden sich auch deswegen durchsetzen, da die dazugehörigen Benutzeroberflächen immer besser und die Dienstleistungen selbst effizienter und billiger sein werden. Diese Entwicklung wird zusätzlich angetrieben durch das große Interesse an Nachrichten, der allgegenwärtigen Präsenz elektronischer Medien, dem Bedürfnis sich mit seinen Kollegen auszutauschen und durch die aufregende Motivation, einen immer größer werdenden Pool an Informationen benützen zu können. Das sind nur einige von vielen Gründen, warum die Menschheit angestrengt an dieser großen Herausforderung des „World Digital Library System“ arbeitet.

2.1 Geschichte

Der Term „digital library“ ist der neueste in einer Reihe von Namen für ein Konzept, an dem nahezu gleich lange gearbeitet wird wie an dem Computer selbst: Die Idee einer vollelektronischen „Bibliothek“, welche traditionelle Bibliotheken unterstützt und deren Funktionalität erweitert - oder diese sogar ersetzt. [10]

Vannevar Bush (1945) [2] schrieb über „Memex“, welches oft als ein System zitiert wird, das die Computeranwendungen des „information retrieval“ stark beeinflußt hat. „Memex“ war ein mechanisches System, basierend auf der Microfilm-Technologie, welches schon damals die Idee von Hypertext umzusetzen versuchte. Die Automatisierung von Bibliotheken begann in den frühen 50er-Jahren mit Lochkarten-Systemen. Licklider (1965) [15] prägte die Phrase „library of the future“ in Bezug auf seine Vision der vollkommen computerisierten Bibliothek. Schon 10 Jahre später schrieb F. W. Lancaster (1978) [14], daß die sogenannte „paperless library“ schon bald Realität sein würde. Zur selben Zeit erfand und benannte Ted Nelson (1974) [17] den hyperspace bzw. den hypertext, war jedoch nie imstande, ein einsetzbares System zu implementieren. In der nahen Vergangenheit wurden auch Ausdrücke wie „electronic library“, „virtual library“, „library without walls“, „bionic library“ und andere verwendet. Karen Drabenstott (1993) [6] schrieb einen exzellenten analytischen Überblick über dieses Thema und über verwandte Literatur.

Der relativ junge Begriff der „digital library“ wurde von der Digital Libraries Initiative geprägt, welche von der National Science Foundation, der Advanced Research Agency und der National Aeronautics and Space Administration der USA finanziert wird. 1994 investierten diese Agenturen 24,4 Millionen USD in sechs amerikanische Universitäten für Forschung im Bereich digital libraries wegen der plötzlichen Explosion des Internets und der Entwicklung grafischer Web-Browser. Der Ausdruck „digital library“ wurde rasch von Computerwissenschaftern, Bibliothekaren und anderen aufgenommen. Trotz der Neuheit dieses Ausdrucks wird also schon seit Jahrzehnten an der Integration von digitalisierten Informationen in Bibliotheken gearbeitet. Man kann deshalb viel von den Bibliothekaren und Informatikern, die bei diesen Projekten noch vor der Zeit des Internets mitgearbeitet haben, lernen und auf ihre Erfahrung in den Bereichen Automation von Bibliotheken bzw. das Finden von Informationen (information retrieval) zurückgreifen.

2.2 Visualisierung digitaler Bibliotheken

Forschung und Entwicklung im Bereich Visualisierung hat in unserer immer mehr an Informationsreichtum zunehmenden Gesellschaft massiv den Weg verändert, wie wir große und komplexe Daten repräsentieren. [8] Viel von der vorausgegangenen Forschung im Bereich Visualisierung wurde von der „scientific community“ angetrieben, die versucht hat, einen Weg zu finden, der die Interaktion mit großen Beständen von wissenschaftlichen Daten erleichtert. Heutzutage gibt es einen neuen Trend: Das explosive Anwachsen des Internets, die vollständige Computerisierung der Wirtschaft und der Einsatz von „data warehouses“ haben dazu geführt, daß in sehr vielen Bereichen eine generelle Befürwortung für neue Visualisierungstechniken entstanden ist und diese Werkzeuge für weite Bereiche der Wirtschaft und Technik verlangt werden. Informationsvisualisierung im kommerziellen Sektor beispielsweise muß es den Benutzern möglich machen, Informationen schnell zu bekommen, den Sinn zu extrahieren und das gewünschte (Such-)Ziel in relativ kurzer Zeit zu erreichen.

Das World Wide Web (WWW), möglicherweise schon der größte zusammenhängende Informationsbestand überhaupt, hat die „visualization community“ in die Lage gebracht, neue Visualisierungen einer großen Anwenderschaft zur Verfügung stellen zu können. Das WWW hat jetzt schon die Art verändert, wie Visualisierung zum Endbenutzer gebracht wird. Man denke hierbei nur an zahlreiche in JAVA oder VRML (Virtual Reality Modelling Language) entwickelte Applikationen, die nur noch die darzustellenden Daten über das Netz empfangen und die Visualisierung am Client ausführen. Die Entscheidung, welche Art von Visualisierung dann schlußendlich verwendet wird, obliegt damit dem Client und ist ganz auf seine Probleme, Anwendungsgebiete und technischen Voraussetzungen abgestimmt. Weiters wird Zeit und Netztransfer eingespart. Es bleibt nur zu hoffen übrig, daß sich vernünftige Standards für diese Sprachen entwickeln.

Im Gegensatz zu wissenschaftlicher Visualisierung, welche hauptsächlich von Personen mit guten Vorkenntnissen verwendet wird, werden heutzutage durch die enorme Verbreitung des Internets hochentwickelte Visualisierungsmethoden von einem sehr unterschiedlichen und nichttechnischen Publikum benutzt. Die Nachfrage nach guter und effektiver Informationsvisualisierung umfaßt alle Lebensbereiche und Interessen. Die große Differenz unter den Anwendern läßt sich auf ihre Bildungsunterschiede, ihre unterschiedlichen Hintergründe, Fähigkeiten und Bedürfnisse zurückführen. Daher werden wir visuelle Darstellungsarten entwickeln müssen, die den spezifischen Bedürfnissen und Problemen der einzelnen Benutzerschichten entsprechen.

Informationen können, wenn sie in hoher Quantität sehr schnell eintref-

fen, eine Informationslawine bilden. Große Textkörper zwingen den Benutzer, viele Dokumente zu lesen, zu verstehen und den Sinn, der sich in den Informationen versteckt, herauszufinden. Wenn man die Informationsressourcen und die dazu gespeicherten Metadaten in visueller Form (z.B. durch Metaphern) darstellt, bekommt der Anwender die Möglichkeit, dieses Meer an Informationen durchblättern zu können und relevante bzw. interessante Textteile schnell identifizieren zu können. Durch gute visuelle Schnittstellen können wir mit großen Datenmengen, wie sie in digitalen Bibliotheken zu finden sind, auf schnelle und effektive Art interagieren und versteckte Charakteristika bzw. Muster herausfinden.

Aufgrund dieser Überlegungen und Bedürfnisse erscheint es uns als sinnvoll und notwendig, neue Visualisierungstechniken speziell für den unversierten Anwender, den „every-day user“, zu entwickeln. Allgemein verständliche, intuitiv interpretierbare Metaphern zur Visualisierung von Metadaten zu Informationsressourcen müssen gefunden werden, um das Blättern in digitalen Bibliotheken effizienter gestalten zu können. Ziel dieser Arbeit war es, für oben aufgezeigte Problemstellungen Lösungsansätze zu entwickeln, d.h. die neuen technischen Möglichkeiten zu nutzen um auf die geänderte Benutzerschicht und das massive Anwachsen der Informationsmenge zu reagieren.

2.3 Zusammenfassung

Dieses Kapitel hat Grundlagen zu digitalen Bibliotheken vermittelt. Neben der Geschichte von „Digital Libraries“ haben wir auch Probleme bei der Visualisierung von digitalen Dokumentsammlungen aufgezeigt. Anbei findet sich auch eine Analyse, warum wir die Verbesserung der grafischen Repräsentation im Bereich Bibliotheksvisualierung als sinnvoll und notwendig erachten.

Kapitel 3

Metadaten für digitale Bibliotheken

Dieses Kapitel beschreibt eingangs kurz das Wesen von Metadaten und gibt anschließend einen Überblick über Metadaten-Standards im Bereich von digitalen Bibliotheken. Dabei wird das Dublin Core Element Set detaillierter beschrieben, da wir dieses Format als eine sehr vielversprechende Metadatendefinition für digitale Bibliotheken einschätzen und mit dem *DC-Server* (siehe 5.2) eine Schnittstelle entworfen haben, die es dem *libViewer* möglich macht, Metadaten im Dublin Core-Format einzulesen.

3.1 Was sind Metadaten?

Metadaten sind strukturierte Daten um die Charakteristika einer Ressource zu beschreiben. Das Wort „meta“, welches aus der griechischen Sprache kommt, beschreibt eine Natur von höherer Ordnung oder von größerer fundamentaler Art. Daher bezeichnet man Metadaten auch als übergeordnete Daten oder als „Daten über Daten“. Ein Metadatensatz (Record) besteht aus einer Anzahl von vordefinierten Elementen, welche die spezifischen Attribute einer Ressource repräsentieren, wobei jedes Element einen oder mehrere Werte haben kann. Jedes Metadaten-Schema hat üblicherweise die folgenden Ausprägungen:

- eine begrenzte Anzahl von Elementen
- einen Namen für jedes Element
- die Bedeutung jedes Elements

Typischerweise beschreiben Metadaten für Bibliotheken den Inhalt, den Aufenthaltsort, physische Attribute, die Art (z.B. Text oder Bild) und die Form (z.B. Ausdruck, elektronische Datei) einer Ressource. Metadatenschemata für veröffentlichte Dokumente enthalten zusätzliche Elemente wie den Verfasser der Arbeit, den Titel, Information darüber wann und wo das Werk publiziert wurde und welche Themenbereiche die Arbeit abdeckt. Dort, wo die Information in analoger Form vorliegt - beispielsweise bei gedrucktem Material - gibt es zusätzliche Metadaten, die den Aufenthaltsort der Ressource genauer beschreiben (z.B. Nummernvergabe in Bibliotheken). In konventionellen Bibliotheken findet man oft separate Kataloge für Metadaten, welche die Bücher sortiert nach Titel, Autor, Thema, usw. auflisten.

3.2 Metadaten-Standards

Im Bereich von digitalen Bibliotheken gibt es eine Vielzahl von Initiativen, die sich mit der Entwicklung von Metadaten-Standards für digitale Sammlungen beschäftigen. In diesem Abschnitt geben wir einen Überblick über die gängigsten Metadaten-Standards.

3.2.1 BibTeX

BibTeX ist eines der älteren Beispiele für eine Metadatendefinition für digitale Bibliotheken und wurde im Jahr 1985 von Olen Patashnik und Leslie Lamport für das Erzeugen von Literaturverzeichnissen unter dem LaTeX System entwickelt. Dieses Format basiert gänzlich auf Zeichen und kann deswegen von Fremdprogrammen leicht eingebunden werden. Es besteht aus Feldern (tags), wobei das BibTex Programm unbekannte Felder ignoriert und daher Erweiterungen möglich macht. BibTeX ist wahrscheinlich das gängigste Format für Literaturverzeichnisse im Internet.

BibTeX unterstützt 15 verschiedene Standard Dokumenttypen (z.B. article, book, masterthesis, proceedings, manual, techreport, usw.) und 2 Erweiterungen (collection, patent). Weiters kann jedem Dokumenttyp ein Set von 24 Attributen zugeordnet werden, wobei bespielsweise der Autor, der Buchtitel, der Verlag oder die Seitenanzahl näher spezifiziert wird. Weil BibTeX sehr populär ist und von vielen Anwendern zur Speicherung von Informationen benutzt wird, haben sich im Laufe der Zeit etliche Erweiterungen zu den ursprünglichen Attributen herausgebildet. Zu den gängigsten Erweiterungen gehören z.B. Felder zur Spezifikation eines Abstracts, einer Copyright-Information oder zum Speichern des Dokumentpreises. Detaillierte Informationen zum BibTeX-Format finden sich wegen dessen weiter Verbreitung auf

vielen Seiten des Internets[1].

3.2.2 MARC

Das MARC-Format ist eines der unfangreichsten Metadaten-Formate und steht für „Machine Readable Catalogue Format“. Es wurde in den 60er Jahren für den Austausch von Datensätzen in Bibliothekskatalogen geschaffen. Aus dem MARC-Grundformat heraus haben sich mehrere Standards abgeleitet, wie z.B. das USMARC-Format in den Vereinigten Staaten. Die „MARC Standards Office“ [2] wurde 1984 zur Unterabteilung der „Library of Congress“ (Washington)[3] und ist für die Pflege des MARC-Standards verantwortlich. Das MARC-Format ist ein sehr komplexes Set von Attributen um Dokumente zu beschreiben und wurde hauptsächlich für bibliographische Daten entwickelt. Aufgrund seiner Komplexität sind für eine korrekte Erstellung von MARC-Datensätzen trainierte Spezialisten notwendig, was eine Anwendung dieses Formats auf professionelle Bibliotheksbereiche beschränkt.

3.2.3 Metadaten für spezielle Anwendungsgebiete

Neben den oben vorgestellten Metadaten-Definitionen gibt es noch eine Vielzahl anderer Standards für Matadaten in digitalen Bibliotheken, die hauptsächlich für spezielle Anwendungsgebiete entworfen wurden.

Zu erwähnen wäre hier das CSDGM-Format (Content Standard for Digital Geospatial Metadata)[4], welches ein allgemeines Set von Terminologien und Definitionen zur Dokumentation von geographischen Daten zur Verfügung stellt.

Das CIMI (Consortium for Interchange of Museum Information) wiederum arbeitet seit 1990 an einem Standard für den Austausch von Informationen unter Museen. Ziel dieser Organisation ist es, derartige Informationen einheitlich zu strukturieren bzw. Metadaten-Standards dafür zu entwickeln (CIMI-Format) um weitverbreitete Suchmöglichkeiten schaffen zu können.[5]

3.2.4 Dublin Core

Das Dublin Core Element Set (DC) ist einer der vielversprechendsten Standards für Metadaten in digitalen Bibliotheken und wurde von der Dublin

[1] http://www2.ecst.csuchico.edu/ jacobsd/bib/formats/bibtex.html
[2] http://lcweb.loc.gov/marc/
[3] http://lcweb.loc.gov/
[4] http://www.fgdc.gov/
[5] http://www.cimi.org/

Core Metadata Initiative[6] entwickelt. Es besteht aus einem Set von 15 Basisattributen um alle Arten von digitalen Dokumenten beschreiben zu können und wurde von einer internationalen und interdisziplinären Gemeinschaft, der auch Bibliothekare angehören, geschaffen.

Die Dublin Core Metadata Initiative ist in der sich rasch weiterentwickelnden Infrastruktur des Internets ein wichtiger Teil geworden. Viele Gemeinschaften arbeiten eifrig an einer allgemeingültigen Basis für die Beschreibung von Ressourcen. Dublin Core bekommt für dieses Vorhaben weitreichende internationale und interdisziplinäre Unterstützung.

Die Dublin Core Metadata Initiative hat formale Verfahren definiert um die Entwicklung von Dublin Core zu unterstützen. Diese Verfahren sind großteils von anderen Gemeinschaften, die Standards entwickeln, übernommen. Die Zielsetzung von Dublin Core ist es, diese formalen Strukturen so effizient und schlank wie möglich zu gestalten und den internationalen Konsens weiterzupflegen, der als das wertvollste Gut dieser Gemeinschaft angesehen wird.

Die zur Zeit vorhandenen Ergebnisse wurden von Individualisten erarbeitet, die mit ihrem eigenen Enthusiasmus und Intellekt an Lösungen mitgearbeitet haben. Um diesen Prozeß jedoch aufrechterhalten zu können, bedarf es in Zukunft einer noch stärkeren Unterstützung.

Alle Elemente sind optional und überlappen sich teilweise mit den Elementen des MARC-Formats, sind aber weniger spezifisch. Dublin Core ist syntax-unabhängig, d.h. die Elemente können beispielsweise in Form von MARC-Records abgelegt oder durch HTML META Tags ausgedrückt werden. Im Gegensatz zu MARC (welches für professionelle Anwendungsbereiche entwickelt wurde) ist die Intention von DC ein allgemein verständliches Set von Elementen zu sein; es wurde nicht geschaffen, um alle spezifischen Bedürfnisse einzelner Gemeinschaften umzusetzen. Jede Gemeinschaft ist aufgefordert, die Basiselemente ihren Anforderungen gemäß zu erweitern. Das DC-Format wird heute in vielen Projekten verwendet um eine Vielzahl von Dokumenten wie z.B. Webpages oder digitale Archive zu beschreiben.

Kategorisierung der Elemente

Untenstehende Tabelle kategorisiert die Elemente in Dublin Core (Version 1.0, Reference Description). Die Metadaten-Elemente reihen sich in 3 Gruppen ein, welche grob die Hauptkategorien der gespeicherten Informationen ausdrücken: (1) Elemente welche sich hauptsächlich auf den Inhalt beziehen, (2) Elemente die das intellektuelle Eigentum beschreiben, (3) Elemente zur

[6] http://purl.oclc.org/dc/

Beschreibung der physischen Manifestation der Ressource.

Informationen über die Ressource / Inhalt	intellektuelles Eigentum	elektronische oder physische Manifestation
Title	Author oder Creator	Date
Subject	Publisher	Type
Description	Other Contributor	Format
Source	Rights	Identifier
Language		
Relation		
Coverage		

Tabelle 3.1: Klassifikation der DC-Elemente

Detailbeschreibung der Elemente

In diesem Abschnitt folgt eine detaillierte Beschreibung der Elemente in Dublin Core, wie sie in der Version 1.0 veröffentlicht wurde.

- Title: Der Name der Ressource, welcher vom Autor oder Verleger vergeben wurde.
- Author oder Creator: Die Person oder Organisation, die in erster Linie verantwortlich für die Herstellung des intellektuellen Inhalts der Ressource ist.
- Subject und Keywords: Das Thema der Ressource. Typischerweise wird das Thema in Phrasen oder Stichwörtern ausdrückt. Die Verwendung eines kontrollierten Vorkabulars bzw. einer formalen Klassifikation wird empfohlen.
- Description: Eine textuelle Beschreibung des Inhalts der Ressource. Bei dokumentähnlichen Objekten beinhaltet die Beschreibung einen Abstrakt oder im Fall von virtuellen Ressourcen eine Inhaltsbeschreibung.
- Publisher: Die Person oder Organisation, welche verantwortlich für die Herstellung der Ressource in seiner momentanen Form ist. Dies kann beispielsweise ein Verlagshaus, eine Universitätsabteilung oder eine gemeinschaftliche Organisation sein.

- Other Contributor: Eine Person oder Organisation, die nicht im Element „Creator“ spezifiziert wurde, aber einen signifikanten intellektuellen Beitrag zur Ressource beigesteuert hat (z.B. Editor, Übersetzer oder Illustrator). Dieser Beitrag ist im Gegensatz zu den im Element „Creator“ spezifizierten Personen bzw. Organisationen sekundär.

- Date: Das Datum, an dem die Ressource in ihrer derzeitigen Form verfügbar gemacht wurde. Das empfohlene Format ist eine in ISO 8601 definierte, achtstellige Nummer der Form YYYY-MM-DD (z.B. 1999-09-01 für den 1. September 1999)

- Resource Type: Die Kategorie bzw. der Typ der Ressource, z.B. home page, novel, poem, working paper, technical report, essay oder dictionary. Der Typ sollte aus Gründen der Interoperabilität aus einer Liste ausgewählt werden, die momentan in den Dublin Core Workshop Series entwickelt wird. Diese Liste (Default List of Dublin Core Resource Types, DCT1) beinhaltet in ihrer aktuellsten Version vom August 1999 folgende Typen[7]:

  - collection
  - dataset
  - event
  - image
  - interactive resource
  - model
  - party
  - physical object
  - place
  - service
  - software
  - sound
  - text

- Format: Das Datenformat und optional die Dimension (z.B. Größe, Dauer) der Ressource. Das Format wird benötigt, die Software und möglicherweise auch die Hardware zu identifizieren, die nötig ist um

[7] Siehe: http://purl.org/dc/documents/wd-typelist.htm

die Ressource zu visualisieren oder damit zu operieren. Wie beim Attribut „Resource Type “ sollte auch hier der Inhalt aus einer Liste von standardisierten und kontrollierten Werten ausgewählt werden, z.B. aus der Liste der Internet Media Types (MIME)[8], die Formate von Computermedien definiert.

- Resource Identifier: Ein String oder eine Nummer zur eindeutigen Identifikation der Ressource. Mögliche Kandidaten für dieses Element wären:
  - URLs und URNs für Netzwerkressourcen
  - International Standard Book Numbers (ISBN)
  - formale Namen
- Source: Information über eine zweite Ressource aus der die gegenwärtige Ressource abgeleitet wurde. Obwohl generell vorausgesetzt wird, daß Elemente nur Informationen über die gegenwärtige Ressource enthalten, kann dieses Element Metadaten über die zweite Ressource zur Verfügung stellen, sofern es wichtig für den Umgang mit der gegenwärtigen Ressource ist.
- Language: Sprache(n) des intellektuellen Inhalts der Ressource. Es wird empfohlen, dieses Feld laut Definition in RFC 1766[9] zu befüllen. Der Aufbau beinhaltet einen 2-Zeichen langen Sprachcode (definiert in ISO 639), der optional einen ebenso 2-Zeichen langen, in ISO 3166 definierten Ländercode angehängt haben kann. So verwendet man zum Beispiel ‚en‘ für Englisch, ‚fr‘ für Französisch oder ‚en-uk‘ für Englisch, welches in Großbritannien gesprochen wird.
- Relation: Die Beziehung dieser Ressource zu anderen Ressourcen. Die Intention dieses Elements ist es, formale Beziehungen zwischen diskret existierenden Ressourcen auszudrücken, z.B. Bilder in einem Dokument, Kapitel in einem Buch oder Stücke in einer Sammlung. Die beste Methode ist es, die Ressource mittels eines Strings oder einer Nummer, die konform zu einem formalen Identifizierungssystem ist, zu referenzieren.
- Coverage: Enthält typischerweise einen räumlichen Ort (Ortsname oder geographische Koordinaten), eine zeitliche Periode (z.B. Periodenlabel,

[8] http://www.isi.edu/in-notes/iana/assignments/media-types/media-types
[9] Siehe: http://info.internet.isi.edu/in-notes/rfc/files/rfc1766.txt

Datum oder Zeitbereich) oder Zuständigkeit (administrative Organisation). Ein kontrolliertes Vokabular, das für dieses Element verwendet werden kann, ist der Thesaurus of Geographic Names (TGN)[10].

- Rights Management: Ein Link zu einer Copyright Notice, zu einem „rights management statement“ oder zu einem Service, das derartige Informationen zur Verfügung stellt. Wird dieses Element nicht spezifiziert, so können keine Annahmen über den Status des Urheberrechts getroffen werden. Eine formale Spezifikation dieses Elements ist derzeit noch in Entwicklung.

3.3 Zusammenfassung

In diesem Kapitel wurde eingangs erklärt, was Metadaten sind und wofür sie gebraucht werden. Es folgte ein Überblick über die bekanntesten Metadaten-Standards wobei Formate wie BibTeX, MARC sowie Dublin Core vorgestellt wurden.

[10] http://shiva.pub.getty.edu/

Kapitel 4

Metaphern für Informationsressourcen

Dieses Kapitel geht im ersten Abschnitt auf Designrichtlinien ein, die bei der Gestaltung von Metaphern zur Visualisierung von Informationsressourcen bzw. zugehöriger Metadaten berücksichtigt werden müssen. Es folgt eine Auflistung aller von uns identifizierten Metaphern samt Begründung, warum uns diese als sinnvoll erscheinen. Der letzte Abschnitt wiederum widmet sich der Umsetzung dieser Metaphern im libViewer und bietet neben Beispielgrafiken auch Überlegungen zur Machbarkeit mittels Computergrafik.

4.1 Designrichtlinien

Um die Metadaten, welche in einer digitalen Bibliothek zu jedem Element gespeichert sind, visualisieren zu können, braucht man für die verschiedenen Metadaten-Attribute ein korrespondierendes Set von Metaphern, die allgemein bekannt sind und deswegen leicht verstanden werden können. Damit ein spezielles Vortraining der Benutzer wegfällt, müssen die Metaphern leicht zu identifizieren sein und sich auf Eigenschaften, wie sie aus der Realität bekannt sind, beziehen. [4]

E. R. Tufte hat in seinem Buch „Envisioning Information“ [27] aus dem Jahr 1990 vier Designrichtlinien für Metaphern zur Repräsentation von Informationen definiert. Wenn man diese Regeln beim Designprozeß von Metaphern zur Visualisierung von Ressourcen in digitalen Bibliotheken anwendet, so sollen die Metaphern

- die Ressource so bezeichnen, daß sie intuitiv erfaßbar wird (labeling),
- quantitative Information transportieren (measuring),

- Realität repräsentieren oder nachahmen und
- die Darstellung beleben und verschönern.

