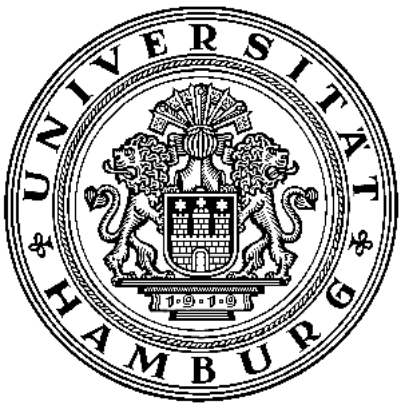


Universität Hamburg
FB Informatik
Datenbanken und
Informationssysteme

STUDIENARBEIT

CobWeb

Ein Prototyp für ein personalisiertes, intranet-orientiertes Literaturinformationssystem zur kooperativen Bearbeitung von Bibliographien

November 1997

Christian Pesch
Buchenstraße 30d
22885 Barsbüttel

Tel.: 040 / 670 41 79

Betreuer:
Prof. Dr. Joachim W. Schmidt

Inhaltsverzeichnis

1 Einleitung und Motivation **1**

1.1 Aufgabenstellung 1

1.2 Struktur der Arbeit 2

2 Methoden und Werkzeuge zur Literaturverwaltung **3**

2.1 Traditionelle Arbeit mit BibTeX-Dateien 3

2.1.1 BibTeX-Format 4

2.1.2 Bearbeitung 5

2.1.3 Werkzeuge 5

2.1.4 spezielle BibTeX-Tools 6

2.2 Verwandte Projekte 6

2.2.1 MeDoc 7

2.2.2 Canonical Bibliography Format 7

2.2.3 APA Style of Citation 7

3 Anforderungen an die werkzeugunterstützte, kooperative Arbeit mit Literaturreferenzen **8**

3.1 Arbeitsszenarien 8

3.1.1 Aufbau eines eigenen Bestandes 8

3.1.2 Pflege eines Bestandes 9

3.1.3 Annotation eines Bestandes 9

3.1.4 Recherche in einem Bestand 9

3.1.5 Akquisition neuen Materials 10

3.1.6 Selektion von Literaturreferenzen 11

3.1.7 Weiterverarbeitung selektierter Literaturreferenzen 11

3.1.8 Zusammenführung separat gewachsener Bestände 11

3.1.9 Abgleich mit separat gewachsenen Beständen 11

3.1.10 Erweiterung des Formats 12

3.2 Benutzergruppen und Arbeitsmodi 12

3.3 Anforderungen an eine Werkzeugunterstützung 14

3.3.1 Grundsätzliche Anforderungen 14

3.3.2 Konkrete Anforderungen 15

3.4 Zugriffsrechte und Rollen 17

4 Verwendete Komponenten, Werkzeuge und Sprachen **18**
4.1 Methoden und Werkzeuge zur Modellierung . . . 18
4.1.1 OMT . . . 19
4.1.2 StP/OMT . . . 20
4.2 Komponenten und Sprachen zur Implementierung . . . 20
4.2.1 Java . . . 21
4.2.2 Client-Server Programmierung mit Java . . . 22
4.3 Methoden zur Erstellung verteilter, persistenter Anwendungen . . . 23
4.3.1 Verteilte Objekte mit RMI . . . 23
4.3.2 Plattformunabhänige Oberflächen mit AWT . . . 24
4.3.3 Datenbankanbindung mit JDBC . . . 24
4.3.4 Relationale Datenbanken . . . 26
4.4 Sicherheit . . . 26

5 Modellierung einer Arbeitsumgebung für die kooperative Bearbeitung von Literaturreferenzen **27**
5.1 BibTeX-Analyse . . . 27
5.1.1 Definition des BibTeX-Formats . . . 28
5.1.2 Literaturreferenzen im BibTeX-Format . . . 28
5.1.3 Definition der Schlüssel . . . 30
5.2 Objektmodell . . . 30
5.2.1 Überblick über die Modellierung . . . 30
5.2.2 Genauere Modellierung . . . 32
5.3 Dynamisches Modell . . . 34
5.4 Funktionales Modell . . . 35

6 CobWeb **37**
6.1 Architekturentwurf . . . 38
6.2 Implementierungsdetails der Architektur . . . 39
6.2.1 Client . . . 39
6.2.2 Server . . . 40
6.2.3 Datensichten . . . 40
6.2.4 Objektverteilung . . . 42
6.3 Modellierung der Implementation . . . 43
6.4 Details der Implementierung . . . 45
6.4.1 Frame-Klassen . . . 45
6.4.2 Ereignisbehandlung . . . 46
6.4.3 Layout-Manager . . . 47
6.4.4 GridBagLayout-Manager . . . 47
6.4.5 Filterung, Matching . . . 49

7 Zusammenfassung und Ausblick **50**
7.1 Zusammenfassung . . . 50
7.2 Bewertung der eingesetzten Werkzeuge . . . 52
7.3 Ausblick . . . 53

A BibTeX-Definition **55**
A.1 Typen von Literaturreferenzen in BibTeX 55
A.2 Attribute von Literaturreferenzen in BibTeX 56

B OMT-Diagramme **59**

C Quelltexte **66**
C.1 Klasse MyFrame 66
C.2 Interface ReferenceStore 67

Literaturverzeichnis **67**

Abbildungsverzeichnis

2.1 Beispiel einer Literaturreferenz in BibTeX 4

4.1 Zitat des Java-Entwicklers James Gosling 21
4.2 Client-Server-Modell 22
4.3 JDBC-Architektur 25

5.1 Klasse reference 31
5.2 Hierarchie der Referenzen 32
5.3 Klasse book 33
5.4 Zustände einer Literaturreferenz 34
5.5 Arbeiten mit einem exportierten Bestand 35
5.6 Arbeiten mit dem Bestand 36

6.1 Architekturüberblick 38
6.2 Technische Komponenten des Clients 39
6.3 Technische Komponenten des Servers 40
6.4 Datensichten 41
6.5 Objektverteilung 42
6.6 Hierarchie der Persistenzklassen 43
6.7 Erster Ansatz für die Hierarchie der Persistenzklassen 44
6.8 Realisierte Hierarchie der Persistenzklassen 45
6.9 Kommunikation zwischen Daten- und Präsentationsklassen 46

B.1 Klassenhierarchie 60
B.2 Klasse reference 60
B.3 Klasse article 61
B.4 Klasse book 61
B.5 Klasse inbook 62
B.6 Klasse booklet 62
B.7 Klasse collection 62
B.8 Klasse incollection 63
B.9 Klasse proceedings 63
B.10 Klasse inproceedings 63
B.11 Klasse manual 64
B.12 Klasse mastersthesis 64
B.13 Klasse techreport 65
B.14 Klasse unpublished 65