Basierend auf diesen Vorsätzen haben wir ein geeignetes Set von Metaphergrafiken kreiert, mit dem es möglich ist, eine Vielzahl der verschiedenen Metadaten-Attribute in digitalen Bibliotheken zu visualisieren. Eine Zuordnung der Attribute zu den Eigenschaften der Metaphern ergibt sich aus dem Anwendungsbereich und kann durch Implementierung eigener Überleitungsschemata völlig flexibel gestaltet werden. Mit dem Dublin-Core Server und dem AltaVista Server haben wir 2 Prototypen realisiert, die auf den jeweiligen Anwendungsbereich zugeschnittene Zuordnungen von Metadaten-Attributen zu den unten vorgestellten Metaphern demonstrieren.

4.2 Metaphern zur Visualisierung von Metadaten in digitalen Bibliotheken

In diesem Abschnitt geben wir einen Überblick über alle von uns identifizierten Eigenschaften von Metaphern zur Visualisierung von Metadaten in digitalen Bibliotheken. Wir haben herkömmliche Dokumentsammlungen - beispielsweise das eigene Bücherregal oder eine öffentliche Bibliothek - in Bezug auf die physischen und optischen Merkmale der Elemente analysiert und deren Eigenschaften kategorisiert.

4.2.1 Typ

Um die Elemente einer digitalen Bibliothek visualisieren zu können, braucht man für jedes dieser Elemente eine physische Repräsentation. Wir haben ein Set von Dokumentypen definiert, mit denen eine Vielzahl der Ressourcen in digitalen Bibliotheken möglichst realistisch repräsentiert werden kann:

- hardcover (Buch mit Harteinband)
- paperback (Buch mit Papiereinband)
- Ordner
- Zeitung
- Boxen für Ressourcen vom Typ Audio, Video und Software

- Boxen zur Repräsentation von Verknüpfungen zu Unter-Bibliotheken (Sublibraries)
- Boxen für alle anderen, nicht näher spezifizierten Elemente

4.2.2 Farbe

Die Farbe eines Gegenstands ist neben seiner physischen Gestalt die dominanteste Eigenschaft, da sie auch aus großer Distanz gut wahrgenommen werden kann. Farbe kann dazu verwendet werden um z.B. Information über die Sprache, das Genre sowie die Zugehörigkeit von Dokumenten zu Serien (man denke nur an die gelbe Farbe bei Langenscheidt-Wörterbüchern) in einer sehr eindrucksvollen Art zu transportieren. Gerade wegen der Dominanz dieser Eigenschaft muß mit der Farbwahl sehr sorgfältig umgegangen werden, um Fehlinterpretationen durch den Betrachter zu verhindern.

4.2.3 Größe

Unbewußt beurteilen wir die Menge der in einem Buch oder Magazin enthaltenen Information intuitiv aus der Größe des physikalischen Objekts. So schließen wir beispielsweise von der Dicke eines Buches automatisch auf die ungefähre Seitenanzahl, mit der wir in weiterer Folge die Menge der enthaltenen Information beurteilen.

4.2.4 Format

Neben dem Typ eines Dokuments vermittelt auch das Format viel Information über das Genre eines Dokuments. So hat zwar ein übergroßes Buch wie z.B. ein Atlas oder ein Bildband grundsätzlich dieselben physischen Merkmale wie ein kleines Taschenbuch (rechteckige Form, Papiereinband), jedoch sind sie in ihren Abmessungen völlig unterschiedlich. Wenn man also Realität imitieren will, kann man das Format auf vielfältige Weise dazu einsetzen, verschiedene Dokumenttypen besser unterscheidbar zu machen.

4.2.5 Logo

Wenn man Dokumente in einem Bibliotheksregal betrachtet, so nimmt man auch sehr stark grafische Muster auf dem Buchrücken wahr. Das Logo des Verlages wird meistens am Buchrücken aufgedruckt und soll dem Leser eine schnelle Zuordnung zum jeweiligen Verlag ermöglichen. Beispiele für solche charakteristischen Emblems sind z.B. das wohlbekannte blaue „L“ bei den

Langenscheidt-Wörterbüchern oder des Pferd mit Reiter bei den Büchern vom Springer-Verlag. Markenzeichen wie Logos prägen sich unbewußt beim Menschen ein und werden demnach auch beim Suchprozeß herangezogen. Neben diesem Aspekt verschönern Grafiken auch das optische Erscheinungsbild und steigern das realistische Aussehen eines Objekts wesentlich.

4.2.6 Text

Bei fast allen Dokumenten (deren Dicke es zuläßt) befindet sich am Einband textuelle Information über den Titel oder den Autor. Hat man die Suche nach einem Dokument schon auf einen Bereich (z.B. auf ein Kästchen eines Bücheregals) eingeschränkt, so zieht man diese Information sehr stark zur Entscheidungsfindung heran, ob ein Dokument relevant ist oder nicht. Neben dem Inhalt der Aufschrift spielt auch das Erscheinungsbild des Textes eine nicht unwesentliche Rolle. So werden große und fettgedruckte Texte viel intensiver wahrgenommen als kleinere Aufschriften. Weiters kann man auch mit der Wahl der Schriftart und ihrer Plazierung etliches bewirken. Wenn man bespielsweise den Titel immer in der gleichen Schriftart gestaltet und an derselben Stelle am Buchrücken aufdruckt, weiß der Informationssuchende intuitiv nach einer gewissen Gewöhnungszeit, wohin er sein Augenmerk legen muß, wenn er die Relevanz der von ihm gesuchten Information anhand des Titels beurteilen will.

4.2.7 Neue Dokumente

Bücher und andere Objekte, die erst vor kurzer Zeit in eine Sammlung aufgenommen wurden, erkennt man meistens schon aus einer großen Distanz, da ihr Einband glänzender ist als bei älteren Werken. Die Ursache dafür ist, daß sich noch kein Staub auf der Oberfläche ablagern konnte und der Einband nicht durch Abnützung in Mitleidenschaft gezogen wurde, wie es bei häufig verwendeten Dokumenten oft der Fall ist. Methoden aus der Computergrafik, die Glanzeffekte und Reflektionen (Spiegelungen) visualisieren, können dazu verwendet werden, dieses Aussehen so realistisch wie möglich nachzubilden um neue Werke in einer Sammlung hervorzuheben.

4.2.8 Abnützung

Im Gegensatz zu den relativ neuen Dokumenten einer Bibliothek gibt es auch Bücher, die schon sehr lange Bestandteil dieser Sammlung sind und dementsprechend eindeutige Abnützungserscheinungen aufweisen. Ausgefranste Einbände, Fingerabdrücke, Schmutz und „Eselsohren“ sind für den

Betrachter ein Indiz für ein abgegriffenes, schon oft benütztes Dokument. Mit geeigneten Methoden aus der Computergrafik wie z.B. dem Texture-Mapping kann dieses Aussehen imitiert werden.

4.2.9 Staub

Üblicherweise sammelt sich auf Büchern im Laufe der Zeit jede Menge Staub an. Wenn nie abgestaubt wird, so vermehrt sich dieser Staub zusehends und verdeckt textuelle Information auf dem Einband. Die Farbe des Buches wird immer stumpfer und grauer. Im Extremfall bilden sich auch Spinnweben um den Einband herum, sollte diese Ressource schon seit einer Ewigkeit nicht mehr verwendet worden sein. Die Intensität der Staubschicht kann deswegen als Metapher dafür verwendet werden, wie lange ein Dokument nicht mehr referenziert wurde.

4.2.10 hervorstehende Bücher

Betrachten wir unser eigenes Bücherregal näher, so können wir erkennen, daß die darin enthaltenen Bücher keinesfalls dieselbe Position im Regal haben. Es ist vielmehr so, daß häufig benutzte Werke dazu tendieren, hervorzustehen und sich so von anderen Exemplaren visuell unterscheiden. Diese Matapher kann man sich beispielsweise zur Visualisierung der Relevanz bei Suchabfragen zunutze machen. Bei digitalen Bibliotheken, die eine Interaktion mit dem Benutzer erlauben und z.B. das Datum des letzten Referenzierens mitprotokollieren, kann diese Metapher ebenfalls sinnvoll eingesetzt werden.

4.2.11 Position im Regal

Wenn man das Bücheregal in mehrere kleinere Regalkästchen unterteilt, kann man durch Einsortieren der Werke in ein bestimmtes Kästchen eine Klassifizierung nach Themengebiet vornehmen. Wie bei konventionellen Bibliotheken können durch dieses Konzept verwandte Themengebiete nahe beieinander angeordnet werden, um dem Informationssuchenden schnell und intuitiv zur gewünschten Ressource zu führen.

4.3 Umsetzung im libViewer

In diesem Abschnitt zeigen wir, wie wir die von uns identifizierten Metaphern für Informationsressourcen im libViewer grafisch umgesetzt haben. Basierend auf diesen Metaphern kann ein Mapping definiert werden, das ähnlich der

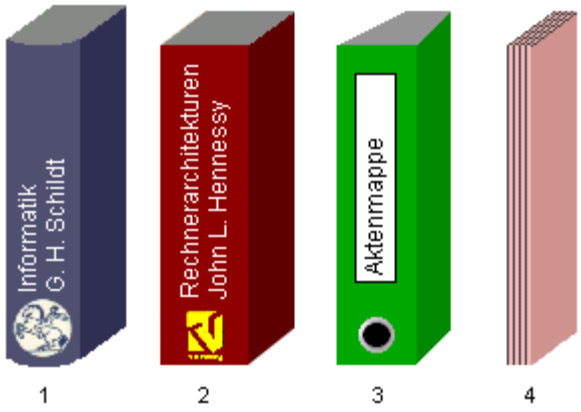

Abbildung 4.1: Dokumenttypen mit starkem Realitätsbezug

„Chernoff-Faces“ für multidimensionale Repräsentation von Räumen [3] die Metadaten-Attribute visualisiert um eine einfache Interpretation der Dokumente zu erlauben.

Wir haben versucht, jede Umsetzung der Metaphern so realistisch wie möglich zu gestalten. Durch die Einschränkung auf Rastergrafik und der sich daraus ergebenden Unschärfe konnten etliche Details nicht so umgesetzt werden, wie es unter Umständen sinnvoll und wünschenswert gewesen wäre. Es wurde auch absichtlich bei Proportionen übertrieben, um wichtige Merkmale zu betonen.

4.3.1 Typ

Abbildung 4.1 zeigt Beispiele, wie wir aus der Realität bekannte Dokumenttypen im libViewer umgesetzt haben.

Ganz links befindet sich ein Exemplar des Typs hardcover, wo man sofort das charakteristische Merkmal dieses Typs erkennt: den Rundrücken. Wir haben die Metapher des vorgewölbten Buchrückens deswegen gewählt, da der Betrachter durch diese Eigenschaft automatisch auf einen dicken und soliden Einband, wie man ihn bei einem Buch mit Harteinband findet, schließt.

Im Gegensatz dazu wirkt der Einband eines Taschenbuches, der meistens aus stärkerem Papier besteht, weniger widerstandsfähig und anfälliger auf Abnützungen. Um diesen Eindruck zu vermitteln, haben wir den Rundrücken des Typs hardcover beim Buch mit Papiereinband durch einen abgeflachten Rücken ersetzt, wie an einem Beispiel des libViewer-Objekttyps paperback (Objekt 2) verifiziert werden kann. Sowohl hardcover als auch paperback bieten die Möglichkeit, auf dem Buchrücken eine Grafik bzw. ein Logo darzu-

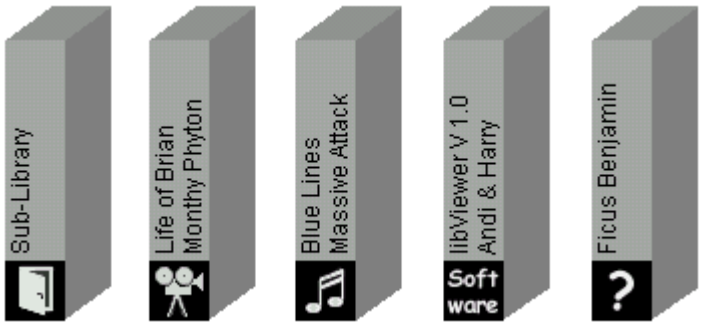

Abbildung 4.2: abstrakte Dokumenttypen

stellen, auf das beispielsweise der Verleger des Dokuments abgebildet werden kann. Auf die Metapher des Logos am Buchrücken wird in Abschnitt 4.3.6 aber noch näher eingegangen.

Das dritte Objekt in Abbildung 4.1 veranschaulicht die Visualisierung des Typs Ordner im libViewer. Es sticht sofort der nach hinten konisch zusammenlaufende Körper in das Auge des Betrachters. Wir haben diese Eigenschaft aus der Tatsache abgeleitet, daß Ordner in der Realität meistens weniger Blätter beinhalten als sie aufnehmen können und die Seitenteile daher nach hinten zusammenlaufen. Die zweite, sehr charakteristische Eigenschaft des libViewer-Objektyps folder ist das fix vorgegebene Logo, welches das bei Ordnern übliche „Guckloch“ andeuten soll. Als letztes Detail ist noch das Etikett zu erwähnen; Hier wurde die Textausgabe am Ordnerrücken (siehe auch Abschnitt 4.3.5) auf eine Zeile eingeschränkt (Ordner haben meistens nur eine Bezeichnung), ein weißer Untergrund gewählt und das Etikett schwarz umrandet. Durch Implementierung all dieser Details wurde versucht, dem Betrachter eine möglichst intuitive Visualisierung zu präsentieren, die er ohne Vorbildung sofort interpretieren kann und damit die Grafik eindeutig als Ordner identifiziert.

Als letzten Vertreter in der Reihe der Dokumenttypen, welche wir aus der Realität abgeleitet haben, zeigen wir eine Umsetzung des libViewer-Typs newspaper (Objekt 4). Vertikale Linien auf der Vorder- und Oberseite sollen hervorstehende Seiten andeuten, wie man sie bei Manuskripten oder Zeitungen häufig findet. Bei diesem Typ gibt es keine Mindestbreite, um den Gegensatz zwischen sehr dünnen Papers bzw. Schriftstücken und dicken, umfangreichen Zeitschriften oder Büchern akkurat visualisieren zu können. Auf eine Textausgabemöglichkeit am Rücken wurde verzichtet, da auch in der Realität Dokumenttypen wie z.B. Zeitungen keine - oder nur eine sehr kleine - Beschriftung an dieser Stelle aufweisen.

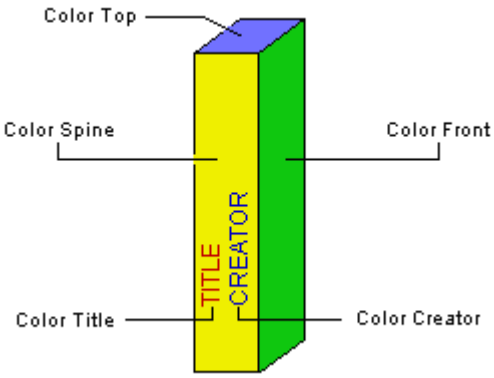

Abbildung 4.3: Farbe - Gestaltungsmöglichkeiten

Zu Dokumenten mit textuellen Informationen findet man relativ leicht eine physische Repräsentation (siehe Abbildung 4.1). Doch in digitalen Bibliotheken werden meistens auch multimediale Daten oder Verknüpfungen (Links) zu anderen Informationsbeständen abgelegt. Wir haben zu diesem Zweck insgesamt 5 Objekttypen im libViewer implementiert, um auch diese Informationstypen visualisieren zu können. In Abbildung 4.2 finden sie Beispiele für diese Objekttypen, wobei das Logo auf der Vorderseite als Metapher für den Typ der Ressource gewählt wurde. So erscheint für Ressourcen vom Typ Video eine Filmkamera auf der dem Benutzer zugewandten Seite der Box, Musiknoten für Audio-Informationen und der Schriftzug „Software“ für Anwendungen, Sourcecode oder ähnliches. Für Links zu anderen Informationsbeständen wie z.B. Unter-Bibliotheken (sublibraries) steht repräsentativ das Symbol einer Tür und alle anderen, nicht näher spezifizierten Elemente einer digitalen Bibliothek werden mit einem Fragezeichen visualisiert. Bei Bedarf können für diese Dokumenttypen auch realistische Metaphern gefunden werden, z.B. Videokassetten für Videos oder Tonbänder für Audioinformationen. Unser Hauptaugenmerk lag aber auf der Visualisierung textueller Informationsressourcen.

4.3.2 Farbe

Der libViewer bietet insgesamt 5 Elemente, die zur Codierung von Metainformationen verwendet werden können (siehe Abbildung 4.3:

- Color Spine (Farbe des Buchrückens)
- Color Top

Abbildung 4.4: Beispiele für Farbgebung

- Color Front
- Color Title
- Color Creator

Abbildung 4.4 zeigt 2 Beispiele, wie eindrucksvoll Farbe Meta-Informationen über ein Werk in einer digitalen Bibliothek transportieren kann. Wenn man beispielweise alle Werke, die vom Typ „dictionary“ sind und vom Verlag „Langenscheidt“ publiziert wurden, gelb coloriert, so kann der Betrachter möglicherweise auf den ersten Blick die Information ableiten, daß die dargestellte Grafik ein Langenscheidt-Wörterbuch repräsentiert. Voraussetzung dafür ist aber, daß der Anwender diese Assoziationen bereits vorher in der realen Welt schon getroffen hat, d.h. in diesem Fall mit einem gelben Bucheinband eben Langenscheidt-Wörterbücher verbindet. Assoziiert er dies nicht, so interpretiert er die gleiche Farbgebung jedenfalls als Metapher für eine Zusammengehörigkeit unter den Dokumenten. Ein weiteres Beispiel für die Zuordung von Metadaten-Attributen auf die Farb-Metapher ist die Diplomarbeit in Abbildung 4.4. Dort wurde für alle Werke vom Typ „thesis“ ein Mapping auf einen schwarz eingefärbten Einband definiert. Die Möglichkeiten, was man letztendlich auf die Farbe abbildet, sind vielfältig. So ist auch das Alter einer Ressource ein Kandidat für Abbildungen auf die Farbe eines Dokuments, indem man z.B. ältere Werke matter und neuere in kräftigeren Farben colorieren kann.

Der große Vorteil der Farb-Charakteristik ist es, daß der Betrachter auch in Übersichtsdarstellungen Ansammlungen von gleichfärbigen Dokumenten erkennt und dies zur Informationssuche verwenden kann. Aus diesem Umstand heraus ergibt sich aber auch ein nicht unwesentliches Problem. Bei zu

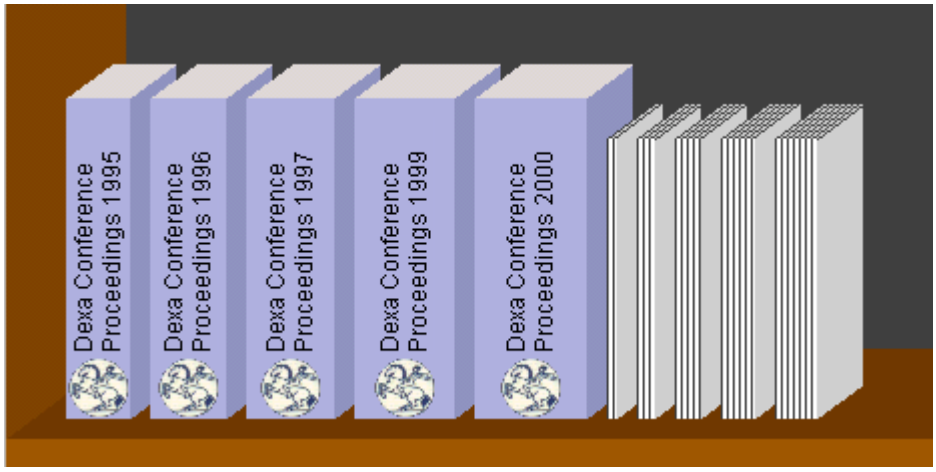

Abbildung 4.5: Beispiele für unterschiedliche Buchdicke

extensivem Einsatz von verschiedenen Farbcodierungen besteht die Gefahr der Erzeugung von grafischen Puzzles[26]. Würde man z.B. für eine große Anzahl von Verlegern eigene Farbmodelle implementieren, so verwirrt dies den Betrachter mehr als es ihm letztendlich zur Informationssuche dient, da er mit der Vielzahl von unterschiedlich gefärbten Dokumenten nichts mehr assoziieren kann.

4.3.3 Größe

Abbildung 4.5 zeigt, wie man die unterschiedliche Menge an enthaltener Information auf das Aussehen von Dokumenten abbilden kann. Wenn man die in dieser Grafik abgebildeten Dokumente untereinander vergleicht, so erkennt man sofort deren unterschiedliche Dicke. Die dünneren Dokumente scheinen wenige Seiten zu haben und werden daher intuitiv mit einer geringen Informationsmenge assoziiert. Hingegen deuten die dicken Bücher bzw. Papers auf eine hohe Seitenanzahl, was wiederum auf eine große Menge an enthaltener Information schließen läßt. Daher erscheint es uns als eine sinnvolle Methode, Metadaten-Attribute wie Seitenanzahl oder Größe in Bytes auf die Dicke des Dokuments abzubilden. Dabei ist aber die Wahl des Abbildungsverfahrens nicht unessentiell. Würde beispielsweise die Dicke proportional mit dem Wert der Seitenanzahl anwachsen, so könnten unter Umständen unrealistische Eindrücke entstehen. Diese Thematik wird aber noch detaillierter in Kapitel 7 beleuchtet.

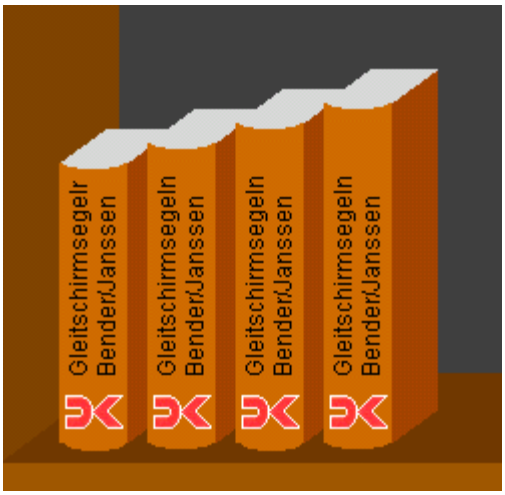

Abbildung 4.6: Beispiele für unterschiedliches Format

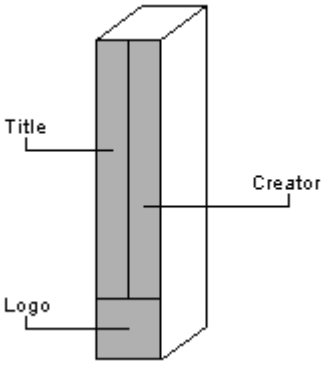

Abbildung 4.7: Elemente des Buchrückens

4.3.4 Format

Abbildung 4.6 zeigt anhand verschiedener Buchhöhen, wie eindrucksvoll unterschiedliche Formate auf den Betrachter wirken. So glaubt man im ganz linken Exemplar ein kleines Büchlein vorzufinden, welches man bei Reisen bequem in die Jackentasche stecken könnte. Hingegen wirkt das Buch mit der größten Höhe (ganz rechts in Abbildung 4.6) eher wie ein Atlas oder dergleichen.

Abbildung 4.8: derzeit vefügbare Logos im libViewer

4.3.5 Text

Der Buchrücken der libViewer-Dokumenttypen hardcover, paperback und media ist in drei Sektionen unterteilt: (a) den Bereich zur Ausgabe des Titels bzw. (b) des Autors (Creator) und einem quadratischen Bereich zur Visualisierung des Logos. Die Aufteilung und Gestaltung dieser Fläche lehnt sich stark an das Aussehen der in der Realität existierenden Buchrücken an, wo meistens der Titel vertikal aufgedruckt vorzufinden ist. Das Verlags-Logo befindet sich üblicherweise im unteren Bereich des Buchrückens. Es besteht also die Möglichkeit, in 2 Textzeilen kurze Zeichenketten auszugeben, die trotz ihrer vertikalen Ausrichtung auch bei flüchtiger Betrachtung noch einwandfrei erkennbar und vor allem lesbar sind.

4.3.6 Logos

Abbildung 4.8 zeigt alle derzeit verfügbaren Grafiken zur Darstellung des Logos am Buchrücken. Zeile 1 beinhaltet alle Verlags-Logos (von Springer bis Addison-Wesley), Zeile 2 und 3 alle Domain-Logos. Viele dieser Grafiken sind transparente gif-Bilder (graphics interchange format) und lassen so die Farbe des Buchrückens durchscheinen. Im Fall der Abbildung des Verlags-Emblems am Dokumentrücken bekommt der Betrachter die Möglichkeit, intuitiv eine Zuordnung zu dem jeweiligen Verlagshaus zu treffen. Durch die Visualisierung der Domain als Logo kann eine sofortige Klassifikation im Sinne des geografischen Aufenthaltsortes einer Ressource durch den Anwender erfolgen.

4.3.7 Effekt „highlighting“

Durch diese Metapher, welche in Abbildung 4.9 dargestellt ist, soll der Eindruck entstehen, daß das von links oben beleuchtete Dokument glänzt. Dabei

Abbildung 4.9: Hochglanz-Einband

Abbildung 4.10: „abgegriffenes“ Buch

wurde das Aussehen von Büchern mit Hochglanzeinband imitiert. Durch diese Metapher können beispielsweise neue Bücher einer Kollektion besonders hervorgehoben werden.