Kapitel 1

Einleitung und Motivation

Die Erstellung einer wissenschaftlichen Publikation ist ein kreativer Prozeß. Unabhängig von der Vorgehensweise des Autors sammeln sich bei der Erstellung einer solchen Publikation im Laufe des Entstehungsprozesses Notizen und Anmerkungen, mit denen Details zur Publikation festgehalten werden. Diese Informationen sind zumeist unstrukturiert und in Umfang und Inhalt schwer einzuordnen, da das einzig Verbindende ist, daß sie im Rahmen der Publikation Verwendung finden könnten. Eine Leistung bei der Erstellung einer wissenschaftlichen Publikation besteht in der Aufbereitung und Strukturierung dieser Informationen.
Neben Informationen, die in engem inhaltlichem Zusammenhang zu der Publikation stehen und deshalb für eine Werkzeugunterstützung kaum in strukturierter Form faßbar sind, sind Informationen über interessante Fundstellen in der Literatur für die vorliegende Arbeit von besonderem Interesse. Sie können und werden unabhängig von der Arbeitsweise und dem Vorgehen ihres Erstellers weitgehend einheitlich strukturiert abgefaßt. Eine Werkzeugunterstützung bietet sich hier an.
Besonders in Arbeitsgruppen, in denen viele wissenschaftliche Publikationen verfaßt werden, lohnt es sich, das Wissen über interessante Fundstellen in der Literatur systematisch zu sammeln und zu pflegen. Beim Erstellen einer neuen Publikation, die inhaltliche Überschneidungen mit bereits erstellten aufweist, kann auf bestehende Literaturreferenzen zurückgegriffen werden. Die Mitglieder einer Arbeitsgruppe können durch den Austausch und das Teilen von Literaturreferenzen unnötige Arbeit vermeiden und durch diese Kooperation profitieren. Durch den Austausch zwischen Arbeitsgruppen können neue Ideen und Anregungen in Form von interessanten Fundstellen in der Literatur entstehen. Bei der Suche nach geeigneter Literatur kann bereits bekannten Literaturreferenzen nachgegangen werden, um Neuerungen oder Änderungen zu entdecken und damit weitere Literaturreferenzen zu den bestehenden hinzufügen. Eine Werkzeugunterstützung für die kooperative Arbeit mit Literaturreferenzen bietet daher einen besonderen Mehrwert für die Benutzer.

1.1 Aufgabenstellung

Das Ziel der vorliegenden Arbeit ist zum einen die Erstellung eines Kataloges von Anforderungen, die an eine Arbeitsumgebung für die werkzeugunterstützte, kooperative Arbeit mit Literaturreferenzen zu stellen sind. Durch die Betrachtung der Arbeitsszenarien bei der kooperativen Arbeit mit Literaturreferenzen, der Benutzer der zu erstellenden Arbeitsumgebung und ihrer Arbeitsmodi soll versucht werden, ein möglichst exaktes Verständnis der werkzeuggestützten, kooperativen Arbeit mit Literaturreferenzen zu erreichen. Als Ergebnis der Analyse soll ein Katalog von Anforderungen an eine Arbeitsumgebung für die kooperative Arbeit mit Bibliogra-

phien erstellt werden. Die Anforderungen stellen den Ausgangspunkt für den Softwareentwurf, der den zweiten Teil der Arbeit ausmachen wird, dar. Mit einer Softwareentwurfsmethode sollen im Rahmen der Modellierung eine Reihe von Modellen erstellt werden, die an den aufgestellten Anforderungen gemessen werden können. Die größte Bedeutung unter den Modellen wird einem geeigneten Datenmodell zukommen, da es die Basis für weitere Modelle ist, die im Prozeß des Softwareentwurfs erstellt werden. Deshalb soll ein besonderes Augenmerk auf das Datenmodell gelegt werden. Es soll im Rahmen einer prototypischen Anwendung entwickelt, in dem Prototypen angewandt und dadurch weiter verfeinert werden. Die Gründe für die Entscheidung, einen Prototypen zu implementieren, und die Erfahrungen, Erkenntnisse und Schwierigkeiten, die bei der Erstellung des Prototypen aufgetreten sind, werden beschrieben.

1.2 Struktur der Arbeit

Die Arbeit befaßt sich mit einer Arbeitsumgebung für die werkzeuggestützte, kooperative Arbeit mit Literaturreferenzen. Kapitel 2 stellt die Methoden und Werkzeuge zur Literaturverwaltung vor, die derzeit zur Arbeit mit Literaturreferenzen eingesetzt werden. In Kapitel 3 werden charakteristische Arbeitsszenarien mit besonderem Hinblick auf den Aspekt der Kooperation erarbeitet. Diese Arbeitsszenarien sollen die Grundlage für einen Katalog von Anforderungen an eine Arbeitsumgebung für die kooperative Arbeit mit Literaturreferenzen sein. Als Basis für die folgenden Kapitel werden in Kapitel 4 die verwendeten Komponenten, Werkzeuge und Sprachen beschrieben. In Kapitel 5 werden die Modelle für die Verwaltung von Literaturreferenzen entworfen. Zur Evaluation der entworfenen Modelle wird eine prototypische Anwendung entwickelt. Die Erfahrungen, Erkenntnisse und Schwierigkeiten bei der Realisierung des Prototypen werden in Kapitel 6 vorgestellt. Die Zusammenfassung und der Ausblick in Kapitel 7 bilden den Abschluß dieser Arbeit.

Kapitel 2

Methoden und Werkzeuge zur Literaturverwaltung

Das Erstellen wissenschaftlicher Publikationen umfaßt als einen wesentlichen Aspekt die Suche geeigneter Literatur und das Aufstellen eines Literaturverzeichnisses mit Literaturreferenzen, die verwandte Publikationen referenzieren. Die Verknüpfung der Publikation mit anderen stellt einen wichtigen Teil der wissenschaftlichen Arbeit dar.
Für das Schreiben wissenschaftlicher Publikationen wurden bereits Ende der siebziger und Anfang der achtziger Jahre Satzsysteme entwickelt und in größerem Umfang bei der Erstellung wissenschaftlicher Publikationen eingesetzt. Das Satzsystem *TeX* mit seiner Erweiterung *LaTeX* konnte sich vor allem im universitären Bereich durchsetzen. Schon bald entstand die Anforderung, neben den Verzeichnissen über Inhalt und Abbildungen der Publikation auch das Literaturverzeichnis automatisch vom TeX-System verwalten zu lassen. Zu diesem Zweck wurde, getreu einem Grundkonzept von TeX, Inhalt und Layout zu trennen, das Programm *BibTeX* entwickelt. Es erlaubt die Definition von Literaturreferenzen, die in verschiedene Typen unterteilt und mit festgelegten Attributen versehen werden. Durch die weite Verbreitung von TeX wurde das BibTeX-Format zu einem Standard für die Verwaltung von Literaturreferenzen.
Dieses Kapitel beschreibt die traditionelle Arbeit mit BibTeX und den Literaturreferenzen, die im BibTeX-Format in Daten abgelegt werden. Weiterhin werden zu der vorliegenden Arbeit verwandte Projekte zur Sammlung, Verwaltung, Bearbeitung, Recherche und Verteilung von Literaturreferenzen vorgestellt.

2.1 Traditionelle Arbeit mit BibTeX-Dateien

BibTeX dient zur automatischen Generierung von Literaturverzeichnissen analog zur Erstellung der Inhalts-, Abbildungs- und Zeichnungsverzeichnisse durch TeX. Durch den verbreiteten Einsatz des TeX-Satzsystems verbreitete sich auch dessen Zusatz BibTeX mit seinem Format für die Darstellung von Literaturreferenzen. Es wird heutzutage trotz seiner Schwächen und besserer Alternativen häufig auch für die Verwaltung, Speicherung und den Austausch von Literaturreferenzen verwendet. Dieser Abschnitt befaßt sich mit dem Format der Literaturreferenzen in BibTeX-Dateien, der Bearbeitung dieser Literaturreferenzen und einer Reihe von Werkzeugen für die komfortablere Bearbeitung und Recherche in BibTeX-Dateien.

```
% Literaturreferenzen für die Studienarbeit

@TECHREPORT(RFPG96,
  AUTHOR      = {Rice, James and Farquhar, Adam and
                 Piernot, Philippe and Gruber, Thomas},
  TITLE       = {Lessons Learned Using the Web
                 as an Application Interface},
  INSTITUTION = {Knowledge Systems Lab.},
  ADDRESS     = {701 Welch Rd, bldg. C, Palo Alto, CA 94304},
  YEAR        = {95},
  MONTH       = {sep},
  TYPE        = {Technical paper},
  OWNER       = {Pesch, Christian},
  REG-DATE    = {27.09.96},
  KEYWORDS    = {WWW, Application Interface, HTML, GUI},
  DIRECTORY   = {http://www-ksl.stanford.edu},
  KEY         = {KSL-95-69},
  COMMENT     = {Ideen, um intuitive Oberflächen
                 mit HTML zu erstellen}
)
```