4.3.8 Effekt „Abgegriffen“

Woran erkennt der Betrachter, daß ein Buch „abgegriffen“ ist, d.h. schon sehr oft gelesen bzw. verwendet wurde? Wir haben bei unserer Untersuchung festgestellt, daß sogenannte „Eselsohren“, ausgefranste Einbände und Fingerabdrücke unbewußt auf den Menschen einwirken und diese Information transportieren. Da es relativ schwierig ist, Eselsohren und ausgefranste Buchkanten mit Computergrafik darzustellen, haben wir uns auf Fingerabdrücke spezialisiert. Abbildung 4.10 zeigt ein Exemplar eines abgegriffenen Buches, mit Fingerabdrücken links und rechts am Buchrücken.

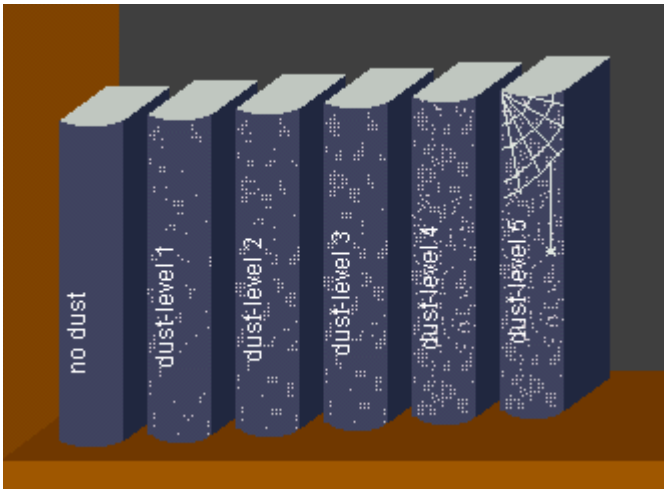

Abbildung 4.11: Staub - verschiedene Stufen

4.3.9 Staub

Insgesamt haben wir im libViewer fünf verschiedene Stufen von „Verstaubtheit“ zu realisieren versucht. Dabei werden unterschiedliche Texturen über den Buchrücken gelegt und sollen so einen staubigen Eindruck imitieren. Abbildung 4.3.9 zeigt alle Staubigkeitsstufen auf einen Blick - von links nach rechts steigert sich der Staubanteil. Ganz rechts befindet sich ein Buch mit der stärksten Staubstufe, welches zusätzlich noch von einem Spinnennetz umspannt ist.

4.3.10 Hervorstehen von Dokumenten im Bücherregal

Diese in Abschnitt 4.2.10 schon beschriebene Metapher wurde im libViewer so umgesetzt, daß Bücher in ihrer z-Position unterschiedliche Werte besitzen können und so im Regal hervorstehen oder nach hinten versetzt sind. In Abschnitt 6.2.1 sind Beispielgrafiken abgebildet, die einen guten Eindruck dieser Visualisierungstechnik liefern.

4.3.11 Regalkästchen

Wie in Abschnitt 4.2.11 schon erwähnt wurde, kann durch die Einteilung des Bücherregals in kleinere Kästchen sehr viel ausgedrückt werden. Für geclusterte Bibliotheksdaten bietet der libViewer eine eigene Darstellungsart, bei der das Bücherregal zusätzlich zu den horizontalen Regalbrettern auch noch vertikale Unterteilungen aufweist. Für Beispielgrafiken zu dieser Matapher

siehe Abschnitt 6.2.1.

4.4 Zusammenfassung

Im ersten Abschnitt dieses Kapitels wurden Designrichtlinien für Metaphern zur Repräsentation von Informationsressourcen vorgestellt. Unter Berücksichtigung dieser Vorüberlegungen haben wir ein Set von Metaphern kreiert, auf das im darauffolgenden Abschnitt 4.2 (Metaphern zur Visualisierung von Metadaten in digitalen Bibliotheken) detailliert eingegangen wird. Viele dieser Metaphern haben wir bei der Analyse von existierenden Bücherregalen im alltäglichen Leben (z.B. dem persönlichen Bücheregal) identifiziert. In Abschnitt 4.3 haben wir dann die Umsetzung der identifizierten Metaphern im libViewer beschrieben. Dieser Abschnitt beinhaltet neben zahlreichen Beispielgrafiken auch Überlegungen zur technischen Umsetzung mittels Computergrafik.

Kapitel 5

Technische Systembeschreibung

Dieses Kapitel beinhaltet eine vollständige Beschreibung aller Klassen des libViewer-libServer Systems. Dabei wurde versucht, Zusammenhänge zwischen Modulen zu verdeutlichen sowie die logischen Abläufe in den Funktionen zu erläutern.

Zweck dieser Klassenbeschreibung ist es, das libViewer-libServer System erweiterbar zu machen, d.h. jedem Entwickler das Basiswissen zum erfolgreichen Einstieg in das System zur Verfügung zu stellen. Dadurch wird es möglich, bespielsweise zusätzliche Server zu Bibliothekssystemen (z.B. dem OPAC-System der TU-Bibliothek Wien) zu implementieren, sowie die Funktionalität des libViewers zu erweitern.

5.1 Klassenbeschreibung libViewer

Der libViewer ist ein Java-Applet zur Visualisierung von digitalen Bibliotheken und wurde speziell für den Internet-Bereich entwickelt. Prämisse bei der Implemetierung war es, den Sourcecode möglichst schlank zu gestalten, d.h. hauptsächlich standardisierte Java-Klassen zu verwenden. Der Vorteil dabei ist, daß keine großen Klassenbibliotheken über das Internet auf den Client (d.h. in den Web-Browser) übertragen werden müssen. Diese Philosophie war nicht immer einfach zu verwirklichen, spiegelt sich aber jetzt in einer kurzen Download-Zeit des Applets wieder. Zum Zeichnen der Grafiken wurden ausschließlich standardisierte Java-Zeichenfunktionen verwendet.

Der libViewer stellt in der Architektur des Systems das Frontend zur Visualisierung der Bibliotheksdaten dar. Mittels Kommunikation über Sockets zu verschiedenen libServern werden vorbereitete Daten im libViewer-Format eingelesen und visualisiert (siehe Abbildung 5.2). Es wird zuerst ein Request an den korrespondierenden *libServer* gesendet, der beispielsweise die Form

einer Suchabfrage (z.B. einer Query an die Suchmaschine AltaVista) oder den Filenamen einer digitalen Bibliothek (z.B. eine Dublin-Core Library) haben kann. Nachdem der Server die Daten vorbereitet hat, werden diese empfangen und im Ausgabefenster dargestellt. Der Anwender hat dann die Möglichkeit, durch die Bibliothek zu navigieren oder nach bestimmten Metadaten-Attributen zu sortieren.

In den folgenden Abschnitten wird jede für den libViewer notwendige Klasse detailliert beschrieben und soll so einen funktionellen Überblick über die Architektur des libViewers liefern. Die beschriebenen Klassen sind in alphabetischer Reihenfolge angeführt.

5.1.1 class `Action`

Diese Klasse stellt eine Datenstruktur zur Verfügung, die Parameter für den Verbindungsaufbau zu den einzelnen Servern speichert. Diese Parameter entsprechen der Definition des libViewer-libServer Protokolls (Version 1.1) und beinhalten für jede Verbindung zu einem libServer:

- URL
- Port
- Description
- Data
- But1
- But2
- But1Txt
- But2Txt
- Reserved

Für detaillierte Informationen zum libViewer-libServer Protokoll siehe Abschnitt 5.4.

5.1.2 class `ActionList`

Mit dieser Klasse werden die einzelnen Verbindungen zu den Servern (class `Action`) dynamisch in einer Liste verwaltet. Folgende Methoden wurden dafür implementiert:

- function `addAction`: Fügt ein Element der Klasse Action zur Liste hinzu.
- function `selectAction`: Wird aufgerufen, wenn der Anwender im libViewer eine neue Verbindung auswählt, und befüllt die diversen Controls mit den entsprechenden Daten.
- function `setupServerChoice`: Befüllt bei der Initialisierung des libViewer-Applets die Choice-Box für die verschiedenen Server-Verbindungen.

5.1.3 class `Book`

Stellt die Überklasse für alle derzeit im libViewer implementierten Dokumenttypen dar. Abbildung 5.1 veranschaulicht die Objekthierarchie der Dokumenttypen grafisch. Die Klasse `Book` selbst ist von der Java-Klasse `Canvas` abgeleitet, die einen rechteckigen Bereich zum Zeichnen von Grafiken zur Verfügung stellt. Diese rechteckigen Bereiche werden beim Zeichnen der Bibliothek einem übergeordneten Zeichenbereich (Java-Klasse `Panel`) hinzugefügt und visualisieren nur den Buchrücken. Die für das dreidimensionale Aussehen zuständigen Flächen (Seiten- und Oberteil) werden mit den von Java zur Verfügung gestellten Zeichenfunktionen realisiert und stehen in keinem direkten Zusammenhang mit den Instanzen der `Book`-Klasse. Folgende Methoden werden von dieser Klasse vererbt:

- function `createTitleArea`: Erzeugt ein rechteckiges Image, das den Titel und Autor des Dokuments beinhaltet. Titel und Autor werden hierfür mittels der Java-Klasse `RotateFilter` um 90 Grad gedreht, um eine vertikale Ausrichtung der Textinformation am Buchrücken zu erreichen.
- function `DrawDust`: Wird aufgerufen, wenn das Dokument Staub auf dem Buchrücken enthalten soll. Derzeit werden insgesamt 5 Levels an Staub unterstützt (von leichtem Staub bis hin zu Spinnweben), wobei eine vorgefertigte Bitmap, die das Aussehen von Staub imitieren soll, als Textur über den Buchrücken gelegt wird. (siehe auch 4.3.9)

- function `DrawWellthumbed`: Diese Funktion wird aufgerufen, wenn das Dokument abgegriffen erscheinen soll. Dabei wird wie bei der Funktion `DrawDust` eine vorgefertigte Textur, die Fingerabdrücke darstellen soll, am linken und rechten Rand des Buchrückens gezeichnet. (siehe auch 4.3.8)

- function `DrawHighlighting`: Diese Funktion stattet neue Dokumente mit einem Glanzeffekt aus. Hierbei wird der RGB-Wert der Farbe des Buchrückens mittels eines Verfahrens um einen definierten Wert erhöht und mit dieser Farbe die linke und obere Kante des Dokumentrückens eingefärbt. Dadurch soll beim Betrachter der Eindruck entstehen, daß das von links oben beleuchtete Dokument glänzt. (siehe auch 4.3.7)

- `Event-Handler`: Der Event-Handler der Klasse `Book` verarbeitet Aktionen, die der Anwender mit der Maus über bzw. in dem rechteckigen Bereich des Buchrückens ausführt. Jedes im libViewer visualisierte Dokument erbt von der `Book`-Klasse diesen Event-Handler. Daher können auf einfache Art und Weise auf das spezielle Dokument bezogene Aktionen realisiert werden. Wird die Maus beispielsweise über einen Buchrücken bewegt, so wird das Detail-Window automatisch mit den Metadaten des Dokuments aktualisiert. Bei einem Klick mit der linken Maustaste auf einen Dokumentrücken wird bei Darstellung mit 33% (Übersichtsdarstellung) hineingezoomt und bei einem Klick mit der rechten Maustaste wird ein neues Browser-Window mit der URL aus dem Metadaten-Attribut Identifier gestartet.

5.1.4 class `Details`

Diese Klasse wird von der Java-Klasse `Frame` abgeleitet und beinhaltet die graphischen Elemente (Labels und Textdisplays) für die einzelnen Metadaten, die im Detail-Window angezeigt werden. Zusätzlich findet sich auch ein Button zum Starten eines externen Viewers (derzeit wird ein neues Browser-Window mit der URL des Metadaten-Attributs Identifier des aktuellen Dokuments geöffnet).

5.1.5 class `DrawArea`

Diese von der Java-Klasse `Panel` abgeleitete Klasse ist das Kernstück des libViewers und stellt den Bereich zur Verfügung, in dem die Elemente der Bibliothek grafisch ausgegeben werden. Immer wenn sich der Inhalt des Ausgabefensters ändert, wird eine neue Instanz des Objekts `DrawArea` erzeugt

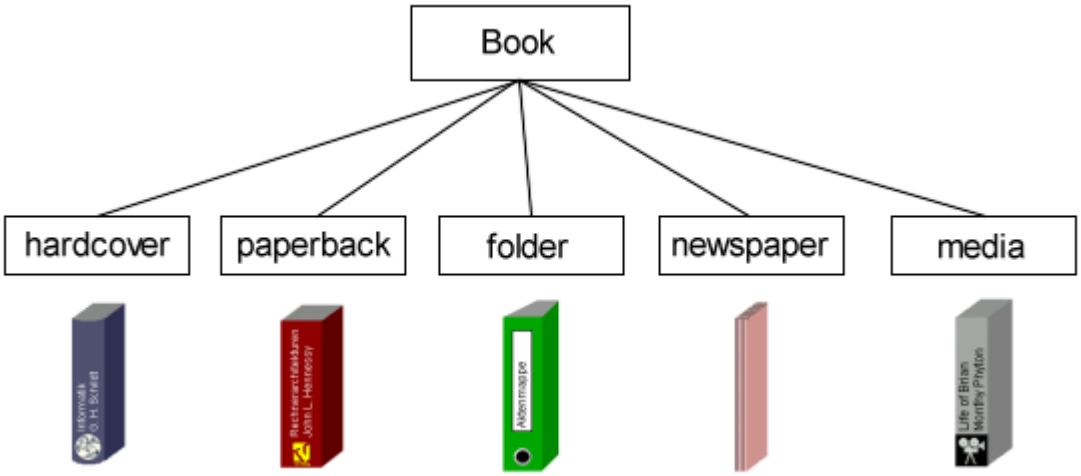

Abbildung 5.1: Objekthierarchie der Dokumenttypen

und dem Constructor die Liste der Bibliotheks-Elemente übergeben. Das Zeichnen der Dokumente wird von mehreren Funktionen bewerkstelligt:

- function `paint`: Diese Funktion zeichnet die Elemente der Liste sequentiell in das Ausgabefenster. Dabei werden folgende Schritte ausgeführt:

  1. Je nach Auswahl des Zoomfaktors (33% oder 100%) Setzen von Steuervariablen, um die unterschiedlichen Größenverhältnisse abzubilden.
  2. Aufruf der Sub-Funktion `DrawShelf`, um das Regal zu zeichnen. Im Fall der 100%-Darstellung wird das Fenster in 2 Regalkästen eingeteilt, bei 33% in 5 Kästen.
  3. Ausführen einer Schleife, die von dem vom Benutzer ausgewählten Startelement der Liste solange Dokumente zeichnet, bis das Ausgabefenster voll ist. Die Zeichenroutine beginnt immer im obersten Regalkasten und zeichnet die Dokumente von links nach rechts auf das Regalbrett. Sobald ein Kästchen bis zum rechten Rand mit Dokumenten vollgefüllt ist, erfolgt ein (Zeilen-)Umbruch in das darunterliegende Kästchen. Sind alle Kästen ausgefüllt oder ist das Ende der Dokumentliste erreicht, wird die Schleife verlassen. Im Schleifenrumpf werden folgende Schritte ausgeführt:

     (a) Zeichnen der Labels am Regalbrett. Am linken Rand eines Regalkästchens wird der Titel des ersten Dokuments und seine

Position in der Bibliothek als Text ausgegeben. Am rechten Rand analog dazu der Titel und die Position des letzten Elements des Regalkästchens.

(b) Berechnen der Shift-Werte (Aufruf der Funktion `CalcShiftValues`, siehe unten)

(c) Hinzufügen des Dokuments zum Ausgabefenster. Je nach Buchtyp (paperback, hardcover, newspaper, folder, media) wird eine Instanz des jeweiligen Objekts, die einen `Canvas` (grafischer Bereich) darstellt, erzeugt und mit der Java-Funktion `Container.add` dem DrawArea-Window hinzugefügt.

(d) Herstellen des 3D-Effekts. Wie schon erwähnt wurde, wird durch das Hinzufügen der einzelnen Elemente als `Canvas` nur der Buchrücken gezeichnet. Um den Dokumenten jetzt ein dreidimensionales Aussehen zu verleihen, muß noch das Seiten- und Oberteil gezeichnet werden. Diese Funktion wird unten noch detaillierter beschrieben.

(e) Aufruf der Funktion `DrawConvexity`. Im Falle eines hardcover-Buches muß die Vorwölbung des Buchrückens gezeichnet werden.

(f) Neuberechnung der Zeichenposition. Die x- und y-Position, wo der Canvas des Buchrückens relativ zum Ausgabefenster gezeichnet wird, wird bei jedem Schleifendurchlauf neu berechnet.

- function `draw3DEffect`: Um dem Dokument ein dreidimensionales Aussehen zu geben, wird mittels der Java-Zeichenfunktion `fillPolygon` ein Seiten- und Oberteil in den entsprechenden Farben gezeichnet. Im Fall des Dokumenttyps newspaper werden zusätzlich vertikale Linien am Oberteil hinzugefügt, um hervorstehende Seiten zu imitieren.

- function `drawConvexity`: Diese Funktion wird nur bei hardcover-Büchern aufgerufen und zeichnet die Ausbuchtung am Buchrücken, um einen Rundrücken darzustellen. Je nach Dicke des Buches wird eine mehr oder weniger gestreckte Ellipse mittels der Java-Zeichenfunktion `fillOval` über den Buchrücken gelegt.

- function `CalcShiftValues`: Diese Funktion berechnet, abhängig vom `ZPOS`-Wert (siehe Abschnitt 5.4), die Veränderung der aktuellen Zeichenposition des Elements in z-Richtung. Durch diese Veränderung

der x- und y- Position des Dokuments wird ein Hervorstehen bzw. ein Einrücken einzelner Dokumente erreicht.

- function `DrawShelf`: Abhängig vom Zoomfaktor wird durch diese Funktion das Regal in das Ausgabefenster gezeichnet. Im Fall von 100% wird das Ausgabefenster in 2 Regalkästen eingeteilt, bei der Übersichtsdarstellung (33%) werden 5 Regalbretter gezeichnet.

5.1.6 class `DrawAreaXY`

Diese Klasse erweitert die Klasse `DrawArea` um Funktionen zur Visualisierung von in mehrere Regalabschnitte einsortierte Bibliotheksdaten. Beinhaltet jedes Element der vom libServer gelieferten Dokumentliste x- und y-Koordinaten, wird automatisch eine Instanz dieses Objekts erzeugt und als Ausgabefenster verwendet. Der Constructor dieser Klasse bekommt als Argument keine Element-Liste, sondern ein zweidimensionales Raster (libViewer-Klasse `Map`) übergeben. Die Segmente des Rasters bewirken im Gegensatz zur sequentiellen Darstellung durch die Klasse `DrawArea` im Ausgabefenster eine zusätzliche horizontale Unterteilung in den Regalkästen. Es wird intern ein virtuelles Fenster in x * y - Dimensionalität der Karte aufgebaut, wobei im Ausgabefenster immer nur ein Ausschnitt derselben angezeigt wird. In diesem Fall wird nicht, wie in der Klasse `DrawArea`, auf die aktuelle Startposition in der Dokumentliste zugegriffen, sondern auf das vom User-Interface bestimmte Regalkästchen im Raster referenziert. Dies gibt dem Anwender die Möglichkeit, das Ausgabefenster in jeder Richtung innerhalb der Grenzen des (größeren) virtuellen Fensters zu bewegen. Die Funktion `drawShelfBorder` zeichnet die für diese Darstellungsart notwendigen vertikalen Regalbretter zur Unterteilung der Regalkästen in horizontaler Richtung. Anstatt der Labels für Titel und aktuelle Position eines Elements in der Dokumentliste werden hier die x- und y-Koordinaten des Knotens auf den Regalbrettern ausgegeben. In nachfolgenden libViewer-Versionen soll diese Beschriftung durch Labels zur Beschreibung der einzelnen Segmente des Rasters ersetzt werden.

5.1.7 class `ElementData`

Diese Klasse repräsentiert den Datentyp zur Speicherung der libViewer-internen Dokumentattribute, die für deren Visualisierung notwendig sind. Für jedes von einem `libServer` zurückgelieferten Dokument wird eine Instanz des Objekts `ElementData` angelegt und mit den entsprechenden Daten befüllt. Die Attribute entsprechen den im libViewer-libServer Protokoll (siehe auch Abschnitt 5.4) definierten Attributen für den Datenaustausch zwischen

einem `libServer` und dem empfangenden libViewer und wurden um für die Programmlogik notwenige Variablen erweitert:

- int `spine_width`: Breite des Buchrückens in Bildpunkten
- int `spine_height`: Höhe des Buchrückens
- int `Key`: Variable für die Programmlogik, wird beim Zoom-In, d.h. wenn der Anwender von der Übersichtsdarstellung in die Detailansicht wechselt, verwendet um die neue Startposition innerhalb der Dokumentliste für die neue Ansicht zu generieren.
- Attribute analog zum Dublin-Core Metadaten Set:
  - String `title`
  - String `creator`
  - String `subject`
  - String `description`
  - String `publisher`
  - String `contributor`
  - String `date`
  - String `type`
  - String `format`
  - String `identifier`
  - String `source`
  - String `language`
  - String `relation`
  - String `coverage`
  - String `rights`
  - String `domain`
- int `page_size`: Größe des Dokuments in Seiten
- int `byte_size`: Größe des Dokuments in Bytes
- int `xpos`: x-Position des Dokuments in der som (self organizing map)
- int `ypos`: y-Position des Dokuments in der som (self organizing map)

- Float `zpos`: z-Position des Dokuments im Regal, d.h. der Wert gibt an, wie sehr ein Dokument hervorsteht
- Image `Logo`: Image am Buchrücken
- boolean `Logo_available`: Flag, ob das Logo verfügbar ist
- String `datatype`: Datentyp des Elements (hardcover, paperback,...)
- int `dust_level`: Je nach Dustlevel (1-5) werden 5 unterschiedliche Grafiken als Textur über den Buchrücken gelegt, die staubiges Aussehen imitieren sollen
- Color `surface_color`: Farbe des Seitenteils
- Color `surface_color2`: Farbe des Oberteils
- Color `spine_color`: Farbe des Buchrückens
- Color `title_color`: Farbe des Textes für den Titel
- Color `creator_color`: Farbe des Textes für den Autor
- boolean `wellthumbed`: Flag, ob das Dokument abgegriffen gezeichnet wird
- boolean `highlighting`: Flag, ob das Dokument einen Glanzeffekt erhält

5.1.8 class `folder`

Diese Klasse wird von der Überklasse `Book` abgeleitet und repräsentiert den libViewer-internen Dokumenttyp `folder`. Überschrieben wird bei diesem Typ die Methode `createTitleArea` der `Book`-Klasse, da für Ordner nur die Möglichkeit der Ausgabe eines Textes (Ordnerbezeichnung) am Buchrücken vorgesehen ist und die Beschriftung das Aussehen eines Etiketts bekommt. Die Funktion `paint` setzt das Aussehen des Buchrückens zusammen und wird wie folgt abgearbeitet:

1. Farbe für Buchrücken setzen
2. Rechteck für Buchrücken zeichnen
3. Logo für Loch einfügen, im Fall des Ordners standardmäßig das Image für das „Guckloch“

4. Image für Title und Creator einfügen

5. Wenn Flag `wellthumbed` gesetzt, Images fuer „abgegriffen“ einfügen

6. Images für Staub einfügen, wenn Dust-Level entsprechend gesetzt ist

7. Wenn Flag `highlighting` gesetzt, dann zeichne Highlighting-Effekt

Der Grundaufbau der Mothode `paint` ist bei allen libViewer-internen Dokumenttypen (hardcover, paperback, folder, newspaper, media) ähnlich und wird daher nur bei dieser Klasse detailliert beschrieben.

5.1.9 class `hardcover`

Abgeleitet von der Überklasse `Book` wird diese Klasse zum Darstellen von hardcover-Büchern verwendet.

5.1.10 class `ItemList`

Diese Klasse wird zur Verwaltung der Elemente in der Dokumentliste verwendet. Dies wird durch ein lineares Array erreicht, wobei die Einträge Objekte des Klasse `ElementData` sind.

5.1.11 class `Logos`

Da zusätzlich zu den Namen der Logos am Dokumentrücken, welche von den libServern an den libViewer gesendet werden, auch eine Pfadangabe zu den eigentlichen Image-Dateien auf dem Filesystem des Hosts (Ursprung des libViewer-Applets) benötigt wird, wurde für diesen Zweck eine Datei entworfen, die eine solche Zuordnung vornimmt. In dieser Datei, die von den libViewer-Clients vom Host geladen wird, sind paarweise Logonamen und zugehörige Filenamen für alle von der jeweiligen libViewer-Version unterstützten Logos abgelegt. In der Klasse `Logos` befindet sich eine Laderoutine, die diese Mappings in eine Liste einliest. Diese Liste wird in der Klasse `Main` dazu verwendet, alle verfügbaren Grafiken vom Host einzulesen.

5.1.12 class `Main`

Diese Klasse wird beim Starten des Applets zuerst aufgerufen und ist von der Java-Klasse `Applet` abgeleitet. In dieser Klasse befinden sich sämtliche GUI-Komponenten für das Layout des libViewers, Event-Handler zur Abarbeitung

von Benutzeraktionen sowie diverse globale Variablen für die Programmlogik. Wir möchten im Anschluss die wichtigsten Methoden dieser essentiellen libViewer-Klasse im Überblick beschreiben:

- function `jbInit`: Fügt alle grafischen Komponenten mittels der Java-Funktion `Container.add` dem Applet hinzu. Dabei werden auch Auswahlboxen wie die Choice-Box für die Sortierkriterien befüllt. Weiters werden alle Grafiken geladen, z.B. die Texturen zur Imitierung der Eigenschaft „abgegriffen“ oder dem Staub, sowie alle Logos für die einzelnen Publisher und Domains.

- function `loadLogos`: Lädt alle Logos vom Host, von dem das Applet gestartet wurde.

- function `Zoom_in`: Ruft die Funktion `refreshWindow` mit der neuen Startposition in der Liste und Einstellung 100% auf.

- function `handleEvent`: Dieser Event-Handler arbeitet alle Benutzeraktionen innerhalb der Oberfläche des libViewers ab. Folgende Events werden behandelt:

  - Der Anwender ändert den Zoomfaktor. Hier wird die Funktion `refreshWindow` mit dem veränderten Zoom-Wert (33% oder 100%) aufgerufen.
  - Der User betätigt einen der beiden Funktionsbuttons, um eine Anfrage an einen libServer zu senden. Dabei wird die Funktion `processRequest` der Klasse `Networking` aufgerufen und dieser der Funktionscode des aktivierten Buttons übergeben. Diese Funktion kontaktiert einen libServer und empfängt anschließend die vom Server zurückgesendeten Daten. Wenn die vom libServer erhaltenen Daten x- und y-Koordinaten aufweisen, d.h. thematisch sortierte Daten sind, so werden im rechten unteren Teil der Oberfläche 4 Buttons sichtbar, die zur Navigation innerhalb des Bücherregals dienen.
  - Der Anwender wählt einen neuen Server in der Server-Auswahlbox aus. In diesem Fall wird die Funktion `selectAction` der Klasse `ActionList` aufgerufen, die alle grafischen Elemente bzw. Variablen mit den neuen Serverdaten (URL und Port des Servers, Belegung der Funktionsbuttons mit entsprechenden Funktionscodes und Beschriftung derselben) bestückt.

- Betätigung der Buttons `start` bzw. `next`. Der Button `next` bewirkt, daß innerhalb der Dokumentliste um den derzeit dargestellten Bereich weitergesprungen wird, d.h. das Dokument, das vorher am Ende der angezeigten Dokumentliste gestanden hat, bildet jetzt den Anfang der Liste. Hier wird wiederum die Funktion `refreshWindow` aufgerufen, die das neue Ausgabefenster mit der geänderten Startposition innerhalb der Dokumentliste erstellt. Der Button `prev` bewirkt, daß an den Anfang der Dokumentliste zurückgesprungen wird.
- Aktivierung der Buttons `up`, `down`, `left` oder `right`. Diese Navigationsbuttons werden immer nur dann angezeigt, wenn die eingelesenen Dokumente x- und y-Koordinaten aufweisen, d.h. geclustert sind. Betätigt der Anwender einen dieser Buttons, so wird die Funktion `refreshWindow` mit dem neuen Startknoten aufgerufen und das Ausgabefenster innerhalb des virtuellen Bereichs (siehe auch Abschnitt 5.1.6) neu positioniert. Der Startknoten stellt immer das ganz links oben im Ausgabefenster abgebildete Regalkästchen dar.
- Checkboxen zum Anzeigen des Detail-Fensters bzw. der Konsole. Wird die Checkbox `show console` aktiviert, erscheint ein zusätzliches Fenster am Bildschirm, in dem die verfügbaren Metadaten eines Dokuments angezeigt werden. Der Inhalt dieses Fensters ändert sich immer dann, wenn der Anwender die Maus über einen Dokumentrücken bewegt. Das durch die Checkbox `show console` aktivierbare Ausgabefenster zeigt Status- und Fehlermeldungen des libViewers an.
- Sortieren der Dokumentliste. In einer Auswahlbox werden alle möglichen Sortierkriterien wie z.B. Sortierung nach Titel, Autor, Größe des Dokuments, usw. verwaltet. Wird nach Einstellen eines Sortierkriteriums der Button `Sort Library` betätigt, sortiert der libViewer die Dokumentliste aufsteigend nach diesem Kriterium. Dies ist derzeit nur bei nicht geclusterten Dokumenten möglich. Nach dem Sortiervorgang wird automatisch die Funktion `refreshWindow` augerufen und damit das Ausgabefenster neu aufgebaut.

- function `refreshWindow`: Diese Funktion wird immer dann aufgerufen, wenn sich der Inhalt bzw. das Aussehen des Ausgabefensters ändern soll. Dabei wird das alte Fenster (wenn vorhanden), das eine Instanz der Klasse `DrawArea` bzw. `DrawAreaXY` ist, mittels der Java-Funktion

`Container.remove` von der Oberfläche entfernt und anschließend eine neue Instanz des jeweiligen Objekts hinzugefügt.

5.1.13 class `Map`

Diese Klasse wird dazu verwendet, geclusterte Bibliotheksdaten in einer Karte abzulegen. Die Datenstruktur ist ein zweidimensionales Array in x- und y-Dimension. Jeder Knoten dieses Arrays kann beliebig viele Dokumente in einer Liste speichern. In dieser Klasse befinden sich auch einige Methoden zur Ermittlung benötigter Werte für die Zeichenroutine, z.B. die größte Dokumentanzahl in einem Regalkästchen.

5.1.14 class `media`

Abgeleitet von der Überklasse `Book` wird diese Klasse zum Darstellen aller Dokumenttypen, die nicht in Textform vorliegen, verwendet. Zu diesen Typen zählen Software, Audio- und Video, Links zu Subblibliotheken und alle anderen, nicht näher spezifizierbaren Ressourcen. Je nach Art der Information wird bei den einzelnen Typen standardmäßig ein Logo am Dokumentrücken dargestellt, z.B. bei Videos eine Filmkamera oder bei Audio-Daten Musiknoten.

5.1.15 class `Networking`

Diese Klasse stellt alle für die libViewer-`libServer` Interaktion notwendigen Funktionen zur Verfügung. Die Hauptfunktion der Klasse ist die Funktion `processRequest`, deren Ablauf im folgenden beschrieben wird.

1. Aufbau der Socketverbindung zwischen libViewer und libServer.
2. Öffnen der Input- und Outputstreams für den Datenaustausch. (siehe Abbildung 5.2)
3. Hat der Aufbau der Verbindung funktioniert, sendet der libViewer über den Outputstream die Anfrage an den Server. Dabei werden der Funktionscode, der Text (z.B. die Query oder der Filename der zu ladenden Bibliothek) und der momentan noch nicht verwendete reservierte Text an den korrespondierenden libServer übermittelt.
4. Schleife zum Empfangen der Dokumentdaten. Die Funktion wartet auf die Antwort des Servers über den Inputstream und liest Zeile für Zeile die Ausprägungen der einzelnen Elemente. Die Attribute, welche der

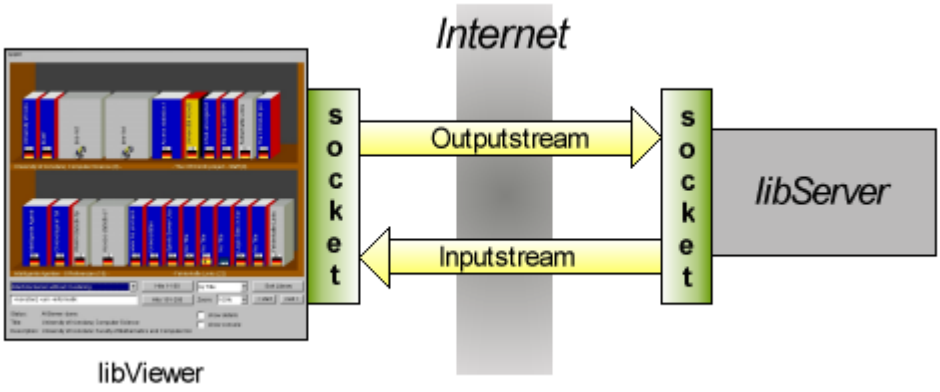

Abbildung 5.2: Schematische Darstellung der libViewer-libServer Kommunikation

libViewer einlesen kann, sind im libViewer-libServer Protokoll (siehe Abschnitt 5.4) definiert. Die empfangenen Daten werden in der Dokumentliste abgelegt, die eine Instanz der Klasse `ItemList` darstellt. Beendet wird die Schleife, wenn der Server die Kommunikation beendet, d.h. vom InputStream end-of-file (EOF) empfangen wird.

5. Schließen der Socketverbindung und des Input- bzw. Outputstreams.

6. Aufruf der Funktion `check_xy_parameters`, die überprüft, ob die empfangene Dokumentliste geclusterte Daten enthält. Sollten alle empfangenen Objekte korrekte x- und y-Koordinaten aufweisen, wird für die Anzeige der Bibliothek die „shelf-representation“ (Darstellung mit Regalkästchen) verwendet.

5.1.16 class `newspaper`

Abgeleitet von der Klasse `Book`, wird diese Klasse für den libViewer-Dokumenttyp newspaper verwendet. Bei diesem Typ werden keine Grafiken und auch kein Text am Buchrücken gezeichnet. Anstatt dessen werden vertikale Linien am Dokumentrücken und am Oberteil gezeichnet um hervorstehende Seiten zu imitieren.

5.1.17 class `paperback`

Repräsentiert den libViewer-internen Dokumenttyp paperback und ist wie alle Typen von der Oberklasse `Book` abgeleitet.

5.1.18 class `PrefsLoader`

Diese Klasse beinhaltet die Laderoutine für die einzelnen Serververbindungen. Es werden sequentiell von einer Datei alle zur Interaktion mit einem libServer notwendigen Attribute (URL, Port, Beschriftung der Funktionsbuttons und deren Funktionscodes und der Default-Text für die Eingabezeile) eingelesen und anschließend in der von der Klasse `Action` bzw. `ActionList` zur Verfügung gestellten Datenstruktur abgelegt.

5.1.19 class `RotateFilter`

Diese von Sun Microsystems entwickelte Klasse stellt Funktionen zur Rotation von Grafiken zur Verfügung. Wir benötigen diese Funktionen zum Rotieren der Textinformation am Buchrücken, da die Standard-Textzeichenfunktionen von Java nur horizontal ausgerichteten Text zeichnen können. Der als Grafik vorliegende horizontale Text wird um 90 Grad gedreht, um den in der Realität vertikal am Buchrücken befindlichen Text darstellen zu können.

5.1.20 class `Settings`

In dieser Klasse werden alle zur Programmsteuerung notwendigen Konstanten definiert und mit Werten belegt.

5.2 Klassenbeschreibung Dublin Core Server

Der Dublin-Core Server ist ein Java-Programm, das digitale Bibliotheken im Dublin-Core Format einlesen (Klasse `FileReader`), in das libViewer-interne Objektbeschreibungsformat konvertieren (Klasse `Converter`) und an den jeweiligen korrespondierenden libViewer zurücksenden kann (Klasse `DCServerMulti`). Abbildung 5.3 stellt die Kommunikation zwischen libViewer und DCServer schematisch dar.

Die Abarbeitung der einzelnen Requests an den DC-Server erfolgt parallel, da für jede Anfrage ein eigener Thread kreiert wird. Die Kommunikation wird mittels Sockets bewerkstelligt und hat für beide Richtungen entsprechende Input- und Outputkanäle. Die digitale Bibliothek kann (a) vom Filesystem, wo der DCServer gestartet wurde, oder (b) durch Angabe einer URL aus dem Internet geladen werden. Durch die Implementierung von verschiedenen Mappings (Publisher, Language, Publisher und Type, Type auf Farbgebung bzw. Logo am Buchrücken) kann das System flexibel auf den jeweiligen Anwendungszweck abstimmt werden.

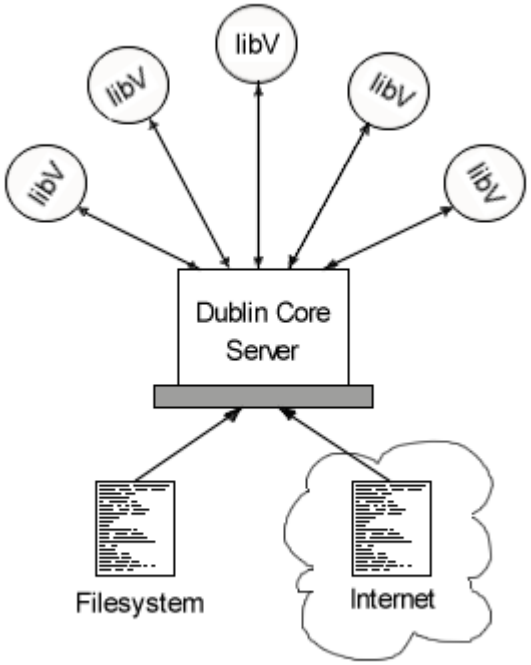

Abbildung 5.3: Schematische Darstellung der libViewer-DCServer Kommunikation

In den folgenden Abschnitten wird jede für den DCServer notwendige Klasse detailliert beschrieben. Es soll so einen funktionellen Überblick über die Architektur des Dublin-Core Servers liefern, wobei die Klassen in alphabetischer Reihenfolge angeführt sind.

5.2.1 class `Converter`

Diese Klasse beinhaltet das Konvertiermodul, welches im Dublin-Core Metadatenformat abgelegte digitale Bibliotheken in ein für den libViewer interpretierbares und darstellbares Format übersetzt. Dabei werden die einzelnen Metadatenattribute der Dokumente auf optische Merkmale der anzuzeigenden Elemente übergeleitet. Folgende Funktion bewerkstelligt diese Aufgabe:

- function `processItem`: Diese Methode erwartet als Eingabeparameter ein Objekt des Typs `Item`, das die im Dublin Core Metadatenformat definierten Attribute speichern kann. Sequentiell wird in dieser Funktion jedes der Attribute abgearbeitet, gegebenenfalls konvertiert und den Variablen des Objekts `ElementData` zugeordnet. Als Ausgabewert liefert die Funktion dann ein Objekt des Typs `ElementData`, dessen Daten anschließend dem libViewer zur Visualisierung gesendet werden. Der Ablauf kann schematisch folgendermaßen beschrieben werden:

  1. Zuweisen aller Attribute aus dem Dublin Core Metadaten Set zu den entsprechenden Variablen im Typ `ElementData`. Diese Daten werden vom libViewer 1:1 im Detail Window angezeigt (siehe 5.1.4).
  2. Mapping Publisher (Dublin Core) auf Logo ausführen. Dabei wird die Subfunktion `getPublisherLogo` aufgerufen, die alle im dafür vorgesehenen publisher-Preferences File erfaßten Mappings sequentiell nach einer Übereinstimmung hinsichtlich des Metadaten-Attributs `publisher` durchsucht und im Erfolgsfall die Variable `Logo` der Klasse `ElementData` mit dem korrespondierenden Logonamen belegt. Eine genaue Beschreibung dieser Mapping-Variante findet sich in Abschnitt 5.2.9.
  3. Mapping Type (Dublin Core) auf libViewer-Datentyp. Die in einem Preferences-File definierten Zuordnungen von Dublin-Core Dokumenttypen auf libViewer-interne Dokumenttypen (hardcover, paperback, newspaper, folder, media) werden mit Aufruf der Unterfunktion `getDatatype` sequentiell nach einem vorhandenen Mapping durchsucht. Sollte für den aktuellen Typ keine Überleitung definiert worden sein, bleibt die Variable `datatype` der

Klasse `ElementData` unbelegt. In diesem Fall wird der Default-Dokumenttyp zur Visualisierung im libViewer herangezogen.

4. Buchdicke ermitteln. Dabei wird je nach Verfügbarkeit der Wert aus dem Metadaten-Attribut `byte_size` oder `page_size` mit einem vordefinierten Faktor multipliziert und so die Breite des Buchrückens in Pixel berechnet. Im Fall des Dokumenttyps newspaper werden eigens nur für diesen Typ definierte Werte verwendet (unterscheidliche Mindestbreite, Skalierungsfaktor).

5. Buchhöhe bestimmen. Derzeit gibt es insgesamt 4 unterscheidliche Buchhöhen, die aus dem Format des Dokuments abgeleitet werden, falls dieses in der Realität in gedruckter Form vorliegt. Akzeptiert werden derzeit die Formatangaben A4, B4, A5 und B5.

6. Ermitteln, ob das Dokument abgegriffen erscheinen soll. Übersteigt die Zahl, wie oft auf das Dokument zugegriffen wurde (Metadaten-Attribut `DC.X-NUMBER_TIMES_REF`) einen definierbaren Schwellwert, so wird das Flag `wellthumbed` des Elements gesetzt.

7. Dust-Level bestimmen. Um zu ermitteln, ob das Dokument ein staubiges Aussehen bekommen soll, wird die Zeitspanne zwischen aktuellem Datum und dem Zeitpunkt des letzten Zugriffs (Metadaten-Attribut `DC.X-LAST_REF`) berechnet. Durch diese Zeitspanne wird dann bestimmt, wie stark das Dokument verstaubt erscheint. Dabei gibt es insgesamt 5 Stufen (Dust-Levels), die jeweils 1 Jahr umfassen. Stufe 1 wird beispielsweise dann ausgewählt, wenn die Zeitspanne ein Jahr nicht übersteigt (d.h. das Buch innerhalb des letzen Jahres referenziert wurde). Analog dazu bekommt ein Dokument mit Zeitspanne 5 oder mehr Jahren die Stufe 5 zugewiesen, die in der Visualisierung Spinnweben am Buchrücken bedeutet.

8. Ermitteln, ob das Dokument einen Hochglanzeffekt bekommen soll. Liegt das Datum, an dem das Dokument erstellt wurde, nicht mehr als ein Jahr zurück, dann wird das Flag `highlighting` gesetzt.

9. Farben (Color top, Color front, Color spine, Color Title, Color Creator) setzen. Es erfolgt der Aufruf der Unterfunktion `setColors`, wo schrittweise die Farben für das übergebene Dokument ermittelt werden. Eingangs setzt die Funktion die Farben der Flächen (Buchrücken, Seiten- und Oberteil) auf einen Default-

wert. Danach werden die in den Mappings (falls vorhanden) definierten Farbkombinationen je nach Priorisierung auf das Element übertragen. Folgende Mappings werden berücksichtigt:

- Typ auf Dokumentfarben
- Publisher (Verlag) auf Dokumentfarben
- Publisher und Typ gemeinsam auf Dokumentfarben
- Sprache auf Dokumentfarben

Je nach Priorisierung kommt ein Mapping früher oder später zur Anwendung, d.h. das zuletzt angewendete Mapping überschreibt eventuell schon vorher dem Element zugewiesene Farben. Das hat den Sinn, daß individuell auf den Anwendungszweck abgestimmte Farbmodelle realisiert werden können. So ist unter Umständen die Farbe, die sich aus der Sprache ergibt, wichtiger als die Farbe, die durch den Dokumenttyp bestimmt wird. Sollte nach Ablauf dieses Verfahrens die Farbe für die Textbausteine am Buchrücken noch unbelegt sein, so wird je nach Helligkeit des Dokumentrückens dessen Farbe automatisch bestimmt.

5.2.2 class `DCServer`

Durch Aufruf der Funktion `main` dieser Klasse wird der Dublin-Core Server gestartet. Zuerst werden alle Mappings geladen und ein Server-Socket auf einem vordefinierten Port geöffnet. In einer Endlosschleife wartet die Funktion dann auf die Kontaktaufnahme eines libViewers. Tritt dieses Ereignis ein, versucht der DCServer die Socketverbindung zu etablieren und erzeugt im Erfolgsfall einen neuen Thread, der durch die Klasse `DCServerMulti` implementiert wird und den Client-Socket als Eingabeparameter erwartet. Diese Technik hat den großen Vorteil, daß der DCServer parallel mehrere Anfragen bearbeiten kann.

5.2.3 class `DCServerMulti`

Diese Klasse wird von der Java-Klasse `Thread` abgeleitet und arbeitet den kompletten Request eines libViewer-Clients ab. Dafür werden zuerst die Kommunikationskanäle geöffnet, das sind ein `BufferedReader`, um den eingehenden Request empfangen zu können, und ein `PrintWriter`, um die Dokumentdaten an den libViewer zurücksenden zu können. Sind die Kommunikationskanäle aufgebaut, liest der DC-Server die erste Zeile des Requests und verzweigt im Fall eines gültigen Funktionscodes (derzeit nur „Laden einer Bibliothek“) in die Unterfunktion `LoadLibrary`.

Die zweite Zeile, die vom libViewer gesendet wird, enthält den Filenamen oder die URL der digitalen Bibliothek. Daher wird zuerst versucht, den String in eine URL zu übersetzen. Je nach Erfolg oder Mißerfolg dieser Operation wird in der Klasse `FileReader` die Funktion `readLibrary` aufgerufen, mit dem Argument, ob die Bibliothek vom lokalen Filesystem oder aus dem Internet über eine URL geladen werden soll.

Wenn die Funktion `readLibrary` eine gültige Bibliothek zurückliefert, wird jedes Dokument aus der geladenen Bibliothek in das libViewer-Dokumentformat konvertiert (Aufruf der Funktion `processItem` / Klasse `Converter`) und anschließend an den libViewer zurückgesendet. Vorausgeschickt wird `$ITEMCOUNT`, wobei das Argument die Anzahl der Dokumente angibt, die der libViewer empfangen wird. Wenn alle Elemente gesendet wurden, terminiert der Thread.

5.2.4 class `ElementData`

Der Aufbau dieser Klasse entspricht exakt dem der gleichnamigen libViewer-Klasse `ElementData` und wurde schon im Abschnitt 5.1.7 beschrieben.

5.2.5 class `FileReader`

In der Klasse `FileReader` sind Funktionen zum Einlesen der digitalen Bibliotheken im Dublin-Core Format enthalten. Die Hauptfunktion ist `readLibrary`, die Dateien vom WWW (über eine URL) oder vom Filesystem des DCServers (über einen Filenamen) lesen kann. Genau genommen werden die Daten zeilenweise eingelesen, geparst und anschließend in einer Dokumentliste (Klasse `ItemList`) abgespeichert. Akzeptiert werden nur Metadaten, die im Dublin-Core Element Set definiert worden sind bzw. in der von uns durchgeführten Erweiterung des Sets enthalten sind. Folgende Metadateneinträge werden akzeptiert:

- `$DC.TITLE`
- `$DC.CREATOR`
- `$DC.SUBJECT`
- `$DC.DESCRIPTION`
- `$DC.PUBLISHER`
- `$DC.CONTRIBUTOR`

- $DC.DATE
- $DC.TYPE
- $DC.FORMAT
- $DC.IDENTIFIER
- $DC.SOURCE
- $DC.LANGUAGE
- $DC.RELATION
- $DC.COVERAGE
- $DC.RIGHTS
- $DC.X-LOCATION.X (X-Position in der som)
- $DC.X-LOCATION.Y (Y-Position in der som)
- $DC.X-SIZE.PAGES (Anzahl der Seiten)
- $DC.X-SIZE.BYTES (Größe in Bytes)
- $DC.X-NUMBER_TIMES_REF (Gibt an, wie oft auf dieses Dokument schon zugegriffen wurde)
- $DC.X-LAST_REF (Datum des letzten Zugriffs auf das Dokument)

Diesen Metadaten-Tags folgt dann eine Zeichenkette, die den entsprechenden Metadatenattributen der Klasse `Item` zugewiesen werden. Um die Anzahl der im Dublin-Core Library File gespeicherten Dokumente zählen zu können, wurde das Metadaten-Attribut `$DC.TITLE` als verpflichtend definiert. Alle anderen Attribute sind optional anzugeben. Innerhalb der Datei werden die Einträge der Bibliothek sequentiell nacheinander aufgelistet, wobei ein neues Dokument mit einem `$DC.TITLE -` Tag beginnen muß. In den darauffolgenden Zeilen können dann die optionalen Attribute angegeben werden. Alle Zeilen, die nicht mit einem `$` beginnen, werden ignoriert. Endresultat ist eine befüllte Dokumentliste, die der aufrufenden Klasse übergeben wird. Sollten beim Einlesen schwerwiegende Fehler aufgetreten sein (z.B. wenn die Datei nicht gefunden wurde), so erkennt dies die aufrufende Funktion daran, daß das Flag `Lib_loaded` auf false gesetzt wurde.

5.2.6 class `Item`

Diese Klasse stellt die Datenstruktur zur Verfügung, um alle im Bibliotheks-File enthaltenen Metadaten verwalten zu können. Die Variablen entsprechen in Anzahl und Namensgebung genau den im Abschnitt 5.2.5 beschriebenen Dublin-Core Metadaten samt den von uns vorgenommenen Erweiterungen.

5.2.7 class `ItemList`

Die Klasse `ItemList` wird dazu verwendet, die Dokumente der Dublin-Core Bibliotheksdatei zur internen Weiterverarbeitung in einer Liste abzulegen. Die Elemente der Liste sind vom Typ `Item` (siehe Abschnitt 5.2.6).