Abbildung 2.1: Beispiel einer Literaturreferenz in BibTeX

2.1.1 BibTeX-Format

Beim Schreiben von wissenschaftlichen Arbeiten wird häufig das Satzsystem TeX [Kopka 92] mit seiner Erweiterung LaTeX [Lamport 85b] genutzt, für das ein BibTeX [Lamport 85a] genanntes Zusatzprogramm existiert. Dieses definiert ein einheitliches Format für Literaturreferenzen, das vom Benutzer erweitert werden kann. Außerdem sorgt es ohne manuelle Eingriffe dafür, daß die zitierte Literatur sortiert und einheitlich formatiert in einem Literaturverzeichnis aufgelistet wird.
Die Literaturreferenzen für das BibTeX-Programm werden als einfache, flache Textdateien abgelegt. Die Strukturierung erfolgt über Schlüsselworte, eine Menge von Trennsymbolen und Name-Wert-Paaren. Eine Literaturreferenz wird von einem Sonderzeichen gefolgt von einem Schlüsselwort eingeleitet, das die Art der Literaturreferenz festlegt. Die Typen von Literaturreferenzen sind in BibTeX vordefiniert. Für jeden Typ sind eine festgelegte Anzahl von vorgeschriebenen und optionalen Attributen (s. Tabelle A.2) definiert. Nicht im BibTeX-Format definierte Attribute werden ignoriert. Die Konsistenz von Attributen, die wie z.B. *address* und *organization* zusammengehören, wird nicht gewährleistet. Über sog. *@STRING*-Makros (siehe [Lamport 85a]) können Kürzeln längeren Texten zugeordnet werden. Das BibTeX-Programm expandiert die Kürzel in einer BibTeX-Datei dann automatisch zu dem angegebenen längeren Text.
Für jede Literaturreferenz muß ein Typ (s. Tabelle A.1) und ein Schlüssel festgelegt werden. Unter dem Schlüssel wird die Literaturreferenz im LaTeX-Text referenziert. Der Schlüssel besteht aus einer beliebigen Folge von Buchstaben und Zahlen. Danach können Attributen Werte zugewiesen werden. Ein Beispiel für eine Literaturreferenz in einer BibTeX-Datei ist in Abbildung 2.1 zu sehen.
Das TeX-System und auch das Zusatzprogramm BibTeX trennen den Inhalt vom Layout. Der Inhalt wird übersetzt, um das Layout zu generieren. Durch das Generieren des Layouts wird die Darstellung des Inhalts möglich. Damit kann ein Dokument betrachtet und ausgedruckt werden. Mittels spezieller Dateien, der *Style-Files*, läßt sich beim Übersetzungsvorgang die Generierung des Layouts beeinflussen. Einige

Style-Files für die Generierung des Layouts von Literaturreferenzen und des Literaturverzeichnisses sind üblicherweise im BibTeX-Paket enthalten. Eine Veränderung dieser Standard-Styles ist durch jeden Benutzer möglich.

2.1.2 Bearbeitung

BibTeX-Dateien sind einfache, flache Textdateien. Dadurch ist es möglich, sie mit allen Werkzeugen zu modifizieren, die Texte in irgendeiner Weise bearbeiten können. Um die Literaturreferenzen zu bearbeiten, genügt also ein einfacher Texteditor, wie z.B. *vi* [Kathy 98], *jed* [Davis 98] oder *textedit* [Etchinson 98]. Der Benutzer muß sich an die Regeln für den Aufbau der BibTeX-Dateien halten, wenn er die Literaturreferenzen erfolgreich in einer Publikation verwenden möchte.
Die übersichtliche Formatierung der Literaturreferenzen, die Sortierung nach bestimmten Kriterien, die Wahl des Literaturreferenztyps und der dazugehörenden Attribute bleibt ganz dem Benutzer überlassen. Der Texteditor unterstützt den Benutzer mit den eingebauten Funktionen zum Suchen und Ersetzen von Texten sowie dem für Texte typischen Ablauf Markieren, Kopieren bzw. Ausschneiden und Einfügen. Je nach Texteditor und Konfiguration wird beim Speichern einer BibTeX-Datei eine bestehende Datei mit gleichem Namen unter einem leicht veränderten Namen als Sicherheitskopie gesichert.

2.1.3 Werkzeuge

Der bekannte Texteditor *XEmacs* [XEmacs Organisation 98] bietet über die Funktionalität eines Texteditors hinaus einen speziellen Textbearbeitungsmodus, den *bibtex-mode*, in dem weitere Funktionen speziell für BibTeX-Dateien zur Verfügung stehen. Die einzelnen Teile der Literaturreferenzen werden zur besseren Übersicht farbig eingefärbt und übersichtlich formatiert. Bei der Eingabe von Klammern wird zu einer schließenden Klammer die öffnende angezeigt, damit der Benutzer die Korrektheit der Eingabe prüfen kann.
Weiterhin gibt es Funktionen, um

- Schablonen zum Ausfüllen für neue Literaturreferenzen einzufügen,
- Literaturreferenzen auf syntaktische Korrektheit zu überprüfen,
- nach Literaturreferenztypen und Schlüsseln zu sortieren,
- überflüssige Klammern zu entfernen,
- Einträge aufzuräumen, d.h. übersichtlich zu formatieren und überflüssige Klammern und Leerzeichen zu entfernen,
- zwischen Einträgen hin- und herzuspringen,
- auf Rechtschreibfehler zu prüfen,
- Attribute auszublenden,
- optionale Attribute zu entfernen und
- eine Hilfe anzuzeigen.

Wie bei anderen Textdateien kann man mit den Unix-Kommandos *grep* und *sed* mehrere BibTeX-Dateien gleichzeitig durchsuchen bzw. Ersetzungen durchführen.

2.1.4 spezielle BibTeX-Tools

Da nicht jeder Benutzer mit den Möglichkeiten eines Texteditors und den Unix-Kommandos zufrieden ist und die Einhaltung der korrekten Syntax dem Benutzer überlassen bleibt, wurden Programme entwickelt, die den Benutzer beim Erstellen und Warten von BibTeX-Dateien unterstützen und das Eingeben, Ändern, Löschen, Suchen und Ersetzen von Literaturreferenzen über eine Benutzungsoberfläche ermöglichen. Programme wie *BibDB/WinBib* [Doron 98], *BibEdit* [Dale 94] und *bibview* [Weizmann 94] bieten

- Masken zur Eingabe und Ansicht der Daten,
- Springen zwischen den Literaturreferenzen und zur ersten und zur letzten Literaturreferenz,
- Auswahl einer Literaturreferenz über den Schlüssel oder über eine Übersicht mit Schlüssel, Erscheinungsjahr und Titel,
- Durchsuchen aller Attribute nach einem Suchbegriff,
- Sortieren und Formatieren der BibTeX-Datei,
- Ersatz und Bearbeiten der @STRING-Makros und
- Im- und Export von und in andere Formate für Literaturreferenzen.