5.2.8 class `Language`

In dieser Klasse befindet sich die Einleseroutine für alle Mappings von Sprache auf die Farben eines Dokuments. Dabei wird eine Datei eingelesen und die jeweiligen Mappings in einer Liste abgelegt. Diese Liste wird nachfolgend in der DC-Server Klasse `Converter` verwendet, um die Metainformation der sprachlichen Herkunft einer Ressource visuell auf das farbliche Erscheinungsbild abzubilden. Zu jeder Sprache kann ein unterschiedliches Farbmodell definiert werden, indem für die Farbe des Buchrückens `(COLOR_SPINE)`, des Seiten- und Oberteils `(COLOR_TOP und COLOR_FRONT)`, für den Text des Titels `(COLOR_TITLE)` und für den Text des Creators am Buchrücken `(COLOR_CREATOR)` ein unterschiedlicher Farbwert eingestellt wird.

5.2.9 class `Publisher`

Ähnlich wie bei der Klasse Language (siehe Abschnitt 5.2.8) wird auch bei dieser Klasse eine Datei mit vordefinierten Mappings eingelesen und in einer Liste gespeichert. Bei dieser Mapping-Art kann einem Verlag (Publisher) ein Logo zugeordnet werden, welches der libViewer anschließend am Buchrücken darstellt. Zusätzlich zum Logo ist es auch möglich, ein unterschiedliches Farbmodell zu definieren. Sollte also im Metadatenattribut `DC.PUBLISHER` des Dublin-Core Element Sets ein Verlag gefunden werden, der in dieser Liste vorhanden ist, so wird dem libViewer eine Kurzbezeichnung dieses Logos übermittelt (siehe auch Abschnitt 5.4, Attribut `IMAGE_LOGO`). Über diese Kurzbezeichnung wird dann im libViewer mittels der Klasse `Logos` (siehe Abschnitt 5.1.11) ein Filename mit Pfadangabe extrahiert, der auf die entgültige Imagedatei verweist.

5.2.10 class `PubTypes`

Durch die Klasse `PubTypes` wird eine Einleseroutine für ein Mapping zur Vefügung gestellt, welches eine Kombination zwischen dem Mapping von einem Verlag und einem Dokumenttyp auf unterschiedliche Farbkombinationen darstellt. Dabei wird wie bei allen derartigen Mapping-Verfahren eine Liste angelegt, die pro Verlag (Publisher) und Dokumenttyp eine Belegung für die im libViewer vorhandenen Farbgebungsmöglichkeiten definiert. Eine Anwendung eines derartigen Mappings ist beispielsweise bei Langenscheidt-Wörterbüchern gegeben, wo für den Verlag „Langenscheidt“ und den Dokumenttyp „dictionary“ die Oberflächenfarbe des Buches als gelb definiert wird.

5.2.11 class `Settings`

In dieser Klasse werden alle für die Programmlogik notwendigen Konstanten zentral verwaltet.

5.2.12 class `Types`

Um Dokumenttypen in der Realität den libViewer-Dokumenttypen zuordnen zu können, stellt die Klasse `Types` eine Einleseroutine zur Verfügung, die derartige Überleitungen einlesen kann und in einer Liste ablegt. Genau genommen kann für alle Dokumenttypen, die im Metadatenattribut DC.TYPE des Dublin-Core Element Sets vorkommen, eine Überleitung auf die libViewer-internen Dokumenttypen (hardcover, paperback, folder, newspaper, media) definiert werden. Zusätzlich kann auch für jeden Dokumenttyp eine spezielle Farbgebung definiert werden.

5.3 Klassenbeschreibung AltaVista Server

Mit dem AltaVista-Server wird es für den libViewer möglich, Queries (Suchabfragen) an die Suchmaschine AltaVista zu senden und das Ergebnis der Abfrage im libViewer-Format zu empfangen. Der schematische Ablauf der Kommunikation zwischen den einzelnen libViewer-Clients und dem AltaVista-Server ist in Abbildung 5.4 dargestellt.

Die Abarbeitung der einzelnen Requests an den AV-Server erfolgt parallel, da für jede Anfrage ein eigener Thread kreiert wird. Die Kommunikation wird mittels Socket bewerkstelligt und hat für beide Richtungen entsprechende Input- und Outputkanäle.

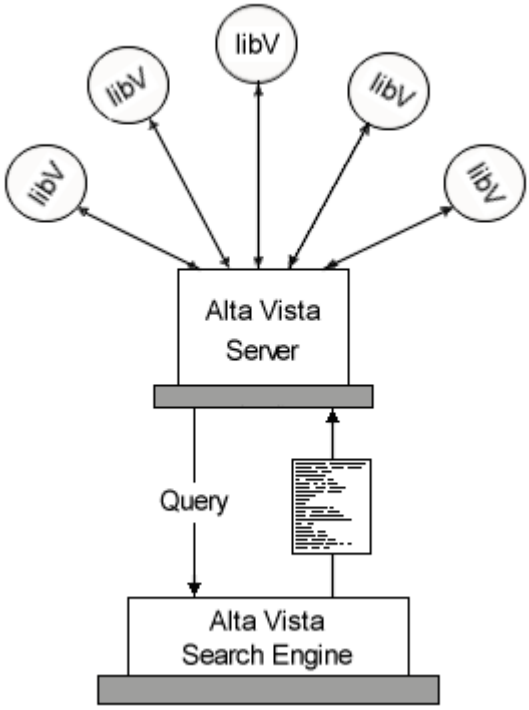

Abbildung 5.4: Schematische Darstellung der libViewer-AVServer Kommunikation

Schickt also ein libViewer-Client eine Suchabfrage an den AVServer, so wird aus der Query, die als Text übermittelt wird, eine Suchabfrage im AltaVista-Format generiert. Diese hat die Form einer URL und kann direkt zum download des Suchergebnisses verwendet werden. Nach Erhalt der Ergebnisse von der AltaVista-Suchmaschine konvertiert der AVServer die Daten in das libViewer-interne Dokumentformat und sendet sequentiell alle Dokumente an den aufrufenden libViewer zurück.

Durch die Implementierung von verschiedenen Mappings (Domain, Language, Type auf Farbgebung bzw. Logo für Domain am Buchrücken) kann das System flexibel auf den jeweiligen Anwendungszweck abstimmt werden.

In den folgenden Abschnitten wird jede für den AV-Server notwendige Klasse detailliert beschrieben. Es soll so einen funktionellen Überblick über die Architektur des AltaVista-Servers liefern. Dabei wurden bestimmte Klassen nur grob beschrieben, da sie schon in Abschnitt 5.2 (Dublin-Core Server) detailliert erörtert wurden. Die beschriebenen Klassen sind in alphabetischer Reihenfolge angeführt.

5.3.1 class `AVServer`

Durch Aufruf der Funktion `main` dieser Klasse wird der AltaVista-Server gestartet. Zuerst werden alle Mappings geladen und ein Server Socket auf einem vordefinierten Port geöffnet. In einer Endlosschleife wartet die Funktion dann auf die Kontaktaufnahme eines libViewers. Tritt dieses Ereignis ein, versucht der AVServer, die Socketverbindung zu etablieren, und erzeugt im Erfolgsfall einen neuen Thread, der durch die Klasse `AVServerMulti` implementiert wird und den Client-Socket als Eingabeparameter erwartet. Diese Technik hat den großen Vorteil, daß der `AV-Server` parallel mehrere Anfragen bearbeiten kann.

5.3.2 class `AVServerMulti`

Diese Klasse wird von der Java-Klasse `Thread` abgeleitet und arbeitet den kompletten Request eines libViewer-Clients ab. Dafür werden zuerst die Kommunikationskanäle geöffnet, das sind ein `BufferedReader`, um den eingehenden Request empfangen zu können, und ein `PrintWriter`, um die Dokumentdaten an den libViewer zurücksenden zu können. Sind die Kommunikationskanäle aufgebaut, liest der AV-Server die erste Zeile des Requests und verzweigt im Fall eines gültigen Funktionscodes (derzeit Laden der Ergebnisse 1-100 bzw. 101-200) in die Unterfunktion `receiveData`.

Die zweite Zeile, die vom libViewer gesendet wird, enthält den eigentlichen Suchstring. Dieser wird dann in das AltaVista-Format übersetzt, d.h. aus

den Keywords wird eine URL zusammengebaut, mit der von AltaVista die gewünschten Suchergebnisse heruntergeladen werden können. Der AltaVista-Server gibt entweder die ersten 100 oder die zweiten 100 Results an den libViewer zurück, je nachdem mit welchem Funktionscode er aufgerufen wurde. Da die Suchmaschine AltaVista aber nur 10 Hits pro Aufruf zurückliefert, wird der Reihe nach immer ein Zehnerblock von Hits heruntergeladen, bis insgesamt 100 Ergebnisse vorliegen oder die Maximalmenge an verfügbaren Hits erreicht ist. Um die Metainformation aus den heruntergeladenen Ergebnisseiten extrahieren zu können, werden die html-Seiten geparst. Folgende Matadaten können derzeit verwertet werden:

- URL des Dokuments
- Titel des Dokuments
- Beschreibung
- Datum der letzten Änderung (last modify date)
- Größe in Bytes
- Sprache, in der das Dokument verfasst wurde
- Domain der Ressource (wird aus der URL extrahiert)

Nach dem Parsing-Vorgang werden die einzelnen Dokumente konvertiert (Aufruf Funktion `processItem` / Klasse `Converter`), d.h. in das libViewer-interne Dokumentformat übersetzt, und an den aufrufenden libViewer zurückgesendet. Vorausgeschickt wird `$ITEMCOUNT`, wobei das Argument die Anzahl der Dokumente angibt, die der libViewer empfangen wird. Wenn alle Elemente gesendet wurden, terminiert der Thread.

5.3.3 class `Converter`

Diese Klasse beinhaltet das Konvertiermodul, welches die von der Suchmaschine AltaVista erhaltenen Dokumente (Hits) in ein für den `libViever` interpretierbares und darstellbares Format übersetzt. Dabei werden die einzelnen Metadatenattribute der Dokumente auf optische Merkmale der anzuzeigenden Elemente übergeleitet. Folgende Funktion bewerkstelligt diese Aufgabe:

- function `processItem`: Diese Methode erwartet als Eingabeparameter ein Objekt des Typs `Item`, das die von der AltaVista-Suchmaschine

erhaltenen Metadaten zu den einzelnen Dokumenten speichern kann. Sequentiell wird in dieser Funktion jedes der Attribute abgearbeitet, gegebenenfalls konvertiert und den Variablen des Objekts `ElementData` zugeordnet. Als Ausgabewert liefert die Funktion dann ein Objekt des Typs `ElementData`, dessen Daten anschließend dem libViewer zur Visualisierung gesendet werden. Der Ablauf kann schematisch folgendermaßen beschrieben werden:

1. Zuweisen aller Attribute, die von der Suchmaschine AltaVista zur Verfügung gestellt werden. Diese Daten werden vom libViewer 1:1 im Detail Window angezeigt (siehe 5.1.4).
2. Mapping Domain (extrahiert aus der URL des Dokuments) auf Logo ausführen. Dabei wird die Subfunktion `getDomainLogo` aufgerufen, die alle im dafür vorgesehenen domain-Preferences File erfaßten Mappings sequentiell nach einer Übereinstimmung hinsichtlich des Metadaten-Attributs `domain` durchsucht und im Erfolgsfall die Variable `Logo` der Klasse `ElementData` mit dem korrespondierenden Logonamen belegt. Eine genaue Beschreibung dieser Mapping-Variante findet sich in Abschnitt 5.3.4.
3. Mapping Type (Dublin Core) auf libViewer-Datentyp. Derzeit wird der Dokumenttyp fix als `html-doc` angenommen, kann aber bei entsprechender Erweiterung der AltaVista-Suchmaschine in Bezug auf die Angabe eines Ressource-Types jederzeit auf mehrere Dokumenttypen umgestellt werden.
4. Buchdicke ermitteln. Dabei wird die von AltaVista gelieferte Größenangabe in bytes mit einem vordefinierten Faktor multipliziert und so die Breite des Buchrückens in Pixel berechnet.
5. Buchhöhe bestimmen. Derzeit wird die Buchhöhe fix mit einem vordefinerten Wert belegt.
6. Ermitteln, ob das Dokument einen Hochglanzeffekt bekommen soll. Liegt das Datum, an dem das Dokument erstellt wurde, nicht mehr als ein Jahr zurück, dann wird das Flag `highlighting` gesetzt.
7. Farben (Color top, Color front, Color spine, Color Title, Color Creator) setzen. Es erfolgt der Aufruf der Unterfunktion `setColors`, wo schrittweise die Farben für das übergebende Dokument ermittelt werden. Eingangs setzt die Funktion die Farben der Flächen (Buchrücken, Seiten- und Oberteil) auf einen Defaultwert. Danach werden die in den Mappings (falls vorhanden) defi-

nierten Farbkombinationen je nach Priorisierung auf das Element übertragen. Folgende Mappings werden berücksichtigt:

- Typ auf Dokumentfarben (derzeit nur mittels `html-doc` möglich)
- Domain auf Dokumentfarben
- Sprache auf Dokumentfarben

Je nach Priorisierung kommt ein Mapping früher oder später zur Anwendung, d.h. das zuletzt angewendete Mapping überschreibt eventuell schon vorher dem Element zugewiesene Farben. Sollte nach Ablauf dieses Verfahrens die Farbe für die Textbausteine am Buchrücken noch unbelegt sein, so wird je nach Helligkeit des Dokumentrückens dessen Farbe automatisch bestimmt.

5.3.4 class `Domain`

Durch die Klasse `Domain` wird eine Einleseroutine für ein Mapping zur Vefügung gestellt, welches bestimmten Domains ein Logo am Buchrücken zuordnet. Dabei wird wie bei allen derartigen Mapping-Verfahren eine Liste angelegt, die pro Domain ein dazugehöriges Logo definiert. Zusätzlich dazu kann pro Domain auch ein eigenes Farbmodell eingestellt werden. Domains werden grundsätzlich als Teil einer URL angenommen, d.h. wird der im Feld Domain als Substring definierte Text in der URL des Dokuments gefunden, wird eine Zuordnung zu dem entsprechenden Logo vorgenommen. Beispielsweise wird der Domain-Substring `.at/` in der URL `http://ifs.tuwien.ac.at/` gefunden und liefert daher eine Zuordnung zu dem entsprechenden Logo, welches dann am Buchrücken dargestellt wird.

5.3.5 class `ElementData`

Der Aufbau dieser Klasse entspricht exakt dem der gleichnamigen libViewer-Klasse `ElementData` und wurde schon im Abschnitt 5.1.7 beschrieben.

5.3.6 class `Item`

Diese Klasse beinhaltet den Datentyp zur Verwaltung der von der Suchmaschine AltaVista zurückgelieferten Metadaten.

5.3.7 class `ItemList`

Die Klasse `ItemList` stellt den Datentyp dar, mit der die von AltaVista zurückgelieferten Dokumentlinks (Hits) in einer Liste verwaltet werden.

5.3.8 class `Language`

Diese Klasse beinhaltet die Laderoutine für das Mapping von der Metainformation Sprache auf die Farben des libViewer-Dokuments. Der Aufbau entspricht exakt dem in Abschnitt 5.2.8 beschriebenen Modul des DCServers.

5.3.9 class `Settings`

In dieser Klasse werden alle für die Programmlogik notwendigen Konstanten zentral verwaltet.

5.3.10 class `Types`

Diese Klasse beinhaltet die Laderoutine für das Mapping von Typen in der Realität auf libViewer-interne Dokumenttypen. Da die AltaVista-Suchmaschine derzeit noch keine Klassifizierung in Dokumenttypen zurückliefert, ist dieses Mapping ohne Wirkung, kann jedoch bei entsprechender Erweiterung der AltaVista-Suchmaschine jederzeit miteingebunden werden. Der Dokumenttyp wird momentan von der Klasse AVServerMulti fix als `html-doc` festgesetzt und dem libViewer zurückgesendet.

5.4 Protokoll

In diesem Abschnitt beschreiben wir das libViewer-libServer Protokoll, welches (a) die Anfrage (Request) an einen libServer und (b) das Format der zurückgelieferten Elemente der Dokumentliste definiert.

5.4.1 Syntax

Datenzeilen beginnen mit dem Zeichen `$`, gefolgt von einem Schlüsselwort (Keyword). Durch ein Leerzeichen getrennt wird das Argument angehängt, das (a) vom Typ String, (b) vom Typ Boolean, (c) vom Typ Integer oder (d) vom Typ Float sein kann. Strings sind Zeichenketten, die auch Leerzeichen enthalten können. Boolsche Argumente können entweder `0` oder `1` enthalten. Integer-Werte sind ganzzahlig und haben den für Integer-Variablen

typischen Wertebereich. Bei Float-Argumenten (Kommawerten) muß der Dezimalpunkt als . ausgeführt sein.

Kommentarzeilen beginnen mit dem Zeichen # und werden beim Einlesevorgang ignoriert.

5.4.2 Request an einen libServer

Diese Anfrage wird von einem libViewer zu einem korrespondierenden libServer gesendet.

- `$SERV_BUT` String, mandatory. Der Funktionscode des Buttons, der aktiviert wurde, z.B. `$SERV_BUT1` oder `$SERV_BUT2`.
- `$SERV_DATA` String, mandatory. Die Daten für die auszuführende Aktion.
- `$SERV_RESERVED` String, mandatory. Reserviert, wird derzeit nicht verwendet.

5.4.3 Response (Antwort) vom `libServer` an den `libViewer`

Dieser Abschnitt beschreibt überblicksmäßig die Struktur der Daten, welche vom libServer an den libViewer nach der Abarbeitung einer Anfrage zurückgesendet werden. [20]

Zuerst kann der libServer eine Aktion an der aufrufenden libViewer zurücksenden. Tabelle 5.1 beschreibt das Datenformat dieser Aktion, welche vom libViewer im dafuer vorgesehenen Action-Dialog hinzugefügt wird und entspricht dem in der Klasse Action (siehe 5.1.1) beschriebenen Aufbau. Der ganze Block ist optional.

Danach wird die Anzahl der Dokumente übermittelt, die folgen werden. Dabei wird das Keyword `$ITEMCOUNT` mit der Dokumentanzahl als Argument gesendet.

Für jedes Element wird dann die in Tabelle 5.2 abgebildete Struktur wiederholt:

5.5 Zusammenfassung

In diesem Kapitel wurden alle Klassen des libViewer-Systems beschrieben, um dieses System auch in Zukunft für Neueinsteiger wartbar und erweiterbar zu machen. Diese Klassenreferenz beinhaltet die Klassen des libViewer-Applets,

Keyword	*Datentyp*	*Beschreibung*
$SERV_URL	String	URL des libServers
$SERV_PORT	String	Port für diese Verbindung
$SERV_DESCR	String	Kurzbeschreibung für diese Verbindung
$SERV_DATA	String	Default-Query für diese Aktion
$SERV_BUT1	String	Funktionscode für Button 1
$SERV_BUT2	String	Funktionscode für Button 2
$SERV_BUT1TXT	String	Label für Button 1
$SERV_BUT2TXT	String	Label für Button 2
$SERV_RESERVED	String	reserviert

Tabelle 5.1: Aktion

sowie die der beiden Server (DCServer und AVServer). Im letzten Abschnitt dieses Kapitels findet sich auch ein grober Überblick über das libViewer-libServer Protokoll, welches die Kommunikation zwischen Client und Server regelt.

Keyword	*Datentyp*	*Beschreibung*
$TITLE	String	Titel des Dokuments (DC)
$WIDTH	Integer	Breite des Buchrückens in Pixel
$HEIGHT	Integer	Höhe des Buchrückens in Pixel
$CREATOR	String	Autor des Dokuments (DC)
$SUBJECT	String	„Subject“ des Dokuments (DC)
$DESCRIPTION	String	Beschreibung des Dokuments (DC)
$PUBLISHER	String	Herausgeber des Dokuments (DC)
$CONTRIBUTOR	String	„Contributor“ des Dokuments (DC)
$DATE	String	Datum der Herstellung (DC)
$TYPE	String	Typ des Dokuments (DC)
$FORMAT	String	Format des Dokuments (DC)
$IDENTIFIER	String	„Identifier“ des Dokuments (DC)
$SOURCE	String	„Source“ des Dokuments (DC)
$LANGUAGE	String	Sprache des Dokuments (DC)
$RELATION	String	„Relation“ des Dokuments (DC)
$COVERAGE	String	„Coverage“ des Dokuments (DC)
$RIGHTS	String	„Rights Statement“ des Dokuments (DC)
$DOMAIN	String	Domain des Dokuments
$SIZE_PAGE	String	Größe des Dokuments in Seiten
$SIZE_BYTE	String	Größe des Dokuments in Bytes
$IMAGE_LOGO	String	Kurzname für Logo
$TEMPLATE	String	libViewer-Datentyp des Dokuments
$WELLTHUMBED	Boolean	Effekt „abgegriffen“
$DUST_LEVEL	Integer	Staub-Level
$COLOR_FRONT	Integer (rgb)	Farbe der Vorderseite
$COLOR_TOP	Integer (rgb)	Farbe der Oberseite
$COLOR_SPINE	Integer (rgb)	Farbe des Buchrückens
$COLOR_TITLE	Integer (rgb)	Farbe des Titeltextes
$COLOR_CREATOR	Integer (rgb)	Farbe des Autortextes
$HIGHLIGHTING	Boolean	Highlighting-Effekt
$XPOS	String	X-Position des Dokuments
$YPOS	String	Y-Position des Dokuments
$ZPOS	String	Z-Position des Dokuments

Tabelle 5.2: Struktur eines Dokuments

Kapitel 6

User Interface

Zwecks dieses Kapitels ist es, das User Interface des libViewers detailliert zu erklären und anhand von Beispielsessions die Interaktion mit den beiden Servern, dem DCServer bzw. dem AVServer, zu beschreiben. Dabei werden die verwendeten Metaphern in der Praxis vorgestellt und Funktionsabläufe in der Benutzerführung verdeutlicht.

6.1 Beschreibung

Dieser Abschnitt beschreibt die Benutzeroberfläche (User Interface) des libViewers. Anhand der Abbildung 6.1 werden alle grafischen Komponenten bzw. Bedienelemente erklärt, die zur Interaktion mit dem libViewer notwendig sind.

Die Oberfläche des libViewer-Applets besteht aus folgenden Elementen:

- (1) Display Window. In diesem Bereich wird die Dokumentliste grafisch ausgegeben.
- (2) Server-Auswahlbox. Mit diesem Bedienelement wird der Server ausgewählt, an den die Abfrage gesendet werden soll. Derzeit kann entweder der AVServer oder der DCServer eingestellt werden.
- (3) Eingabezeile. Dieses Textfeld dient zur Eingabe der Abfrage, die an den korrespondierenden libServer gesendet wird. Der Inhalt dieses Feldes ist im Fall eines DC-Server-Requests der Dateiname einer Bibliothek im Dublin-Core Format bzw. beim AVServer eine Suchabfrage (Query).
- (4) Funktionsbutton 1. Durch Betätigung dieses Buttons wird der Request mit Funktionscode 1 an den libServer gesendet. Funktionscodes werden im Server-Preferences File (siehe Kapitel 5), Abschnitt 5.1.18 definiert und beschreiben die Art der Funktion, die der kontaktierte

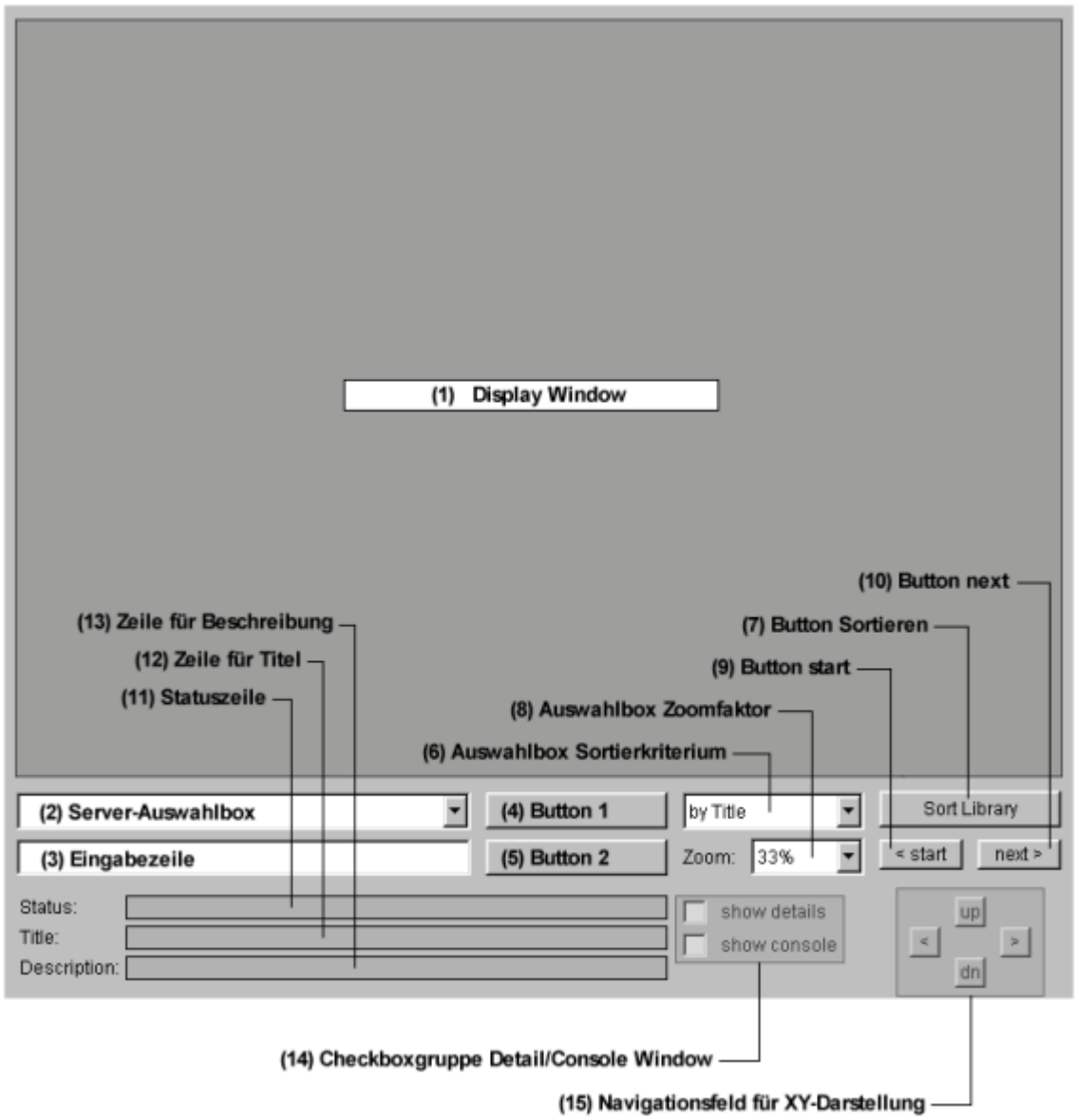

Abbildung 6.1: Oberfläche des libViewers

Server ausführen soll. Dies ist im Fall des DCServers das Laden einer Bibliothek bzw. beim AVServer eine Suchabfrage an die Suchmaschine AltaVista.

- (5) Funktionsbutton 2. Analog zu (4) wird ein Request abgeschickt, jedoch mit Funktionscode 2. Dieser Button wird gewöhnlicherweise mit einer alternativen Funktion belegt, z.B. bei Aufruf des AVServers eine Abfrage, die die Hits 101 bis 200 zurückliefert.

- (6) Auswahlbox Sortierkriterium. Hier wird das Kriterium eingestellt, nach welchem Metadaten-Attribut die aktuell geladene Dokumentliste sortiert werden soll. Derzeit mögliche Sortierkriterien sind Title, Creator, Publisher, Format, Type, Language, Domain, Byte-Size, Page-Size und Date.

- (7) Button Sortieren. Durch Betätigen dieses Buttons wird die aktuelle Dokumentliste nach dem in (6) eingestellten Kriterium sortiert.

- (8) Auswahlbox Zoomfaktor. Dieses Bedienelement dient zur Auswahl des gewünschten Zoomfaktors. Derzeit werden die Einstellungen 33% (Übersichtsdarstellung) und 100% (Detaildarstellung) unterstützt.

- (9) Button start. Dient dazu, an den Anfang der Dokumentliste zu springen. Im Fall der XY-Darstellung (Einteilung in Regalkästchen bei geclusterten Dokumentlisten) wird innerhalb des aktuellen Regalkästchens navigiert. Diese Funktion ist nur in der Detaildarstellung (100%) aktiv.

- (10) Button next. Mit diesem Button wird innerhalb der Dokumentliste um den derzeit dargestellten Bereich weitergesprungen, d.h. das Dokument, das vorher am Ende der angezeigten Dokumentliste gestanden hat, bildet jetzt den Anfang der Liste.

- (11) Statuszeile. In dieser Textzeile werden Statusmeldungen ausgegeben, z.B. der Ladefortschritt beim Laden einer Bibliothek.

- (12) Zeile für Titel. Immer wenn der Anwender die Maus über einen Buchrücken bewegt, wird diese Textzeile mit dem Titel des jeweiligen Dokuments aktualisiert.

- (13) Zeile für Beschreibung. Analog zu (12) wird hier die Beschreibung des Dokuments ausgegeben.

- (14) Checkboxgruppe Detail/Console Window. Mit Aktivierung der jeweiligen Checkbox wird ein eigenständiges Fenster geöffnet - das Detail-Window zur Anzeige der Metadaten eines Dokuments bzw. die Console, die zur Fehlersuche implementiert wurde.

- (15) Navigationsfeld für XY-Darstellung. Diese Funktionsgruppe wird nur dann angezeigt, wenn eine geclusterte Dokumentliste geladen wurde. Dies ist dann der Fall, wenn alle empfangenen Dokumente gültige x- und y-Koordinaten aufweisen. Mit den 4 Buttons (up, down, left, right) kann der Anzeigebereich innerhalb des virtuellen Fensters bewegt werden. Dabei wird immer um einen Knoten bzw. ein Regalkästchen in die entsprechende Richtung weitergesprungen. Nähere Infos zu dieser Funktionalität finden sie auch in Abschnitt 5.1.12.

In den folgenden Abschnitten wird obige Numerierung zur Beschreibung der Funktionalität des libViewers verwendet.

6.2 Beispielsessions

Folgende zwei Abschnitte beschreiben anhand von Beispielsessions die Funktionsweise des libViewers und der beiden Server. Zuerst wird die Interaktion mit dem DCServer abgehandelt und gleichzeitig die verwendeten Metaphern in der Praxis vorgestellt. Der zweite Teil stellt den AltaVista-Server zur Visualisierung von Suchergebnissen vor.

6.2.1 Interaktion mit dem Dublin-Core Server

Um eine digitale Bibliothek vom DCServer laden zu können, muß zuerst in der Server-Auswahlbox (2) der Eintrag „Dublin Core Server“ ausgewählt werden. Es folgt die Eingabe eines Dateinamens in das dafür vorgesehene Eingabefeld (3). Um den Request an den DCServer abzusenden, betätigt man den Funktionsbutton1 (4), welcher durch die Auswahl von „Dublin Core Server“ mit „Load Library“ beschriftet ist. Nachdem der Dublin-Core Server intern die Bibliothek geladen hat, sendet er die Dokumente sequentiell an das libViewer-Applet zurück. Der Ladefortschritt in Prozent kann dabei in der Statuszeile (11) mitverfolgt werden. Ist die Dokumentliste vollständig geladen, baut der libViewer das Ausgabefenster (1) neu auf und zeichnet die Bibliothek.

Abbildung 6.2 zeigt die Visualisierung einer Beispielbibliothek in Übersichtsdarstellung, die standardmäßig nach dem Ladevorgang eingestellt wird. In dieser Darstellung sind schon einige wichtige Attribute erkennbar, wie der

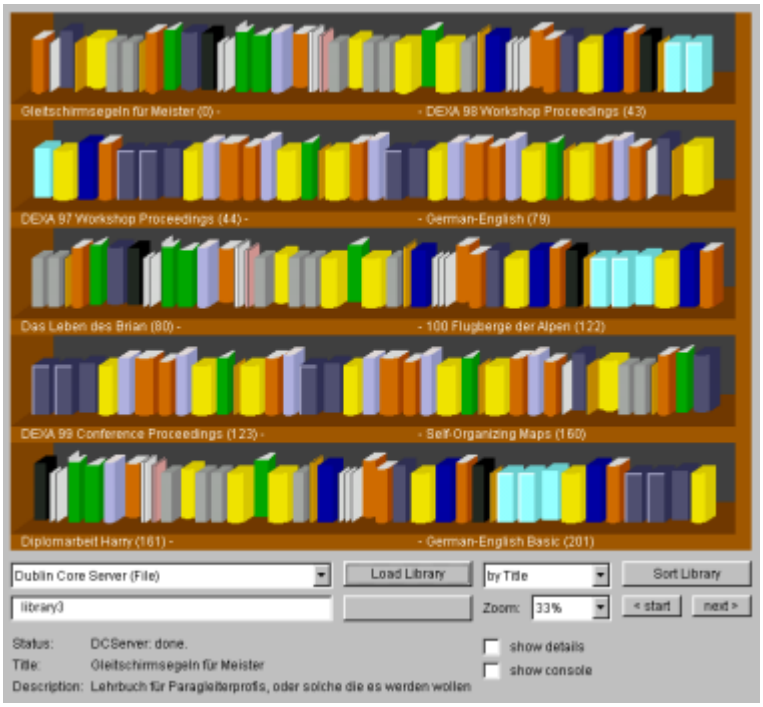

Abbildung 6.2: Übersichtsdarstellung einer Dublin-Core Library

Dokumenttyp, die Dicke des Buchrückens oder die Farbgebung. Die Metaphern Staub, Text am Buchrücken etc. sind - wie auch in konventionellen Bibliotheken - erst in der Detaildarstellung sichtbar.

Um jetzt von der Übersichtsdarstellung in die Detaildarstellung zu wechseln, d.h. einen speziellen Bereich zu vergrößern, kann entweder mittels der Auswahlbox Zoomfaktor (8) oder durch Klick mit der linken Maustaste auf einen Buchrücken hineingezoomt werden. Abbildung 6.3 zeigt die Detailansicht eines Ausschnittes aus unserem Beispiel einer Dublin-Core Bibliothek. In dieser Darstellung kann mehr sehr schön die unterschiedlichen Dokumenttypen erkennen. Dazu gehören hardcover- und paperback-Bücher, technische Reports sowie Papers, die leicht anhand ihrer korrespondierenden physischen Repräsentation identifiziert werden können. Beispielsweise befinden sich in dieser Ansammlung technische Reports des libViewers bzw. des somViewers, welche als grüne Ordner visualisiert sind. Weiters stechen sofort die 4 unterschiedlichen Langenscheidt-Wörterbücher als gelb colorierte hardcover-Bücher in das Auge des Betrachters. Die physische Repräsentation, d.h. der verwendete libViewer-interne Dokumenttyp, wird durch ein Mapping des Original-Dokumenttyps auf die libViewer-internen Dokumenttypen ermittelt. In diesem Beispiel werden „journal papers“ als auch „conference

Abbildung 6.3: Detailansicht einer Dublin-Core Library

papers“ auf den Typ hardcover übergeleitet. Der Unterschied zischen den beiden Ressource-Types liegt in diesem Fall nur mehr in der anderen Farbgebung, indem die „journal papers“ dunkler als die weißen „conference-papers“ dargestellt werden. Ein weiteres Beispiel, wo mehrere Ressource-Types einem libViewer-internen Dokumenttyp - in diesem Fall dem Typ Ordner - zugeordnet werden, sind der technische Report und die Dokumentation. Der Unterschied zwischen Dokumentation und technischem Report läßt sich anhand der unterschiedlichen Höhe des Ordners erkennen. Auf diese Art und Weise können alle Dokumenttypen, die das Dublin-Core Element Set für Ressource-Types vorsieht, einer physischen Repräsentation zugeordnet werden. Theoretisch könnte man für jeden Dokumenttyp aus Dublin Core eine eigene physische Repräsentation schaffen. Die Evaluierung unseres Systems hat aber gezeigt, daß ein Mapping wie in unserem Beispiel in den meisten Fällen ausreicht und die intuitive Erfaßbarkeit zusätzlich steigert, da viele Unterarten in den Metadaten eher unwichtig bei der Suche nach einer bestimmten Information sind.

Mit derselben Technik wurden auch andere Metadaten-Attribute den verfügbaren Metaphern zugeordnet. So kann beispielsweise der Verlag anhand des Logos am Buchrücken identifiziert werden, sofern dieses für einen

bestimmten Verlag vorhanden ist (z.B. Springer, Langenscheidt, etc.). Sehr gut kann man auch die Größe eines Dokumentes abschätzen, welche auf die Dicke des Buchrückens abgebildet wird. Ein gutes Beispiel für unterschiedliche Größe bzw. Buchdicke sind die 4 Langenscheidt-Wörterbücher in Abbildung 6.3. Ein weiteres Mapping bildet den Zeitpunkt des letzten Zugriffs auf die Metapher Staub ab. Für diesen Zweck wurden insgesamt 5 Staubigkeitsstufen realisiert, von wenigen Staubpartikeln bis hin zu Spinnweben, die die Hälfte des Buchrückens verdecken. Wurde also auf eine Ressource schon sehr lange nicht mehr zugegriffen, erhält sie Spinnweben auf dem Buchrücken, wie es beim hardcover-Buch links unten in Abbildung 6.3 nachvollziehbar ist. Wurde ein Dokument hingegen schon sehr oft verwendet, erhält es ein „abgegriffenes“ Aussehen - ein Beispiel dafür ist der Ordner oder das blaue paperback-Buch, welche jeweils am rechten Rand der beiden Regalkästchen zu finden sind. Eine weitere Metapher zur Visualisierung der Relevanz eines Dokuments oder des Zeitpunkts des letzten Zugriffs auf eine Ressource stellt die unterschiedliche Anordnung der Bücher in z-Richtung dar. Einige der Dokumente in Abbildung 6.3 stehen hervor und vermitteln den Eindruck, daß man diese leichter aus dem Regal entnehmen kann. Hingegen sind unwichtige oder schon länger nicht mehr benutzte Bücher nach hinten versetzt, wie man beispielsweise beim dritten Buch im oberen Regalkästchen nachvollziehen kann.

Natürlich kann durch die grafische Repräsentation mittels Metaphern nicht die ganze Vielfalt an vorhandenen Metainformationen sinnvoll visualisiert werden. Aus diesem Grund wird beim Bewegen des Mauszeigers über einen Buchrücken das Detail-Window, das durch Aktivierung der dafür vorgesehenen Checkbox (14) sichtbar wird, mit den jeweiligen Metadaten des Dokuments befüllt. Abbildung 6.4 zeigt ein Beispiel, wie die vorhandenen Metadaten zu einem Dokument im Detail-Window angezeigt werden.

Eine weitere Funktionalität des libViewers ist das Sortieren einer Dokumentsammlung nach bestimmten Kriterien, die durch die vorhandenen Metadaten vorgegeben sind. Abbildung 6.5 zeigt die Sortierung unserer Beispielbibliothek nach dem Metadaten-Attribut Publisher (Verlag).

Zusätzliche Information liefert die Beschriftung des Regalbrettes, wo derzeit jeweils der Titel des ersten und des letzten im aktuellen Regalkästchen befindlichen Dokuments ausgegeben wird. Diese Beschriftung hängt letztlich vom Server ab und kann beispielsweise durch Ausgabe des Metadaten-Attributs, nach dem sortiert wurde, ersetzt werden.

Abbildung 6.6 zeigt die Visualisierung einer nach Themengebieten klassifizierten Bibliothek in der Übersichtsdarstellung. Die Dokumente sind in dieser Darstellung in verschiedene Regalkästchen, wobei jedes ein unterschiedliches Themengebiet repräsentiert, einsortiert. Die Beschreibung des Themenge-

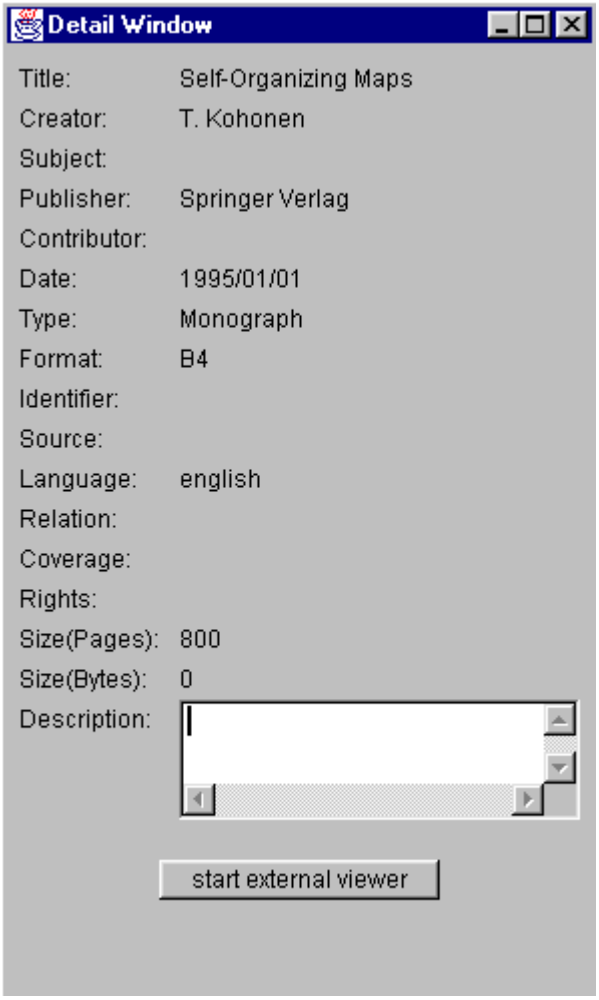

Abbildung 6.4: Detail-Window

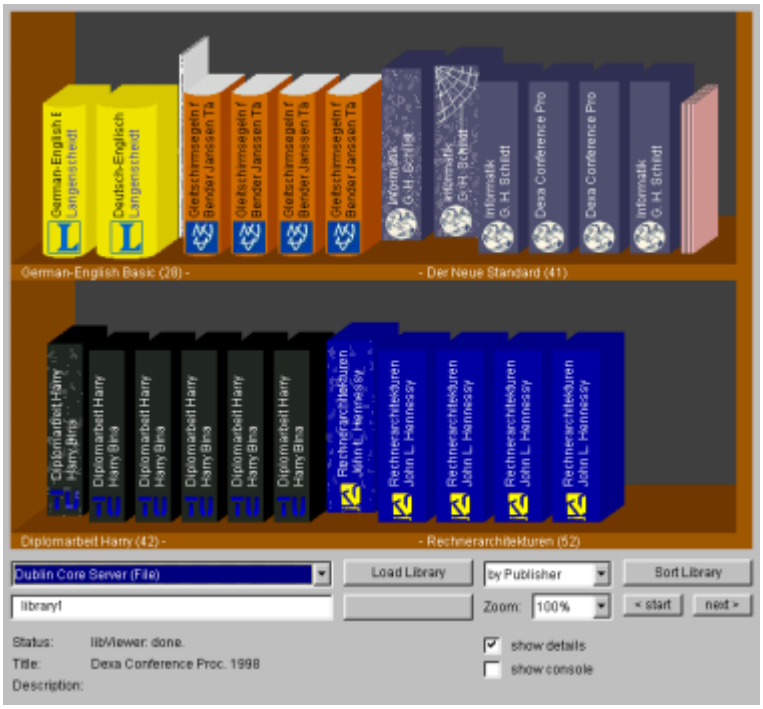

Abbildung 6.5: Dublin-Core Library sortiert nach Publisher

biets durch Schlüsselwörter (Keywords) befindet sich dabei auf den Regalbrettern und ermöglicht die Orientierung im Raster. Derzeit wird statt der Schlüsselwörter die jeweils aktuelle Position eines Segments im Raster ausgegeben, kann aber jederzeit durch die Ausgabe der Schlüsselwörter ersetzt werden. Zur Navigation innerhalb des Rasters befinden sich im rechten unteren Teil des Applets 4 Buttons (15), die ein Weiterspringen um jeweils ein Segment in jede Richtung innerhalb des Rasters ermöglichen. Abbildung 6.7 zeigt die Detailansicht der Darstellung einer nach Themengebieten klassifizierten Bibliothek.

6.2.2 Interaktion mit dem AltaVista Server

Durch den Alta-Vista Server wird es möglich, Suchabfragen an die Search-Engine AltaVista zu senden und das Ergebnis grafisch im libViewer anzuzeigen. Nachdem der AVServer die Query empfangen hat, kontaktiert er die Suchmaschine AltaVista und holt eine Liste von 100 Hits - je nach Aktivierung des Funktionsbutton1 (4) oder des Funktionsbutton2 (5) die Ergebnisse 1-100 oder 101-200. Um die Suchergebnisse in das libViewer-interne Dokumentformat übersetzen zu können, werden diese geparst und alle verwertba-

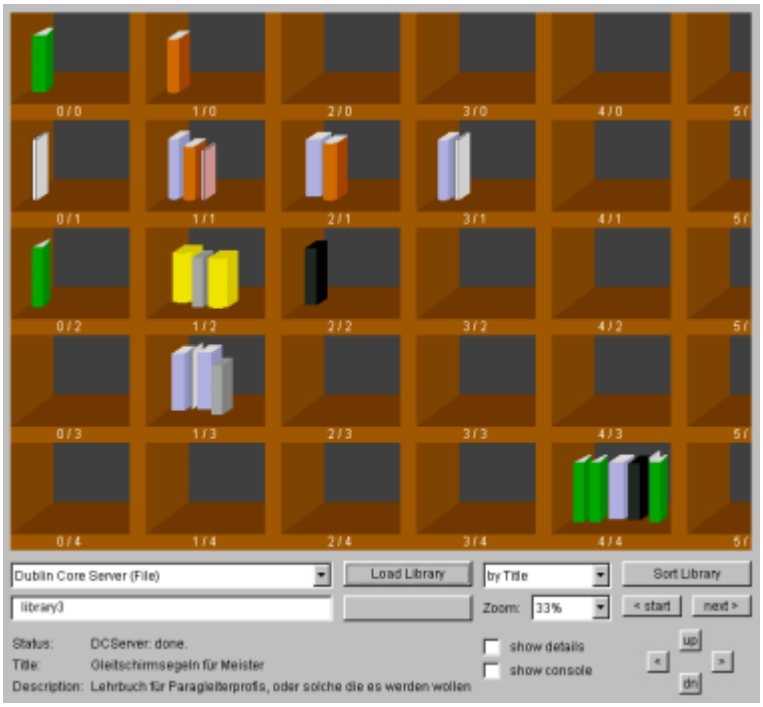

Abbildung 6.6: Dublin-Core Library nach Themengebieten klassifiziert

ren Metadaten den entsprechenden Attributen zugeordnet. Derzeit werden von AltaVista der Titel, eine Beschreibung, die URL, die Sprache und Größe (in Bytes) sowie das Datum der Erstellung des Dokuments zurückgeliefert. Aus der URL können wir zusätzlich die Domain extrahieren. Nach dem Parsing werden diverse Mappings ausgeführt, z.B. wird die Sprache auf die Dokumentfarben abgebildet oder für die Domain eines Hits eine Landesflagge als Logo am Buchrücken definiert. Nach Verarbeitung der Daten werden diese als Dokumentliste an den libViewer zurückgesendet.

Nach Eingabe der Suchabfrage in die Eingabezeile (3) wird der AVServer durch Betätigen des Button1 (4) oder Button2 (5) kontaktiert. Nach Abarbeitung der Anfrage bzw. Erhalt der Dokumentliste wird das Ausgabefenster neu aufgebaut. Abbildung 6.8 zeigt die libViewer-Visualisierung des Suchergebnisses der Query „deutsche Literatur“ in Übersichtsdarstellung. Aus dieser Darstellung kann der Betrachter sofort die sprachliche Herkunft der Dokumente ableiten. Neben Dokumenten in deutscher Sprache (schwarz-rot-gold coloriert) befinden sich auch englischsprachige Dokumente (blau-weiß-rot) im visualisierten Suchergebnis. Wird von AltaVista keine Metainformation zur Sprache eines Dokuments geliefert, erscheint dieses standardmäßig in grauer Farbe. Weiters kann man schon in der Übersicht erkennen, ob ein Dokument

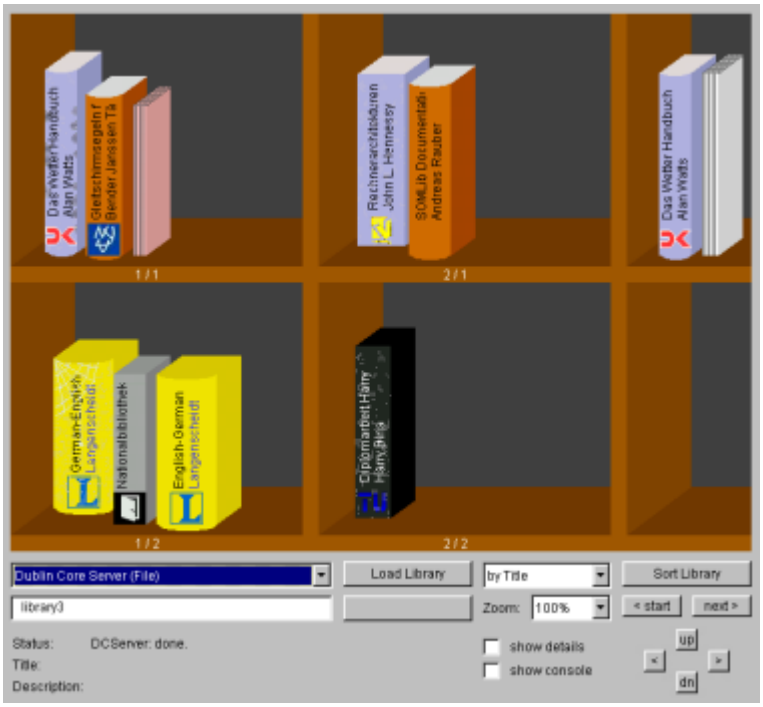

Abbildung 6.7: Detailansicht einer nach Themengebieten klassifizierten Bibliothek

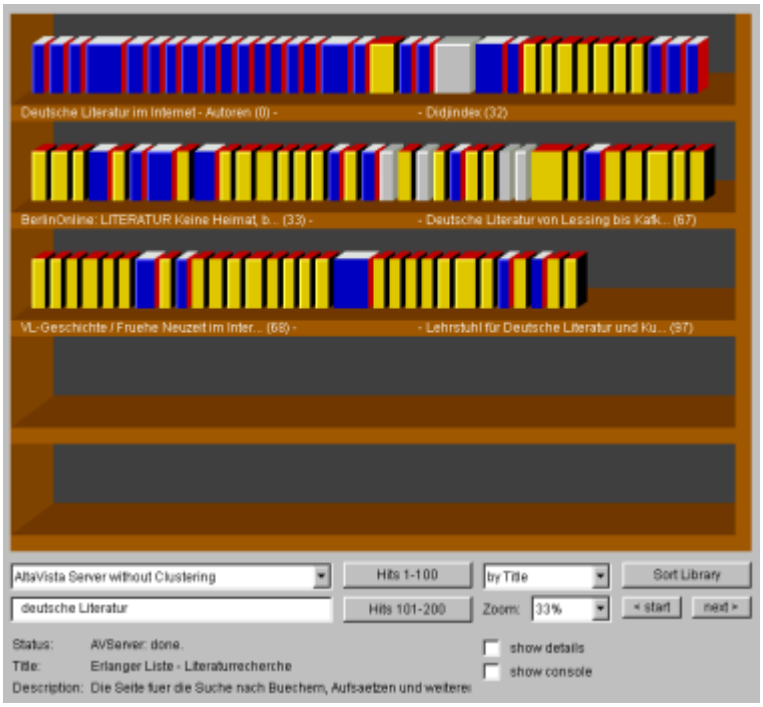

Abbildung 6.8: AltaVista Suchergebnis - Übersichtsdarstellung

neu (d.h. nicht älter als 1 Jahr) ist, da es in diesem Fall einen Hochglanzeffekt bekommt. Auch die Größe eines Dokuments kann anhand der Buchdicke beurteilt werden.