2.2 Verwandte Projekte

Bei den mit der vorliegenden Arbeit verwandten Projekten lassen sich unterschiedliche Zielrichtungen unterscheiden. Eine Reihe von Projekten versuchen, möglichst große Mengen an Literaturreferenzen zu sammeln, einzuordnen und zu pflegen, um eine schnelle und erfolgreiche Recherche zu ermöglichen. Zu diesen Projekten zählt *BibNet* [Diverse 97a], eine über das Internet verteilte Datenbank, die die Literaturreferenzen im BibTeX-Format mit einigen proprietären Erweiterungen verwaltet. Beim *Hypertext Bibliography Projekt* [Jones 97] und beim *Bibliography Server* [Ley 97] werden die Bibliographien jeweils in einer Datenbank verwaltet, auf die per *World-Wide Web Browser* zugegriffen werden kann.
Der *APA Style of Citation*, beschrieben in Abschnitt 2.2.3, versucht, den neuen Anforderungen, die durch das Internet an die Verwaltung von Literaturreferenzen gestellt werden, durch entsprechende Erweiterungen des BibTeX-Formats und das Aufstellen von Konventionen für die Erstellung und Pflege von Literaturreferenzen gerecht zu werden.
Weit über das Erfassen von Literatureferenzen hinaus geht das *MeDoc*-Projekt, das in Abschnitt 2.2.1 beschrieben wird. Es hat zum Ziel, eine so große Menge von Informatikliteratur an jedem Arbeitsplatz in Deutschland zur Verfügung zu stellen, daß der Bestand für die wissenschaftliche Recherche ausreicht. Die Verwaltung geeigneter Literaturreferenzen zur Recherche ist dabei nur eine Teilaufgabe innerhalb des Projektes.
Eine größere Anzahl von Arbeiten und Programmen befaßt sich mit dem Konvertieren von unterschiedlichen Bibliographie-Formaten in andere. Stellvertretend für diese Arbeiten wird das *Canonical Bibliography Format* in Abschnitt 2.2.2 vergestellt, das ganz pragmatisch versucht, über ein kanonisches Format die Konvertierung zu vereinfachen.
Andere Projekte wie *Collection of Computer Science Bibliographies* [Diverse 97b] sammeln und warten eine Liste von Verweisen auf Bibliographiebestände. In diesem

Fall wird eine Liste von Verweisen auf Informatik-Bibliographiebestände in Nordamerika verwaltet. Per *World-Wide Web Browser* kann ein Bestand aus der Liste ausgewählt und dann direkt auf den Bestand zugriffen werden.

2.2.1 MeDoc

Im Rahmes des MeDoc-Projektes [Breu, Brüggemann-Klein 96] wird ein elektronischer Publikations- und Nachweisdienst für Informatik aufgebaut. Das Ziel ist die Bereitstellung einer kritischen Masse von Informatikliteratur, die elektronisch vom Arbeitsplatz aus recherchiert und abgerufen werden kann.
Durch das Aufkommen der vielfältigen Kommunikationsmöglichkeiten des Internet sind neue Formen des Publizierens machbar geworden. Das MeDoc-Projekt soll eine so weit wie möglich harmonisierte Basis schaffen, damit weitere Projekte im Rahmen der Weiterentwicklung und der teilweisen Neugestaltung des wissenschaftlich-technischen Informationswesens darauf aufbauen können. Am Beispiel von Informatik-Fachliteratur und der technischen Basis des Internet soll prototypenhaft für den Umbauprozeß der wissenschaftlichen Publikationslandschaft die erste digitale, online zugreifbare Volltext-Bibliothek Deutschlands erstellt werden.

2.2.2 Canonical Bibliography Format

Das Canonical Bibliography Format von Dana Jacobsen [Jacobsen 95] definiert ein kanonisches Format für Literaturreferenzen und eine Abbildung für einige gängige Formate von Literaturreferenzen auf das kanonische Format. Der Hauptaugenmerk liegt auf dem Konvertieren von unterschiedlichen Formaten für Literaturreferenzen. Durch das kanonische Format muß ein Programmautor, der ein eigenes Format in ein anderes, bereits unterstütztes konvertieren möchte, lediglich mit dem eigenen Format und dem kanonischen Format sowie der Abbildung zwischen den beiden vertraut sein. Um von einem beliebigen Format in beliebiges anderes zu konvertieren, wird immer zunächst in das kanonische Format und von dort weiter abgebildet. Dies setzt voraus, daß das kanonische Format eine Obermenge aller zu konvertierenden Formate darstellt.

2.2.3 APA Style of Citation

Der APA Style of Citation [Crane, Li 96] enthält Vorlagen für Literaturreferenzen und beschreibt Konventionen, die das Layout von Literaturreferenzen in wissenschaftlichen Arbeiten betreffen. Für die von BibTeX unterstützten Typen von Literaturreferenzen wird vorgeschlagen, wie *Univeral Ressource Locator* (URL), die im World-Wide Web benutzten Adressen, in Literaturreferenzen angegeben werden können. Der Schwerpunkt liegt auf Literaturreferenzen aus elektronischen Quellen. Es wird beschrieben, wie Zeitungsartikel, Nachrichten in Diskussionslisten und Diskussionsforen sowie persönliche Nachrichten (EMail) zitiert und in BibTeX-Dateien abgelegt werden können.

Kapitel 3

Anforderungen an die werkzeugunterstützte, kooperative Arbeit mit Literaturreferenzen

Eine Arbeitsumgebung für die kooperative Arbeit mit Literaturreferenzen soll dem Benutzer eine über die Möglichkeiten der im vorigen Kapitel vorgestellten, traditionellen Arbeit mit BibTeX-Dateien hinausgehende Funktionalität bieten. Deshalb ist entscheidend, daß die unterschiedlichen Arbeitsweisen und Anforderungen der Benutzer im Mittelpunkt stehen. Diese herauszuarbeiten ist eine der beiden Aufgaben dieser Arbeit. In die Arbeitsszenarien hinein spielt der Aspekt der Kooperation der Benutzer, der zu Arbeitsszenarien und damit Anforderungen führt, die bei autonomer, nicht kooperativer Arbeit nicht auftreten.
In diesem Kapitel wird ein Katalog von Anforderungen an eine Arbeitsumgebung für die kooperative Arbeit mit Literaturreferenzen aufgestellt. Diese Anforderungen bilden die Grundlage für Entwurf einer Arbeitsumgebung und motivieren sich aus typischen Arbeitsszenarien bei der kooperativen Arbeit mit Literaturreferenzen. Neben den Arbeitsszenarien führt die Betrachtung von Benutzergruppen und Arbeitsmodi zu einer weiteren Präzisierung der Anforderungen.

3.1 Arbeitsszenarien

Die Arbeitsszenarien für die kooperative Arbeit mit Literaturreferenzen beschreiben typische Vorgänge während des Umgangs mit den Literaturreferenzen. Die Arbeitsweise bei der kooperativen Arbeit ist ähnlich zur Einzelarbeit, was den Aufbau, die Pflege und die Recherche in einem Bestand angeht. Durch die Kooperation mit anderen Benutzern ist die Anzahl der Arbeitsszenarien jedoch umfangreicher: Die Zusammenführung und der Abgleich separater Bestände kommt bei der Einzelarbeit nur in Form einer Akquisition neuen Materials vor. Die Recherche bei mehreren Benutzern muß den weiteren Möglichkeiten, die durch mehrere Benutzer entstehen, Rechnung tragen.