Selbstverständlich kann auch hier in die Detaildarstellung verzweigt werden - entweder durch Klick auf einen Buchrücken für das „Zooming“ eines speziellen Bereichs oder durch Verändern des Zoomfaktors (8). Abbildung 6.9 zeigt einen Teilbereich unserer Beispielquery.

Aus dieser Darstellung können wesentlich mehr Details herausgelesen werden als in der Übersicht. Erst jetzt wird der Titel des Dokuments (falls dieser von AltaVista zurückgeliefert wurde) am Buchrücken sichtbar. Sehr eindrucksvoll ist auch das Logo am Buchrücken, welches die Domain im Sinne von Landesflaggen und Domainbezeichnung darstellt. So ist aus Abbildung 6.9 ersichtlich, daß Hits aus den Domains at, de, uk, it und ch in unserem Suchergebnis enthalten sind. Für nicht länderspezifische Domains wie .com wird ein entsprechendes Logo mit dem Text der Domain verwendet, wie anhand des 1. Dokuments in Abbildung 6.9 nachvollzogen werden kann. Ist die Domain aber nicht übersetzbar, d.h. kein Logo für diese definiert worden, wird ein spezielles Logo für unbekannte Domains am Buchrücken angezeigt (siehe Dokument 2). Durch Klick mit der rechten Maustaste auf einen Buchrücken

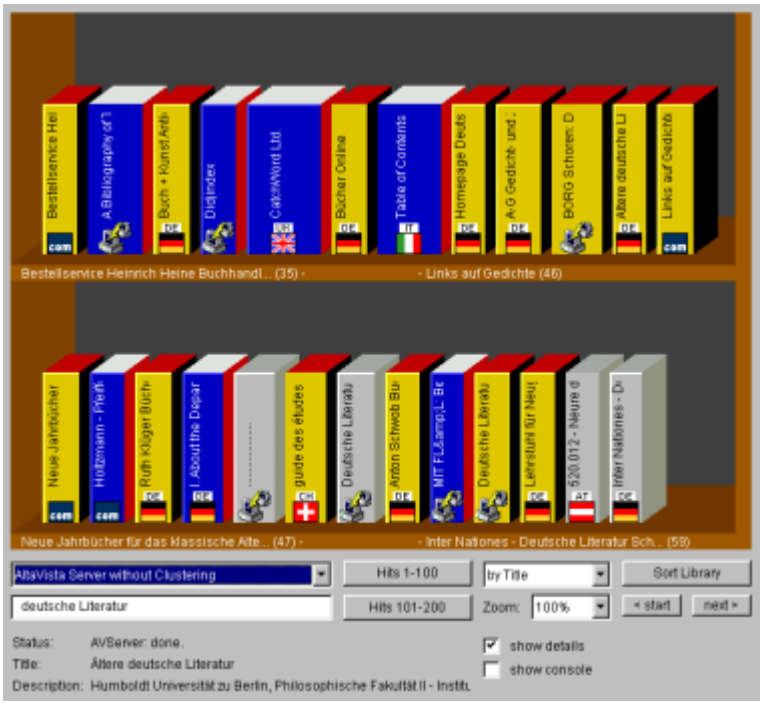

Abbildung 6.9: AltaVista Suchergebnis - Detailansicht

öffnet sich ein Web-Browser mit der jeweiligen URL des Dokuments.

6.3 Zusammenfassung

In diesem Kapitel wurde eingangs das User Interface des libViewer-Applets beschrieben. Erklärt wurde dabei die Funktionsweise jeder grafischen Komponente. Es folgten zwei Beispielsessions, in welchen die Interaktion des libViewers mit den beiden Servern, dem DCServer und dem AVServer anhand von Beispielgrafiken erklärt wurde. Dabei konnten die in Kapitel 4 vorgestellten Metaphern in der Praxis, d.h. bei der Visualisierung einer digitalen Bibliothek, betrachtet werden.

Kapitel 7

Usability Evaluation

Nachdem wir die ersten Prototypen des libViewers implementiert hatten stellten wir das System einer Gruppe von Anwendern vor. Diese Gruppe setzte sich aus Bibliothekaren und Informationswissenschaftern, aber auch aus Studierenden von Studienrichtungen abseits der Informatik (z.B. Juristik) zusammen. Wir wollten von diesen Personen ein Feedback über die Usability (Benutzbarkeit) unseres Systems gewinnen. Die meisten Anwender waren sofort fasziniert von der neuartigen grafischen Repräsentation und empfanden die verwendeten Metaphern als selbsterklärend und logisch. Nach und nach wurden jedoch einige Punkte angesprochen, die das System verbessern könnten. Dieses Kapitel gibt einen Überblick über die einzelnen Verbesserungsvorschläge und die erhaltene Kritik.

7.1 Mappings

Am Anfang unserer Entwicklung wollten wir ein komplexes Set an Mappings von Metadaten-Attributen auf Metaphern verwirklichen. Eine Idee war dabei, die Möglichkeiten derart zu gestalten, daß alle Metadaten-Attribute Einfluß auf jede Metapher nehmen können, d.h. ein völlig flexibles und vollständiges Mappingschema zur Verfügung steht. Nach reiflicher Überlegung sind wir aber zu dem Entschluß gekommen, daß sorgfältig ausgewählte Mappings in den meisten Fällen völlig ausreichen, ja sogar vorteilhaft sind.

Unser erstes Ziel war es, Bücher so realistisch wie möglich als ihre Gegenstücke aus der realen Welt nachzubilden. Dies hat beispielsweise zu einer Vielzahl an möglichen Mappings von Verlag (Publisher) und Dokumenttyp auf die Farbe eines Dokuments geführt. Zwei Beispiele, wo dieses Mapping sinnvoll angewendet erscheint ist die Farbgebung für Wörterbücher des Verlages Langenscheidt bzw. Pons, die entweder gelb oder grün dargestellt wer-

den. Diese Farbzuordnung erlaubt es dem Betrachter, derartige Bücher sofort in größeren Dokumentsammlungen ausfindig zu machen und zu identifizieren. Für allgemeinere Anwendungen erscheint dieses Verfahren jedoch als unnotwendig kompliziert, da der Betrachter nur eine bestimmte Anzahl an Farbzuordnungen unterscheiden kann.

7.2 Größe und Buchdicke

Wir haben herausgefunden, daß neben der physischen Repräsentation einer Informationsressource die Größe ein sehr hilfreiches Gestaltungsmerkmal darstellt. Man kann die Anzahl an verfügbarer Information viel leichter visuell durch z.B. die Buchdicke vermitteln als durch textuelle Repräsentation wie beispielsweise die Angabe in Bytes oder Seitenzahlen. Speziell beim AltaVista-Server haben wir aber festgestellt, daß die Unterschiede in der Buchdicke bei bestimmten Queries nicht mehr unterschieden werden können. Dies verlangt nach einem verfeinertem Berechnungsverfahren, wo statt bisher die Buchdicke nicht linear mit der Seitenanzahl bzw. der Größe in Bytes anwächst sondern logarithmisch ermittelt wird. Dies korrespondiert auch mit der Realität, wo Bücher mit großen Seitenanzahlen durch dünnere Seiten handlicher gestaltet werden und dadurch dem linearen Anwachsen der Buchdicke pro Seitenanzahl widersprechen. Zusätzlich dazu ist der Einband eines Buches dicker als die tatsächlichen Seiten und stellt einen fixen Anteil in der Dicke des Buchrückens dar, was ebenso einem linearen Anwachsen der Buchdicke entgegenwirkt.

7.3 Hochglanzeffekt

Der Hochglanzeffekt zur Identifizierung neuer Dokumente hat sich bei der Evaluierung als ein sehr wirkungsvolles Merkmal herausgestellt. Dennoch haben manche Anwender diese Metapher beim erstmaligen Betrachten nicht sofort bemerkt und mußten erst dezidiert darauf hingewiesen werden. Dies stellt zur Diskussion, ob wir dieses Merkmal dominanter als bisher gestalten sollen.

7.4 Sprache als Farbe codiert

In Bezug auf das Mapping des Metadaten-Attributs Sprache auf die Farbe des Dokuments haben wir konträre Rückmeldungen erhalten. Während manche die Codierung der Sprache in die Landesfarben (z.B. schwarz-rot-gelb für die

Begrenzungsflächen des Buches bei deutscher Sprache) als sehr intuitiv beurteilten, waren andere im ersten Moment etwas irriert. Nach einer gewissen Eingewöhnungszeit haben aber alle diese Metapher perfekt interpretiert, was den Einsatz dieses Gestaltungsmerkmales bestätigt und als hilfreich erachten läßt.

7.5 Landesflaggen für Domains

Beim AltaVista-Server haben wir ein spezielles Mapping von der Domain eines Dokuments im Internet auf das Logo am Dokumentrücken geschaffen. Der AVServer extrahiert zuerst aus der URL des Dokuments eine Domain, die wiederum einem Image zugeordnet wird. Beispielsweise erhalten auf diese Weise Dokumente der Domain „at“ die österreichische Landesflagge als Image am Buchrücken. Es hat sich gezeigt, daß diese Metapher den Anwendern viel dabei geholfen hat, die gewünschten Dokumente zu finden, insbesondere dann, wenn die Suchabfrage zu unpräzise formuliert wurde. Diese Metapher kann auch sehr gut dazu verwendet werden, domainspezifische Ressourcen auf einfache Art und Weise aufzuspüren.

Kritik wurde natürlich dann geübt, wenn für ein spezifisches Land keine Flagge vorhanden war. Dazu kann gesagt werden, daß wir versucht haben, für die wichtigsten Länder Logos zu erstellen. An einer Vervollständigung dieses Sets wird derzeit gearbeitet.

Manche Personen (vor allem Informationswissenschafter) haben bemerkt, daß die Landesflaggen im Gegensatz zu den nicht-nationalen Domainsymbolen wie „.com“, „.org“, „.net“ usw. zu dominant seien. Der Anwender könnte deswegen unter Umständen Ressourcen übersehen, weil sie nicht mit den dazugehörigen Landesflaggen ausgestattet sind. Dieses Problem tritt besonders bei der Domain „.com“ auf.

7.6 zusätzliche Funktionalität

Bei der Interaktion der Personen mit dem libViewer wurden Erweiterungsmöglichkeiten genannt, die sinnvoll in das System integriert werden könnten. Das Sortieren der Bibliothek nach unterschiedlichen Metadaten-Attributen hat den Personen bereits sehr beim Suchprozeß geholfen, sobald sie ihr wichtigstes Relevanzkriterium identifiziert hatten. Speziell bei einer großen Anzahl von Dokumenten wäre ein zusätzlicher Filtermechanismus gewinnbringend, um unerwünschte Dokumenttypen oder Domains ausscheiden zu können (z.B. nur Bücher aus der Domain .at oder nur Ordner).

7.7 Orientierung der Bücher

Von mehreren Beobachtern wurden bezüglich der Orientierung der Bücher in den Regalen Alternativen vorgeschlagen. Obwohl der Text am Buchrücken nur aus wenigen Worten besteht und auch ohne große Anstrengung bzw. Verdrehen des Kopfes gelesen und aufgenommen werden kann, argumentierten manche Anwender, daß bei aufeinanderliegenden Büchern gleichsam als Stapel die textuelle Information besser erkannt wird als bei vertikaler Ausrichtung. Tatsächlich stammt die vertikale Ausrichtung von Büchern vom Problem der Gravitation, weil nur so einfach Exemplare aus dem Regal entnommen werden können, dem die Organisation in Stapelform naturgemäß entgegenwirkt. Im Cyberspace von digitalen Bibliotheken besteht dieses Problem nicht; es wäre durchaus praktizierbar, die Bücher in einem Stapel vertikal aufeinander abzulegen bzw. zu visualisieren. Gerade das widerspricht aber der im Menschen verankerten Denkensweise des Aufbaus von Bibliotheksregalen und wurde auch von mehreren Anwendern schon angesprochen. Aus diesen Vorgaben heraus erscheint es als unumgänglich, die Frage der Orientierung im Detail zu analysieren.

7.8 3D-Repräsentation

Manche Begutachter haben den Vorschlag geäußert, eine dreidimensionale Repräsentation zur Darstellung der Bibliothek zu verwenden, in der sie sich bewegen und Bücher aufsammeln können. Da aber die meisten Anwender heutzutage noch keine 3D plug-ins auf ihrem Computersystem installiert haben und 3D-Brillen kaum Verbreitung gefunden haben, erscheint uns die zweidimensionale Darstellung als die gebräuchlichere Methode. Das wurde schlußendlich auch von den Anwendern anerkannt und der Pseudo 3D-Effekt in Form von Bücherregalen und Büchern als ausreichend empfunden. Die Erweiterung des libViewers durch dreidimensionale Grafiken und Animationen ist aber für die Zukunft eine nicht uninteressante Herausforderung.

Kapitel 8

Related Work

Mit dem massiven Ansteigen der Menge an Informationen, die in digitaler Form vorliegen, wurden hochentwickelte Methoden erschaffen, die die Interaktion und den Umgang mit elektronischen Informationssammlungen ermöglichen. Die Forschung im Bereich Information Retrieval (IR) hat Systeme hervorgebracht, die neben einfachen Suchabfragen auf große Datenbanken (bespielsweise nach dem Namen des Titels oder Autors) auch Volltextsuche in großen Textkörpern erlauben. Weiters wurden Methoden entwickelt, die das Suchen von Dokumenten zu bestimmten Themengebieten oder speziellen Konzepten erlauben ([9], [11], [25]). Neben dem „dokument retrieval“ wurden auch Systeme geschaffen, die ein Set von Dokumenten analysieren und so das Beantworten von Anfragen (question answering) erlauben ([1]). Diese Systeme versuchen, die Semantik von Anfragen zu untersuchen und so eine Antwort basierend auf der in der darunterliegenden Dokumentsammlung gespeicherten Information zu kreieren.

Während die meisten dieser Methoden die Selektion einer Untermenge von Einträgen in einer digitalen Bibliothek ermöglichen, bleiben wir immer noch mit dem Problem konfrontiert, daß

- das Identifizieren relevanter Einträge aus einer oft sehr großen Untermenge, die von den Suchmaschinen zurückgeliefert wird, schwierig ist und
- das Auffinden relevanter Informationen, wenn keine detailliertere Suchabfrage mehr erstellt werden kann, oft problematisch ist.

Dieses Problem bezieht sich auf das Blättern in Dokumentarchiven oder der Archiverforschung, welches im Gegensatz zum eigentlichen „document retrieval“ steht. Um in der Lage zu sein, eine Dokumentsammlung durch

Blättern bzw. interaktiver Suche („Browsing“) erforschen zu können, benötigen wir eine Visualisierung, die einen raschen Überblick über die enthaltene Information liefert. Die Notwendigkeit einer verbesserten visuellen Repräsentation von Biblioheken wurde schon in einigen Projekten adressiert, wobei versucht wurde, eine geeignete Zugriffsmöglichkeit auf digitale Dokumentsammlungen zu schaffen. In den folgenden Abschnitten wird auf einige dieser Projekte näher eingegangen und so ein Überblick über ähnliche Projekte im Bereich Bibliotheksvisualisierung hergestellt.

8.1 Bookhouse Projekt

Das Bookhouse Projekt ([18]) war eine der ersten Anwendungen von Metaphern im Bereich von digitalen Bibliotheken, wo eine Datenbank aus Dokumenten als Haus, bestehend aus verscheidensten Räumen repräsentiert wurde. Die primäre Absicht hinter den Bookhouse-Metaphern war es, Gelegenheitsanwendern die Möglichkeit zu geben, effizient in einer Datenbank zu navigieren. Es wurde dabei die Strategie verfolgt, die Datenbankstruktur in einen Kontext einzubetten, welcher direkt durch das User Interface visualisiert werden kann. Diese Methodik sollte es dem Anwender ermöglichen, schnell seine dafür notwendigen Fähigkeiten zu entwickeln ohne ein komplexes Vortraining zu absolvieren.

Eingangs bietet sich dem Besucher der digitalen Bibliothek das Bild eines Hauses mit geöffneter Tür, durch die Menschen in die Bibliothek hineinströmen. Nachdem man selbst mittels Mausklick das Haus betreten hat, gelangt man in einen Vorraum, wo der Anwender zwischen drei möglichen Datenbanken (Bücher für Kinder, Bücher für Kinder und Erwachsene bzw. Bücher nur für Erwachsene) wählen kann. Danach gelangt der User in einen weiteren Raum, wo je nach vorheriger Auswahl Kinder bzw. Erwachsene nach Büchern suchend abgebildet sind. Dort kann man dann einer Reihe von Suchstrategien folgen, die durch unterschiedliche Grafiken vermittelt werden. Es wurde beispielsweise die Metapher einer Uhr verwendet, um in der zeitlichen Dimension zu suchen oder ein Globus, wenn Dokumente nach ihrer geografischen Beziehung gesucht werden sollen. Je nach Auswahl des gewünschten Suchkriteriums wird dann eine Suchabfrage erstellt (hauptsächlich durch boolsche Operationen) und schlußendlich eine Menge von gefundenen Büchern zurückgeliefert, wobei die Ergebnisse samt der verfügbaren Metadaten in einem Textfenster ausgegeben werden.

Die Idee des Bookhouse Projekts findet sich auch heute noch in laufenden Projekten wieder. Das Intelligence Systems Laboratory (ISL), welches 1993 vom Institut für Computerwissenschaften der Roskilde Universität gegründet

Abbildung 8.1: Eingangshalle im Bookhouse

wurde, hat in Kooperation mit der Royal School of Librarianship (Dänemark) ein Projekt initiiert, wobei die Ideen des Bookhouse-Systems weiterverfolgt werden und auf dieser Basis eine WWW-Applikation implementiert wird. Ein Prototyp dieser Anwendung findet sich in dänischer Sprache im Internet unter http://www.isl.ruc.dk/boghus. Abbildung 8.1 zeigt ein Bild dieses Prototyps, das einen Eindruck von der grafischen Gestaltung des User Interfaces liefern soll.

Abseits der eingangs kurz beschriebenen Metaphern zur Visualisierung verschiedener Interaktionsmechanismen finden sich im Bookhouse Projekt keinerlei Metaphern zur Repräsentation von unterschiedlichen Dokumenttypen sowie zur Darstellung der verfügbaren Metadaten. Dies macht den libViewer zu einem komplementären und ergänzenden System.

8.2 Xerox PARC

Als Teil des Information Visualization Projects ([23]) wurden bei Xerox PARC[1] einige Visualisierungstechniken für information retrieval sowie zu Repräsentationszwecken von Informationen entwickelt. Über viele Jahre hinweg wurde die Anwendung von interaktiven Grafiken und Animationen erforscht um das Problem der Visualisierung von größeren Informationsmengen lösen zu können. Diese Arbeit basiert auf der Annahme, daß viele komplexe Aufgabenstellungen beim Arbeiten mit Informationen vereinfacht werden können, indem man komplexe kognitive Abläufe auf das menschliche Wahrnehmungssystem überträgt.

[1] http://www.parc.xerox.com/parc-go.html

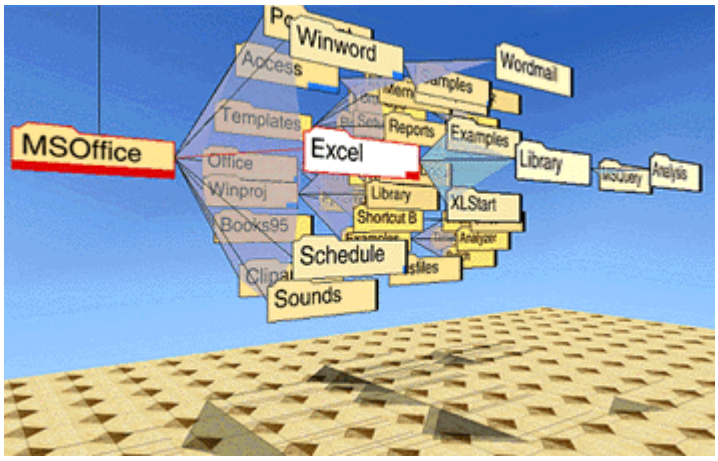

Abbildung 8.2: Visualisierung mit Conetree

Der Information Visualizer (IV) basiert auf der Metapher von dreidimensionalen Räumen, wo versucht wird, einen großen Arbeitsraum zu erschaffen, der wiederum viele einzelne Arbeitsbereiche enthält. Zusätzlich dazu wurden andere neuartige Bausteine entwickelt um neue Musterbeispiele im Feld von Benutzeroberflächen aufzuzeigen. Die IV Architektur hat die Entwicklung eines Sets von animierten Visualisierungstechniken für hierarchische Informationen ermöglicht, zu der Cone-Tree, Perspective Walls und Table Lens gehören. Abbildung 8.2 zeigt ein Beispiel einer Visualisierung mit Cone Tree.

Viele dieser Techniken nützen den Anzeigebereich ganzheitlich, wobei eine große Anzahl an Bildpunkten (Pixel) dem fokussierten Bereich zugeordnet werden und im Kontext dazu stehende Informationen weniger detailliert dargestellt werden. Dadurch wird es möglich, viel mehr Informationsobjekte auf einmal darstellen zu können, als es bei unverzerrter Visualisierung der Fall wäre. Beispielsweise können so die obersten 600 Knoten des Xerox-Organigramms auf einem Blick visualisiert werden, was anderenfalls ein 80-seitiges Dokument beanspruchen würde.

Eine weitere Anwendung im Information Visualizer, die von spezieller Bedeutung für digitale Bibliotheken ist, nennt sich „Butterfly Citation Browser“. Er wurde zur Visualisierung von wissenschaftlichen Artikeln geschaffen und besitzt vielfältige Modelle zur Darstellung von Informationen über Personen, Zeiten, Orte und Beziehungen durch Zitate. Butterfly erlaubt es dem Anwender, schnell durch die über Zitatverweise hergestellten Beziehungen zwischen Dokumenten zu navigieren, indem er mit virtuellen dreidimensionalen Objekten interagiert, die Artikel (papers) und Zitat-Beziehungen untereinander repräsentieren. Abbildung 8.3 soll einen Eindruck von der Benutzeroberfläche dieses einzigartigen Systems liefern.

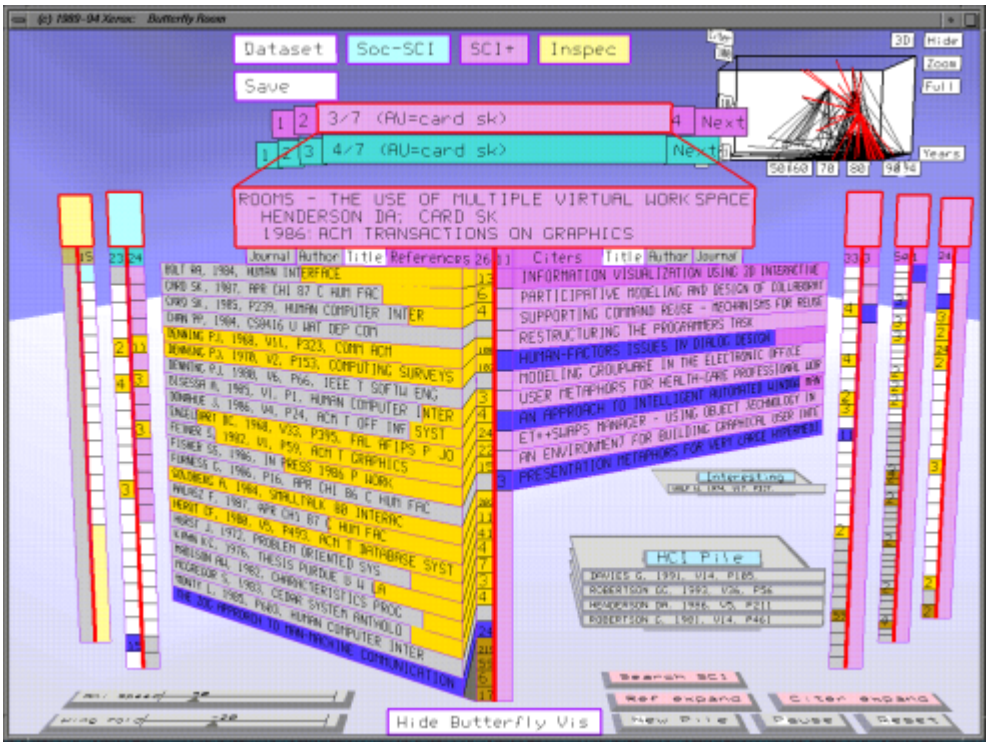

Abbildung 8.3: Butterfly Citation Browser

Bei allen Xerox PARC-Projekten zum Thema Informationsvisualisierung liegt der Fokus mehr auf der Visualisierung von Dokumentinhalten als den Metadaten, dem Schwerpunkt des in dieser Diplomarbeit vorgestellten Systems.