3.1.1 Aufbau eines eigenen Bestandes

Bei der Beschäftigung mit einer Aufgabenstellung, der Arbeit in Projekten, allgemein, wenn es darum geht, eine wissenschaftliche Publikation zu erstellen, sucht ein

Benutzer für ihn interessante Literatur. Wenn eine Fundstelle in der Literatur für die aktuelle Arbeit, aber auch für andere Arbeiten relevant ist, erstellt der Benutzer eine Literaturreferenz, die in die Reihe bestehender Literaturreferenzen eingeordnet wird. Häufig ist eine bereits existierende Literaturreferenz der Ausgangspunkt für eine neu erstellte Literaturreferenz. Möchte der Benutzer jedoch eigene Anmerkungen zu einer übernommenen Literaturreferenz verfassen, dann übernimmt er die Literaturreferenz in seinen Bestand und ergänzt seine Anmerkungen. Durch die vollständige Neuerstellung oder das Kopieren und Annotieren bestehender Literaturreferenzen beginnt den Aufbau eines eigenen Bestandes.

3.1.2 Pflege eines Bestandes

Eine Literaturreferenz, die ein Benutzer anlegt und zu einem Bestand hinzufügt, bleibt nicht unverändert. Beim ersten Anlegen stehen oftmals noch nicht alle möglichen Informationen für die Literaturreferenz zur Verfügung. So werden von den zur Verfügung stehenden zunächst die in den Augen des Benutzers wichtigen, also die hinreichend eindeutig bezeichnenden, Informationen vermerkt. Dazu gehören der oder die Autoren, der Titel und der Typ der Literaturreferenz, sofern dieser eindeutig bestimmbar ist. Weitere Informationen über den Bezug der Literatur können noch dazukommen, um die Literatur schneller wiederfinden zu können. Stammt die Literatur aus einer Bibliothek, so bietet ein Vermerk der Signatur der Bibliothek für den Benutzer die Möglichkeit, die Literatur ohne eine erneute Recherche mit dem Recherchesystem der Bibliothek schneller wiederzufinden. Gehört die Literatur einem anderen Benutzer, ist ein Vermerk darüber sinnvoll, um zu einem späteren Zeitpunkt wieder an die Literatur heranzukommen.
Neben dem schrittweisen Sammeln der Informationen für eine Literaturreferenz werden Korrekturen z.B. der Rechtschreibung vorgenommen und Änderungen der referenzierten Literatur an der Literaturreferenz nachvollzogen, z.B. wenn sich der Zustand einer Literatur von "submitted" zu "accepted" ändert.

3.1.3 Annotation eines Bestandes

Für den Benutzer gewinnt eine Literaturreferenz durch die Zuordung zu einer eigenen Publikation und die Anmerkung z.B. über die Nutzbarkeit in weiteren Publikationen an Bedeutung.
Durch weiteres Lesen, Nachdenken und Gespräche gewinnt der Benutzer einen Eindruck über die Literatur und trägt eine kurze Zusammenfassung des Inhalts oder die Kurzfassung (*Abstract*) der Literatur ein. Dadurch steigt die Aussagekraft einer Literaturreferenz für den Benutzer selbst, für andere Benutzer, aber insbesondere für Benutzer, die gemeinsam an einer Publikation arbeiten, noch weiter.
Im Laufe der Zeit kommen so Informationen durch die Pflege der Literaturreferenzen hinzu, die einen Wert für den Benutzer, der die Literaturreferenz erstellt hat, aber auch für Benutzer, die auf der Suche nach geeigneten Literaturreferenzen sind, darstellen. Dieser Wert beruht nicht allein auf den Informationen, die die Literaturreferenz ausmachen, sondern auch auf den zusätzlichen Annotationen des Benutzers und auf dem Vorteil, auf einen gepflegten Bestand zuzugreifen.

3.1.4 Recherche in einem Bestand

Die Recherche in einem Bestand umfaßt die Suche nach geeigneten Literaturreferenzen in sowohl vom Benutzer als von anderen Benutzern angelegten Literaturreferenzen. Hierbei sind zwei Arten von Recherchen zu unterscheiden. Bei der traditionellen Arbeitsweise sucht der Benutzer nach Literatur durch die teilweise oder vollständige Angabe

- des Titels,
- des oder der Autoren,
- eines Schlagwortes,
- des Erscheinungsjahres oder
- der Signatur der Bibliothek.

Wenn diese Informationen nicht bekannt oder zu ungenau sind, werden andere, weitergehende Recherchemöglichkeiten benötigt. Diesen ist gemein, daß sie über die oben angegebene Liste hinausgehende Informationen für die Literaturreferenzen benötigen, um Fragestellungen beantworten zu können,

- welche Literatur zu einem gegebenen Thema noch vorhanden ist,
- welches Standardwerke der Literatur für ein Thema sind,
- wie der Schlüssel eines Standardwerkes, dessen Inhalt der Benutzer kennt und immer wieder zitiert, lautet,
- welche Literatur ein Autor außer der dem Benutzer bekannten noch verfaßt hat,
- welche Literatur im einem Zeitraum, beispielsweise in der letzten Woche, neu hinzugekommen ist,
- welche Literatur dem Benutzer bekannte Personen in der letzten Zeit gelesen haben,
- wer einen Literaturtitel gelesen hat und Fragen beantworten könnte, und
- wie die Literatur zügig zu beschaffen ist - hilfreich ist die Information, ob eine dem Benutzer bekannte Person ein Exemplar besitzt, das kurzfristig entliehen werden kann, oder alternativ wie die Signatur der Literatur einer nahegelegenen Bibliothek lautet.

Das Interesse an den Aktivitäten und dem Wissen von dem Benutzer bekannten Personen ist bei der kooperativen Arbeit in Arbeitsgruppen von besonderer Bedeutung. Ein Vorteil entsteht dadurch, daß der Einzelne durch die Kooperation mit anderen zu einem Ergebnis kommt, das über das hinaus geht, was er alleine geschafft hätte.

3.1.5 Akquisition neuen Materials

Ein Benutzer findet durch die Benutzung einer Suchmaschine eine interessante URL im World-Wide Web und legt eine Literaturreferenz dafür an. Diese Literaturreferenz bekommt eine Annotation, die andere Benutzer darauf hinweist, daß es sich um eine interessante Fundstelle für eine gemeinsame Publikation handelt.
Literaturreferenzen, die bei der Recherche eines Benutzers in Beständen, die außerhalb der Arbeitsumgebung verfügbar sind, gefunden werden, werden dem Bestand hinzugefügt, sofern sie nicht bereits existieren. Die Angabe, welcher Benutzer die Literaturreferenz erstellt hat, informiert darüber, wie die Literaturreferenz in den Bestand gekommen ist. Auf gleiche Art und Weise können die Literaturreferenzen in den Literaturverzeichnissen, die üblicherweise am Ende von Publikationen stehen, durch die Recherche in der Arbeitsumgebung und in Bibliothekssystemen vervollständigt und in den Bestand aufgenommen werden.

3.1.6 Selektion von Literaturreferenzen

Die Selektion von Literaturreferenzen erfolgt inkrementell begleitend zur Erstellung einer Publikation. Dies kann auf unterschiedliche Arten geschehen. Beim ersten Ansatz ordnet ein Benutzer eine Literaturreferenz einer Publikation zu. Damit wird ausgedrückt, daß die gefundene Literatur im Rahmen der Erstellung der Publikation von Interesse ist. Beim zweiten Ansatz selektiert ein Benutzer für eine zu erstellende oder erstellte Publikation Literaturreferenzen und ordnet sie zu, wenn die Literatur bei der Arbeit an der Publikation von Interesse ist oder erwähnt werden könnte.
In beiden Fällen ist es unwichtig, ob die referenzierte Literatur in der Publikation zitiert wird oder nur in einem weiteren Zusammenhang mit ihr steht. Durch die Selektion gruppiert der Benutzer die Literaturreferenzen und hat eine persönliche Sicht, die die Literaturreferenzen in Kategorien trennt. Die Kategorien kann der Benutzer selbst vorgeben.

3.1.7 Weiterverarbeitung selektierter Literaturreferenzen

Literaturreferenzen werden nicht nur selektiert, um dem Benutzer eine Strukturierungsmöglichkeit für die Literaturreferenzen zu geben. Sämtliche Funktionen der Arbeitsumgebung können auf Selektionen von Literaturreferenzen durchgeführt und diese damit gezielt verarbeitet werden.
Beispielweise werden für den Übersetzungsvorgang einer LaTeX-Datei genau die Literaturreferenzen benötigt, die in einer Publikation verwendet werden. Eine Selektion, die eine Menge von Literaturreferenzen dieser Publikation zuordnet, dient dazu, die in der Publikation referenzierte Literatur in eine BibTeX-Datei zu exportieren. Diese wird dann zur Erstellung der fertigen Publikation verwandt und enthält genau die Literaturreferenzen, die in der Publikation verwendet wurden.
Umgekehrt läßt sich zu einer Publikation über die selektierten Literaturreferenzen die Literatur wieder herausfinden, die der Benutzer bei der Arbeit an der Publikation von Interesse hielt.

3.1.8 Zusammenführung separat gewachsener Bestände

Durch die übliche Vorgehensweise, daß jeder Benutzer seinen persönlichen Bestand an Literaturreferenzen aufbaut, ergeben sich Überschneidungen und Unterschiede beim Aufbau des Bestandes. Das Zusammenführen der Bestände schafft für mehrere Benutzer die Voraussetzung dafür, eine gemeinsame Menge von Literaturreferenzen zu pflegen, um gemeinsam an Publikationen zu arbeiten und vom Mehrwert einer kooperativen Arbeitsumgebung zu profitieren. Die separat gewachsenen Bestände können disjunkte bis fast identische Mengen von Literaturreferenzen enthalten. Bei der Zusammenführung der Bestände ist für eine Literaturreferenz, die in mehr als einem Bestand besteht, eine Anpassung nötig. Die Benutzer entscheiden, welche Informationen aus den zusammenzuführenden Literaturreferenzen in die gemeinsame Literaturreferenz aufgenommen werden.

3.1.9 Abgleich mit separat gewachsenen Beständen

Wenn Bestände separat aufgebaut und gepflegt werden, besteht für den Benutzer die Möglichkeit, einen eigenen Bestand mit einem separat gewachsenen abzugleichen. Interessant und für den eigenen Bestand von Vorteil sind Änderungen und Ergänzungen des separat gewachsenen Bestandes, die so in den eigenen Bestand eingepflegt werden können.
Da die Pflege von Literaturreferenzen vergleichsweise mühsam und zeitaufwendig ist, kann der Benutzer durch einen automatischen Abgleich mit einem separat gewachsenen Bestand zumindest teilweise davon entlastet werden. Diese Erleichterung

setzt natürlich voraus, daß der Benutzer ein gewisses Vertrauensverhältnis zu den Benutzern hat, die den separat gewachsenen Bestand pflegen und erweitern. Wenn ein Benutzer Probleme hat, Änderungen von anderen Benutzern am eigenen Bestand zu tolerieren, wird er besser kontrollieren, welche Änderungen vorgenommen wurden und selber entscheiden, welche Änderungen in den eigenen Bestand übernommen werden. Der Eingriff des Benutzers in den Abgleich ist in jedem Fall nötig, wenn es zu Konflikten zwischen den Änderungen im separat gewachsenen Bestand und Änderungen im eigenen Bestand kommt.
Beim Abgleich der Bestände ist für eine Literaturreferenz, die in mehr als einem Bestand besteht, eine Anpassung nötig. Die Benutzer entscheiden, welche Informationen aus den abzugleichenden Literaturreferenzen in die eigene Literaturreferenz übernommen wird.

3.1.10 Erweiterung des Formats

Um den Anforderungen neuer Entwicklungen, die noch nicht absehbar sind, zu entsprechen, können die Benutzer das Format der Literaturreferenzen erweitern. Dies wird durch eine Abstimmung über die Änderungen und/oder Erweiterungen sowie die anschließende Umsetzung erreicht. Geändert oder erweitert werden kann die Menge von Literaturreferenztypen als auch die von der Arbeitsumgebung unterstützten Attribute der jeweiligen Typen.

3.2 Benutzergruppen und Arbeitsmodi

Zur weiteren Präzisierung der Anforderungen werden neben den Arbeitsszenarien eine Reihe von zu erwartenden, typischen Benutzern in Gruppen betrachtet. Das Kriterium für eine Abgrenzung zwischen den Gruppen ist die Charakteristik der Arbeitsweisen der Benutzer. Aus den Benutzergruppen sind die für die Erstellung der Anforderungen relevanten ausgewählt. Diese sind im Folgenden kurz beschrieben:

- Die erste Benutzergruppe stellen die Anfänger, was den Aufbau eines Bestandes an Literaturreferenzen angeht, dar. Sie haben noch keine eigenen Literaturreferenzen erstellt und sind vornehmlich daran interessiert, schnell geeignete Literaturreferenzen für eine von ihnen erstellte Publikation zu finden. Dadurch, daß andere Benutzer bereits Literatur in größerem Umfang gelesen und daraus Literaturreferenzen erstellt haben, wird das Auffinden geeigneter Literaturreferenzen deutlich erleichtert. Die neuen Benutzer können von existierenden Sammlungen von Literaturreferenzen mit ähnlicher Ausrichtung natürlich nur dann profitieren, falls solche existieren und den Benutzer bekannt sind. Dann können gefundenen Literaturreferenzen übernommen und in der eigenen Publikationen verwendet werden, ohne nochmals den Vorgang von Auffinden, Erfassen und Pflegen, der in den Arbeitsszenarien detailliert beschrieben ist, durchgehen zu müssen. Bei der Erstellung einer neuen Arbeit entstehen dennoch zusätzlich einige neue Literaturreferenzen, die die Benutzer anderen Benutzern zugänglich machen, damit diese wiederum davon profitieren können.

- Die zweite Benutzergruppe ist die der längerfristig in die Arbeit mit Literaturreferenzen involvierten Benutzer, die einen eigenen Bestand von Literaturreferenzen aufgebaut und gepflegt haben. Diesen Bestand von Literaturreferenzen möchten sie anderen Benutzer zur Verfügung stellen, um gemeinsam mit den Literaturreferenzen zu arbeiten und vom Mehrwert der Arbeitsumgebung zu profitieren.

- Die Gruppe der unabhängigen, autonomen Benutzer hat einen eigenen Bestand von Literaturreferenzen aufgebaut und gepflegt und möchte diesen anderen Benutzern zur Verfügung stellen, gleichzeitig ihre Autonomie, was den Bestand angeht, wahren. Diese Benutzer sind vor allem an einem Abgleich mit den gemeinsamen Literaturreferenzen interessiert, um ihre eigenen Literaturreferenzen aktuell und konsistent zu halten und über interessante Neuerscheinungen auf dem Laufenden zu bleiben.

- Die Außenstehenden, die nichts von der Arbeitsumgebung wissen, eigene Literaturreferenzen pflegen und möglicherweise fremde Arbeitsumgebungen nutzen, sind von den obigen Gruppen völlig unabhängig. Für sie ist es nicht möglich oder sinnvoll, die Arbeitsumgebung zu wechseln und eine andere oder Teile einer anderen Arbeitsumgebung zu benutzen. Es kann wünschenswert sein, diese Literaturreferenzen in die gemeinsame Menge von Literaturreferenzen der Arbeitsumgebung einzubringen. Bei Interesse kann ein Abgleich durchgeführt werden und die Benutzer der Arbeitsumgebung können die Veränderungen der abgeglichenen Literaturreferenzen nachvollziehen.