8.3 SOMLib System

Um die grundsätzliche Limitierung der meisten Suchmaschinen auf eindimensionale Rankinglisten zu überwinden, haben wir das SOMLib System entwickelt ([21], [22]), wobei ein Set von Dokumenten automatisch aufgrund des Inhalts in einer 2-dimensionalen Karte organisiert wird. Die „self organizing map“ (SOM) ([13]), ein populäres Modell eines unüberwachten, neuronalen Netzes wird dazu verwendet, ein inhaltbasierendes Clustering von Dokumenten auszuführen. Dieser Ansatz wurde wurde schon in einer Anzahl weiterer Projekte mit Fokus auf Dokumentklassifikation verwendet ([12], [16]). Ein Web-basierendes Interface erlaubt die interaktive Erforschung der Dokumente, wobei Informationsressourcen der Sammlung, die zu ähnlichen Themengebieten gehören, nahe beieinander liegen bzw. Cluster ausbilden. Diese Organisation ist der in realen Bibliotheken sehr ähnlich, wo Bücher ähnlicher Themengebiete in separaten Bücherregalen abgelegt sind. Ein Nachteil des Standard SOMLib Web Interfaces zur Kartendarstellung ist aber, daß keine weiteren Metainformationen zu den einzelnen Dokumenten extrahiert werden können. Zur Analyse dieser zweidimensionalen Karten (SOMs) wurde im Rahmen des SOMLib-Projektes auch der somViewer entwickelt, um die Zusammengehörigkeit einzelner Cluster zu erforschen und die Struktur der Karten abstrakt visualisieren zu können.

8.4 Sartiaux Collection

Die CNAM Bibliothek (Conservatoire National des Arts et Mtiers, Frankreich) hat ein Projekt initiiert, in welchem eine antiquarische Kollektion von wissenschaftlichen und technischen Büchern digitalisiert werden soll ([5]). Diese Kollektion ist Teil der Sartiaux Collection und umfaßt ca. 1200 Werke über Elektrizität und Physik vom 17. bis zum 19. Jahrhundert. Endziel dieses Projekts ist es, die digitale Bibliothek via World Wide Web verfügbar zu machen.

Der Hintergrund dieser Forschung ist es, durch möglichst realistische Nachbildung (Digitalisierung) des Buchrückens dem Literatursuchenden die Möglichkeit zu geben, abseits von textbasierender Metainformation auf ei-

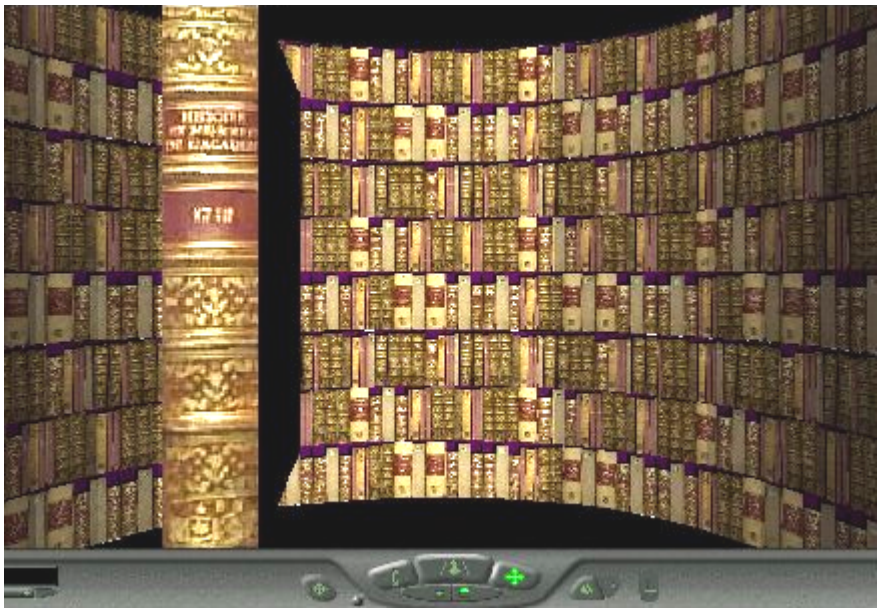

Abbildung 8.4: zylindrisches Bücherregal mit VRML

ne intuitiv interpretierbare Darstellung in Form von digitalisierten Grafiken zugreifen zu können. Es wird argumentiert, daß beispielsweise mit Kenntnis des Themengebiets und des Stils, wie das Werk eingebunden ist, ein Experte erkennen kann, um welche Bücher es sich handelt. Der geschulte Experte ist so in der Lage, durch derartige Visualisierungstechniken wesentlich schneller die gewünschten Ressourcen zu finden.

Wegen der Verbreitung günstiger und leistungsfähiger Methoden in der 3D-Visualisierung wie VRML (Virtual Reality Markup Language) konnten in diesem Projekt einige Prototypen mit dreidimensionalen Grafiken geschaffen werden. Dabei wurden auch Überlegungen zur Bücheregal-Metapher gemacht, die Modelle des Regals als Ebene, Zylinder und Spirale umfassen. Abbildung 8.4 zeigt einen Screenshot des zylindrischen Bücherregals, wo digitalisierte Grafiken der Buchrücken als Texturen auf einen Zylinder projiziert wurden. Eine Hälfte der Bücher ist in diesem Bild sichtbar, die andere Hälfte befindet sich hinter dem Betrachter. Ein vom Anwender selektiertes Buch wird besonders hervorgehoben und steht im Bücherregal nach vorne.

8.5 Insight

InSight ist ein kommerzielles Produkt von Silicon Graphics[2] und wurde zum Anzeigen von Online-Manuals auf SGI-Rechnern entwickelt. Das Besondere an diesem Hypertext-Utility ist es, daß zur Visualisierung der verfügbaren Manuals die Buch-Metapher verwendet wird (siehe Abbildung 8.5). Je nachdem, für welche Benutzergruppe ein Manual erstellt wurde (End-User, Administrator, Entwickler), finden sich am Buchrücken unterschiedliche Muster bzw. Farben. Dadurch kann sofort vom Betrachter intuitiv eine Zuordnung zu bestimmten Genres erfolgen. Die Orientierung der Bücher in den Regalen ist vertikal, d.h. die Bücher sind in Form eines Bücherstapels organisiert. Durch „drag-and-drop“ kann ein Buch zum persönlichen Bücheregal hinzugefügt werden und mittels eines Suchdialogs können auch Abfragen an das System gestellt werden. Mittels Doppelklick auf ein Buch wird dieses geöffnet und steht dann online zum Lesen zur Verfügung.

Außer den oben erwähnten Metaphern (Farbe und Muster am Buchrücken) zur Darstellung einer Zielgruppe verwendet InSight keine weiteren Metaphern zur Visualisierung von Metadaten. Dabei wäre es aber leicht möglich, beispielsweise die Größe eines Manuals auf die Buchdicke abzubilden, um dadurch das intuitive Auffinden einer gewünschten Anleitung zu verbessern.

8.6 The Shape of Shakespeare

Ein Ansatz zur Visualisierung von Texten wurde in [24] vorgestellt, wo mit impliziten Flächen versucht wird, Inhalte von Dokumenten und Beziehungen zwischen diesen zu repräsentieren. Dokumente, die zu gleichen Themengebieten gehören, können so anhand ihrer dreidimensionalen Form identifiziert werden.

Eine einfache Methode wurde definiert, um den Inhalt eines Dokuments auf dessen Form abzubilden. Durch dieses Verfahren wird es möglich, Ähnlichkeiten in Bezug auf Dokumentinhalte zu erkennen bzw. Unterschiede festzustellen. Eine andere Methode vollzieht visuelles Clustering, wobei mit Hilfe von „similarity scoring“ Dokumente in einem dreidimensionalen Raum angeordnet werden. Dokumente, die in starkem Bezug zueinander stehen, verschmelzen zu einer zusammenhängenden Form. Durch diese Techniken können sowohl eigenstehende Dokumente als auch aus dem Textkörper extrahierte Metadaten visualisiert werden. Mit der Kombination beider Methoden besteht die Möglichkeit, lokale und globale Information zum Kontext der

[2] http://www.sgi.com/

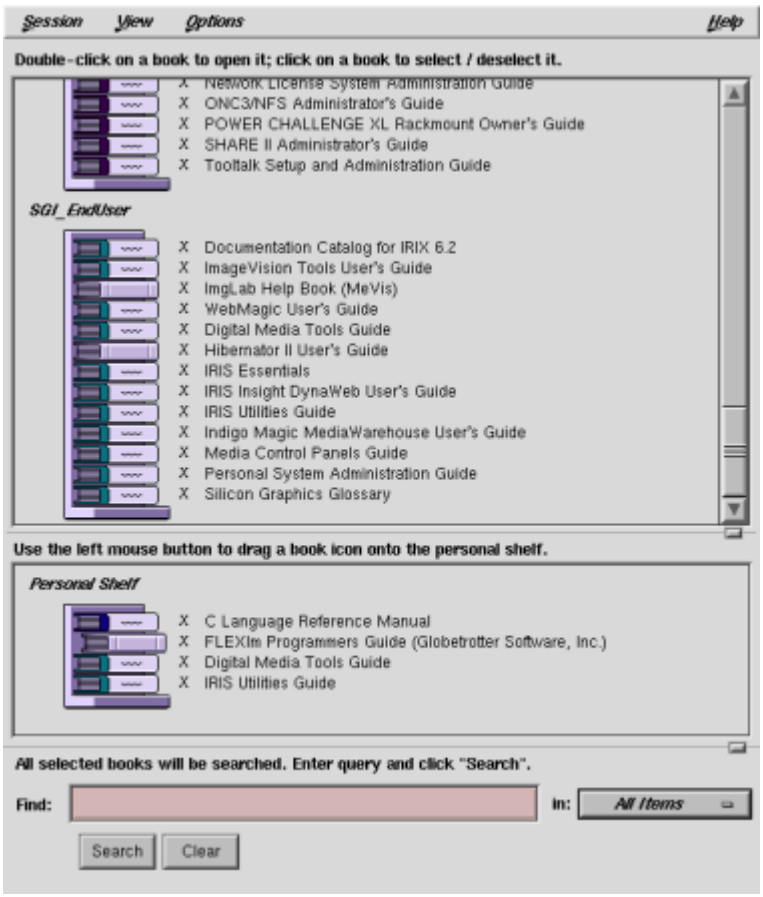

Abbildung 8.5: Benutzeroberfläche von InSight

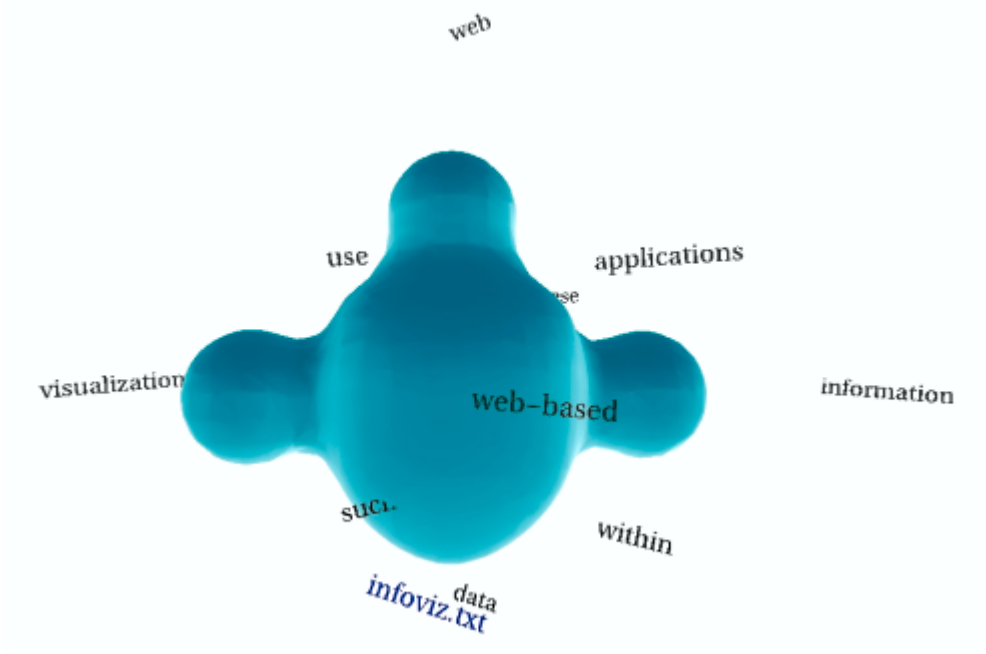

Abbildung 8.6: Blobby-Text Visualisierung

Dokumente darzustellen.

Ein Prototyp, welcher die Ideen aus der Textvisualisierung und dem „implizit surface modeling“ umzusetzen versucht, wurde in VRML implementiert und bedient sich sogenannter „blobby models“. Abbildung 8.6 zeigt ein einzelnes Dokument, welches als blobby text visualisiert ist. Das darunterliegende Dokument ist ein paper mit dem Namen „Web-based Information Visualization“. Die am häufigsten vorkommenden Wörter sind „web“, „visualization“ und „information“, was auch mit dem Titel konsistent ist. Diese Terme korrespondieren mit den 3 erkannbaren Ausbuchtungen auf der linken, rechten sowie oberen Seite des Modells.

8.7 Zusammenfassung

Alle in diesem Kapitel vorgestellten Projekte befassen sich mit den Themen Dokument- und Bibliotheksvisualisierung bzw. der Darstellung von „information space“. Die durch die physische Repräsentation eines Objektes transportierte Information wird dabei aber nicht vermittelt, d.h. die Information, die intuitiv durch das Aussehen eines Objektes wie z.B. bei einem Hardcover-Buch oder einem Buch mit Papiereinband oder einer Videokasette übertragen wird, geht verloren. Aus diesem Grund werden wichtige Metadaten in textu-

eller Form ausgegeben, mit allen Konsequenzen auf die intuitive Erfaßbarkeit der darzustellenden Metainformationen.

Kapitel 9

Zusammenfassung und Ausblick

In dieser Diplomarbeit haben wir das libViewer-System vorgestellt, ein generisches Interface zur Visualisierung elektronischer Dokumentsammlungen. Durch dieses Interface wird es möglich, Dokumente basierend auf der vorhandenen Metainformation durch eine intuitive, grafische Darstellung zu repräsentieren. Während bei den meisten Schnittstellen zu digitalen Bibliotheken Metadaten zu den einzelnen Dokumenten (Typ, Alter, Größe, etc.) vorwiegend textuell dargestellt werden, haben wir Metaphergrafiken verwendet, um Metainformationen auf selbsterklärende Art und Weise zu vermitteln.

Der libViewer als grafische Benutzerschnittstelle ist ein Java-Applet, welches in einem Zeitraum von ungefähr 12 Monaten entwickelt wurde und einzelne Server kontaktiert, um digitale Dokumentsammlungen laden zu können. Der Server vollzieht nach Erhalt einer Anfrage ein auf den Anwendungsbereich abgestimmtes Mapping von Metadaten auf die im libViewer realisierten Metaphern und sendet die konvertierte Dokumentliste an den aufrufenden Client zurück. Wir haben zur Demonstration der Fähigkeiten unseres Systems zwei Server implementiert. Der Dublin-Core Server macht es möglich, Bibliotheksdateien im Dublin-Core Format laden zu können und besitzt eine Reihe von Mappings zur Demonstration der vielfältigen Information, welche durch Metaphergrafiken vermittelt werden kann. Der AltaVista-Server kontaktiert die Suchmaschine AltaVista mit einer Suchabfrage, parst die zurückgelieferten Dokumente bzw. Hits und vollzieht ein Mapping von den gewonnenen Metadaten auf die im libViewer realisierten Metaphern. Dadurch soll dem Anwender eine intuitive, verbesserte visuelle Darstellung des Suchergebnisses zur Verfügung gestellt werden.

In Kapitel 1, der Einleitung, haben wir zuerst die Idee beschrieben, die uns zur Realisierung eines derartigen Systems bewegt hat. Dabei wurde auch auf mögliche Benefits für die Benutzer von digitalen Bibliotheken eingegangen. In diesem Kapitel findet sich auch eine kurze Entwicklungsgeschichte, in der

wir den kreativen Prozeß näherzubringen versuchten.

Kapitel 2 vermittelt Grundlagen zu digitalen Bibliotheken. Neben der Geschichte von „Digital Libraries“ haben wir auch Probleme bei der Visualisierung von digitalen Dokumentsammlungen aufgezeigt. Anbei findet sich auch eine Analyse, warum die Verbesserung der grafischen Repräsentation im Bereich Bibliotheksvisualierung eine Herausforderung darstellt.

Das dritte Kapitel hat sich mit Metadaten für digitale Bibliotheken beschäftigt. Dabei wurde das Wesen von Metadaten näher betrachtet und gängige Metadaten-Standards (BibTeX, MARC, Dublin Core, etc.) vorgestellt.

„Metaphern für Informationsressourcen“ ist der Titel für eines der Kernkapitel (Kapitel 4) dieses Werkes. Eingangs haben wir notwendige Designrichtlinien beschrieben, die bei der Visualisierung von Informationsressourcen durch Metaphergrafiken zu berücksichtigen sind. Abschnitt 2 dieses Kapitels bietet eine detaillierte Beschreibung aller von uns identifizierten Metaphern zur Visualisierung von Metadaten in digitalen Bibliotheken. Der letzte Teil dieses Kapitels beschreibt anhand von Beispielgrafiken die Umsetzung der identifizierten Metaphern im libViewer.

Kapitel 5 (Technische Systembeschreibung) stellt für die Erweiterungsfähigkeit des libViewer-Systems eine umfassende Klassenbeschreibung zur Verfügung. Dabei wurden alle Java-Klassen des libViewers, DCServers und des AVServers berücksichtigt. In die technische Systembeschreibung wurde auch das libViewer-libServer Protokoll inkludiert, das den Datenaustausch zwischen Client und Server regelt.

In Kapitel 6 haben wir das User Interface des libViewers beschrieben. Nach einer detaillierten Auflistung und Erklärung der einzelnen grafischen Komponenten des libViewer-Applets folgt eine Beispielsession, wo die Interaktion mit dem DCServer sowie mit dem AVServer anhand von Screenshots illustriert wird.

Die Ergebnisse der von uns durchgeführten „Usability Evaluation“ konnten in Kapitel 7 nachvollzogen werden. Diese Erkenntnisse haben wir durch die Befragung unterschiedlicher Anwendergruppen gewonnen, zu denen u.a. unversierte Computeranwender, Bibliothekare und Experten auf dem Gebiet „User Interface Design“ zählen.

Schlußendlich gab Kapitel 8 einen Überblick über verwandte Arbeiten im Bereich Informationsvisualisierung. Projekte wie das „Bookhouse Project“, das SOMLib System oder das kommerzielle Produkt „Insight“ wurden ebenso in diese Aufstellung miteinbezogen wie die Forschungsarbeiten bei Xerox PARC.

Durch die vielversprechenden Resultate der Anwenderbefragung motiviert, planen wir für die Zukunft eine Reihe weiterer Server zur Anbin-

dung großer Bibliothekssysteme. Dazu gehört beispielsweise ein Server für das OPAC-System, welches für Suchzwecke alle österreichischen Bibliothekssysteme verbindet. Eine weitere, sehr vielversprechende Anwendung unseres Systems ist ein Server zur Anbindung des Internet-Buchhandels Amazon. Neben der Implementierung weiterer Server planen wir auch eine Erweiterung des libViewer-libServer Protokolls. Dadurch wird es möglich, daß der Server eine Beschreibung des Query-Interfaces an den libViewer senden kann und so die Interaktionsmöglichkeiten wesentlich erweitert werden können. Dies ist vor allem für den Amazon.com-Server interessant, wo beispielsweise Suchabfragen durch detailliertere Kriterien effizienter oder die Möglichkeiten zur Interaktion vielfältiger gestaltet werden können. Durch all diese geplanten Erweiterungen soll vor allem erreicht werden, daß wir unser System hinsichtlich seiner Benutzbarkeit besser beurteilen können, da es dann einer breiteren Anwenderschicht zur Verfügung steht.

Literaturverzeichnis

[1] M. D. Aliod, J. Berri, and M. Hess. A real world implementation of answer extraction. In *Proc. Workshop on Natural Language and Information Systems (NLIS'98) in Conjunction with DEXA'98*, Vienna, Austria, 1998.

[2] Vannevar Bush. As we may think. *Atlantic Monthly*, (176):101 – 108, 1945.

[3] H. Chernoff. The use of faces to represent points in k-dimensional space graphically. *Journal of the American Statistical Association*, (68):361–368, 1973.

[4] W. G. Cole and J. G. Stewart. Metaphor graphics to support integrated decision making with respiratory data. *Intl. Journal of Clinical Monitoring and Computing*, 10:91–100, 1993.

[5] P. Cubaud, C. Thiria, and A. Topol. Experimenting a 3d interface for the access to a digital library. In *Proc. ACM Conf. on Digital Libraries (DL98)*, Pittsburgh, PA, 1998.

[6] Karen Drabenstott. Analytical review of the library of the future. *Council on Library Resources*, 1993.

[7] Richard K. Futura Edward A. Fox, Robert M. Akscyn. Digital libraries. *Communications of the ACM*, 38(4):23 – 28, 1995.

[8] N. Gershon, S. Eick, and S. Card. Information visualization. *Interactions*, 2, March and April, 1998.

[9] U. Hahn, M. Klenner, and K. Schnattinger. Automatic concept acquisition from real-world texts. In *AAAI Spring Symp. on Machine Learning in Information Access*, Stanford, USA, 1996.

[10] Stephen P. Harter. What is a digital library? definitions, content and issues. In *Proc. of the Int'l. Conf. on Digital Libraries and Information Services for the 21st Century (KOLISS DL '96)*, Seoul, Korea, September 10-13, 1996.

[11] M. A. Hearst. Using categories to provide context for full-text retrieval results. In *Proceedings of RIAO, Intelligent Multimedia Information Retrieval Systems and Managemen*, New York, NY, 1994.

[12] S. Kaski, T. Honkela, K. Lagus, and T Kohonen. WEBSOM—self-organizing maps of document collections. In *Elsevir Publ.* Elsevir Publications, 1997.

[13] T. Kohonen. *Self-Organizing Maps*. Springer Verlag, Berlin, Germany, 1995.

[14] F. Wilfrid Lancaster. Toward paperless information systems. *Academic Press*, 1978.

[15] J.C.R. Licklider. Library of the future. *M.I.T Press*, 1965.

[16] X. Lin, D. Soergel, and G. Marchionini. A self-organizing semantic map for information retrieval. In *Proc. Int'l ACM SIGIR Conf. on R & D in Information Retrieval*, Chicago, IL, 1991.

[17] Theodor H. Nelson. Computer lib. *Nelson*, 1974.

[18] A. M. Pejtersen. A library system for information retrieval based on cognitive task analysis and supported by an icon-based interface. In *Proc. of the ACM SIGIR Conference on Research and Developement in Information Retrieval (SIGIR'89)*. ACM, 1989.

[19] A. Rauber and H Bina. A metaphor graphics based representation of digital libraries on the World Wide Web: Using the libViewer to make metadata visible. In *Proc. DEXA-Workshop on Web-based Information Visualization (WebVis99)*, Florence, Italy, 1999.

[20] A. Rauber and Harald Bina. libviewer - libserver interface description. Technical report, Institute for Software Technique, Technical University of Vienna, http://www.ifs.tuwien.ac.at/ifs/, 1999.

[21] A. Rauber and D. Merkl. Creating an order in distributed digital libraries by integrating independent self-organizing maps. In *Proc. Int'l Conf. on Artificial Neural Networks (ICANN'98)*, Skövde, Sweden, 1998.

[22] A. Rauber and D. Merkl. The SOMLib digital library system. In *Proc. European Conference on Digital Library Systems*, Paris, France, 1999. LNCS, Springer Verlag.

[23] G. R. Robertson, S. K. Card, and J. D. Mackinlay. Information visualization using 3d interactive animation. *Communications of the ACM*, 36:57 – 71, April 1993.

[24] R.M. Rohrer, D.S. Ebert, and J.L. Sibert. The shape of shakespeare: Visualizing text using implicit surfaces. In *IEEE Symposium on Information Visualization (INFOVIS'98)*, North Carolina, 1998.

[25] G. Salton, J. Allan, and C. Buckley. Approaches to passage retrieval in full text information systems. In *Proc. of the Int'l. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 49 – 58, Pittsburg, USA, 1993.

[26] E. R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, Connecticut, 1983.

[27] E. R. Tufte. *Envisioning Information.* Graphics Press, Connecticut, 1990.