- Die letzte Gruppe ist die der Literatur recherchierenden, die Literatur suchen und sich die Vorteile eines existierenden Bestandes an Literaturreferenzen zu Nutzen machen können. Sie sind an der Pflege und der Erweiterung des Bestandes an Literaturreferenzen nicht beteiligt.

Verbunden mit den Benutzergruppen sind die Arbeitsmodi, d.h. die Art und Weise, die Arbeit mit den Literaturreferenzen zu organisieren. Einige Arbeitsmodi, die für die Erstellung der Anforderungen von Bedeutung sind, werden im Folgenden kurz vorgestellt:

- Der Benutzer arbeitet mit der Arbeitsumgebung sowohl an einem vernetzten als auch zeitweise an einem nicht vernetzten Arbeitsplatz. Diese Unabhängigkeit vom Ort kommt der Arbeitsgewohnheit des Benutzers entgegen, der häufiger den Ort wechselt und vor Ort seine Arbeit erledigt. Entscheidet der Benutzer sich, autonom (nicht vernetzter Arbeitsplatz) zu arbeiten, exportiert er den gesamten Bestand oder einen Teilbestand von Literaturreferenzen. Danach bearbeitet er diese Kopie des Bestandes unabhängig von den anderen Benutzern. Ein Import der getrennt gepflegten Literaturreferenzen und der Abgleich mit den gemeinsamen Literaturreferenzen pflegt die Änderungen und Neuerungen in den gemeinsamen Bestand ein.

- Noch weitergehend als die Unabhängigkeit von der Vernetzung ist das autonome Arbeiten, das den Benutzer unabhängig von der Verfügbarkeit eines Systems, der Wahl eines Systems oder einer Systemumgebung oder eines Netzanschlusses macht.

- Die kooperative Arbeit mehrerer Benutzer besteht darin, auf einen gemeinsamen Bestand zuzugreifen, die Literaturreferenzen von anderen Benutzern zu pflegen, sofern diese dazu etwa im Rahmen der Arbeit an gemeinsamen Publikationen autorisiert haben. Durch die Annotation und die Eingabe von Inhaltsverzeichnissen, Zusammmenfassungen und Abstracts, die bei der kooperativen Arbeit mehrerer Benutzer einen Mehrwert für alle Mitglieder der Gruppe schafft, erhält man bei der Recherche treffendere Anfragenergebnisse und kann schneller geeignet Literatur auffinden. Das Einbringen und Abgleichen weiterer Bestände bietet den Vorteil, eine größere Menge von Literaturreferenzen zur Recherche zur Verfügung zu haben und eine bessere Ausbeute an Fundstellen vor allem bei weniger verbreiteter, sehr spezieller Literatur zu haben.

3.3 Anforderungen an eine Werkzeugunterstützung

Aus den Arbeitsszenarien, den Benutzergruppen und den Arbeitsmodi leitet sich ein Katalog von Anforderungen für eine Arbeitsumgebung zur kooperativen Arbeit mit Literaturreferenzen her. Den Hintergrund für die Anforderungen bildet der im vorigen Abschnitt beschriebene Status quo bei der Arbeit mit Literaturreferenzen. Der Katalog von Anforderungen unterteilt sich in grundsätzliche und konkrete Anforderungen.

3.3.1 Grundsätzliche Anforderungen

Die grundsätzlicheren Anforderungen, die recht abstrakt den großen Rahmen bilden, der den konkreteren Anforderungen zugrunde liegt, sind:

Qualitätssicherung Ein Bestand an Literaturreferenzen stellt durch die Arbeit und Mühe, die zu seiner Erstellung und Pflege notwendig war, einen Wert für den Benutzer dar. Durch die Wahl geeigneter Formate für Import und Export der Daten und ein Datenmodell, das etwaige Neuerungen und Änderungen berücksichtigen könnte, kann diese Investition gesichert werden. Die Anforderung nach Sicherung der Qualität stellt durch ihre Allgemeinheit ein Oberziel für die anderen Anforderungen dar.

Mehrwertbildung Die Kooperation der Benutzer steht im Mittelpunkt der Mehrwertbildung der Arbeitsumgebung. Diese sollte dem Benutzer einen Mehrwert, eine über die Möglichkeiten der traditionellen Arbeit mit BibTeX-Dateien (s. Abschnitt 2.1) hinausgehende Funktionalität. Je mehr Benutzer Gebrauch von der Arbeitsumgebung machen, desto höher fällt dieser Mehrwert aus, wenn gleichzeitig die Qualitätssicherung sichergestellt ist.

Flexibilität Bei der kooperativen Arbeit mit Bibliographien sollte eine Arbeitsumgebung keine Einschränkungen auferlegen, die unnötige Beschränkungen des Benutzers bedeuten. Die Freiheit des Benutzers, Ort und Zeit der Arbeit und seine Arbeitsweise flexibel bestimmen zu können, kommt den Gewohnheiten des Benutzers entgegen. Auf längere Sicht ist es notwendig, flexibel auf das Aufkommen neuer Anforderungen reagieren zu können und diese in die Arbeitsumgebung zu integrieren.

Personalisierung Jedem Benutzer steht eine persönliche Sicht der Literaturreferenzen zur Verfügung. Der Mehrwert der Arbeitsumgebung beruht im Wesentlichen auf der Zuordnung von Besitzern zu Literaturreferenzen sowie auf der Möglichkeit der Selektion und der Annotation innerhalb der persönlichen Sicht.

Kontinuität, Wahrung von Investitionen Eine Arbeitsumgebung für die Arbeit mit Bibliographien, die zu erstellen ist, kommt in eine Umgebung, in der große Bestände von Literaturreferenzen bestehen, die Arbeit an Publikationen eine alltägliche Sache ist, für die Werkzeuge zur Verfügung stehen und genutzt werden. Durch die Nutzung und behutsame Erweiterung der benutzten Dateiformate sowie die Option, Bestände bei Bedarf wieder aus der Arbeitsumgebung herauszuholen und im gewünschten Format abzulegen, wird der Wahrung der getätigten Investitionen Rechnung getragen.

Kompatibilität, Offenheit Die Kompatibilität zu bestehenden Werkzeugen und deren Dateiformaten sichert die Wahrung von Investitionen und ist Voraussetzung für den Einsatz einer Arbeitsumgebung. Die auf einen längeren Zeitraum

angelegte Migration mit der Perspektive, die Arbeit mit Bibliographien fast ausschließlich mit einer Arbeitsumgebung zu erledigen, erfordert eine Flexibilität dieser im Umgang mit neuen Anforderungen. Eine offene, erweiterbare Architektur erleichtert die Integration neuer Anforderungen, auch wenn diese zum Zeitpunkt der Erstellung der ersten Anforderungen noch nicht vorhersehbar waren.

Autonomie, Verfügbarkeit Die Möglichkeit, daß der Benutzer vollkommen losgelöst von zentralen Komponenten seine Arbeit erledigen kann, ist notwendig für die Akzeptanz der Benutzer. Diese Autonomie unterstützt die unterschiedlichen Arbeitsweisen und -gewohnheiten der Benutzer und führt zu einer deutlich besseren Akzeptanz des Systems bei den Benutzern. Die Verfügbarkeit zu steigern, ist wesentlich für die Wahrung der Investitionen und verbessert die Bereitschaft der Benutzer, die Arbeitsumgebung zu nutzen. Das wiederum ist eine zentrale Voraussetzung, um die Bildung eines Mehrwertes zu erreichen.

3.3.2 Konkrete Anforderungen

Die konkreteren Anforderungen, die zum Teil der Unterstützung und Realisierung der grundsätzlicheren (und abstrakteren) Anforderungen dienen, sind:

Mehrwert Die Arbeitsumgebung sollte einen Mehrwert bieten. Der Mehrwert der Arbeitsumgebung kann aus den Möglichkeiten bestehen

- Suchen über einen verteilten Bestand an Literaturreferenzen zu ermöglichen,
- Konsistenz zu wahren,
- die Qualität von eingebrachten Literaturreferenzen durch die Nutzung zusätzlicher Informationen aus der gemeinsamen Menge von Literaturreferenzen zu erhöhen,
- jeder Literaturreferenz einen Besitzer, der ihn in die gemeinsame Menge von Literaturreferenzen eingebracht hat, zuzuordnen,
- das Sammeln zusätzlicher Informationen wie persönlicher Annotationen, Besitzer, Aufnahmedatum, Änderungsdatum und anderer Informationen, die nicht in BibTeX-Dateien abgelegt werden, zu ermöglichen,
- jedem Benutzer eine eigene, persönliche Sicht auf die von ihm gestellten Suchanfragen, Annotationen und selektierten Literaturreferenzen zur Verfügung zu stellen,
- den Besitzer einer Literaturreferenz auf einem Kommunikationsweg zu erreichen, um auf der Suche nach geeigneter Literatur Ansprechpartner zu finden oder um über Änderungen und Neuerungen zu informieren und
- ein Modell zur Einsicht und Bearbeitung festzulegen, das dem Arbeiten in der jeweiligen Arbeitsgruppe angemessen ist.

Genau definierte Begriffe Die Benutzer sollen ein möglichst einheitliches Verständnis der in der Arbeitsumgebung benutzten Begriffe haben, damit sie bei der Arbeit mit dem Literaturreferenzen im Idealfall zu identischen Entscheidungen kommen, wenn es um die Einordnung von Literaturreferenzen in Kategorien und das Füllen der Attribute der Literaturreferenzen mit Inhalt geht. Deshalb ist eine genaue und intuitive Begriffsdefinition für die Literaturreferenztypen und deren Attribute notwendig.

Zitierregeln Durch das Vorschlagen und Befolgen von Zitierregeln ist eine geeignete Präsentation der Literaturreferenzen, die aus dem Bestand ausgewählt wurden, unabhängig von dem Benutzer, der die Literaturreferenzen eingetragen hat.

Eingabehilfen Diese bieten Erklärungen und Beispiele zur Auswahl eines geeigneten Literaturreferenztyps bei der Neuanlage von Referenzen, aber auch bei der Eingabe und Pflege der Attribute. Beispiele erleichtern die einheitliche Eingabe von Daten, die Benutzer sonst je nach Interpretation und persönlicher Vorliebe vornehmen. Mit Vorgaben für Eingaben, wo dies sinnvoll ist, und einer Möglichkeit zur Auswahl aus den häufigsten Eingaben wird eine höhere Konsistenz der Eingaben erzielt.

Eingabeunterstützung Die Überprüfung auf korrekte Rechtschreibung, Duplikate und ähnlich klingende Worte vermeidet Rechtschreibfehler und doppelte Einträge. Durch die Markierung optionaler und obligatorischer Attribute sieht der Benutzer die relevanten Informationen einer Literaturreferenz auf einen Blick. Mit einer fehler- und schreibweisetoleranten Suche kann eine größere Anzahl geeigneter Fundstellen ausgegeben werden. Die Vorgabe von Schlüsselwörtern zu Publikationen erleichtert bei der Eingabe die Zuordnung von Schlüsselwörtern zu Publikationen und erhöht die Qualität der Fundstellen bei der Recherche.

Generierung von BibTeX-Schlüsseln Die Schlüssel für das Zitieren von Literaturreferenzen werden nach einem leicht verständlichem Schema von der Arbeitsumgebung generiert. Damit entfällt die Suche nach den Literaturreferenzen in der Arbeitsumgebung während des Schreibens einer Publikation, sofern der Benutzer die Namen der Autoren noch erinnert. Alle Literaturreferenzen bekommen einen neuen, eindeutigen Schlüssel innerhalb der Arbeitsumgebung. Um in bereits vor Einsatz einer Arbeitsumgebung verfaßten Arbeiten keine Schlüssel ändern zu müssen, besteht die Möglichkeit, eine Literaturreferenz mit einem zweiten Schlüssel zu versehen, der beim Import oder Abgleich von Literaturreferenzen belegt, wird und diesen zweiten Schlüssel zu benutzen. Eine Literaturreferenz kann dann unter dem alten oder dem neuen Schlüssel angesprochen werden. Der Benutzer muß sich für eine Form entscheiden, wenn er Literaturreferenzen selektiert und exportiert.

Im- und Export Die Möglichkeit eines Im- und Exports folgt aus den grundsätzlichen Anforderungen nach Qualitätssicherung und Kontinuität. Diese Anforderungen ergeben sich dabei aus den Arbeitsszenarien über die Zusammenführung und den Abgleich separat gewachsener Bestände. Konkret bedeutet dies, daß eine Möglichkeit vorhanden sein muß, Literaturreferenzen in die Arbeitsumgebung hinein- und auch wieder hinauszubringen. Die Kompatibilität zu bestehenden Dateiformaten für Literaturreferenzen gehört zu den Anforderungen für den Im- und Export.

Personalisierung Nach der Authentisierung findet der Benutzer die Arbeitsumgebung beim nächsten Start wieder so vor, wie er sie verlassen haben. Einstellungen, die er getätigt hat, Selektionen, die er definiert hat, stehen ihm wieder zur Verfügung. Zwischenzeitliche Veränderungen an Literaturreferenzen, die ein Benutzer selektiert hat, werden kenntlich gemacht.

Umgebungsunabhängigkeit Aus den grundsätzlichen Anforderungen der Flexibilität und Verfügbarkeit an eine Arbeitsumgebung, ergibt sich, daß die Arbeit mit der Arbeitsumgebung unabhängig von Rechnern und Rechnerplattformen sein soll. Die Arbeit kann vor Ort am Arbeitsplatz erledigt werden. Existiert

kein Netzanschluß, so kommt eine autonome Variante der Arbeitsumgebung zum Einsatz, die bei der Arbeit keine Einbußen in der Funktionalität zur Folge hat. Ein Abgleich der autonom bearbeiteten Literaturreferenzen sorgt dafür, daß die autonome Variante aktuell bleibt.

3.4 Zugriffsrechte und Rollen

Eine Literaturreferenz gehört immer einem Benutzer. Er kann, wenn er mit anderen Benutzern eine gemeinsame Arbeit erstellt, eine Literaturreferenz einem anderen Benutzer überlassen. Dieser ist dann Besitzer der Literaturreferenz.
Es ist denkbar, daß einige Benutzer ihre gemeinsame Menge an Literaturreferenzen Außenstehenden zum Suchen zur Verfügung stellen möchten. In diesem Fall ist sichergestellt, daß kein Außenstehender in der Lage ist, Literaturreferenzen zu verändern. Auch ist die Suche nach Annotationen, die Benutzer erstellt haben, unterbunden, um nicht die Kontrolle über eventuell persönliche Daten abzugeben.
Für unterschiedliche Arbeitsgruppen und Aufgabenbereiche sind folgende Möglichkeiten, mit dem Zugriff auf Literaturreferenzen umzugehen, denkbar:

restriktive Zugriffspolitik Ein Benutzer sucht sich nach dem Lesen einer Literatur die entsprechende Literaturreferenz heraus und fügt einen Kommentar an. Sollte er der Besitzer der Literaturreferenz sein, kann er weitere Informationen ergänzen oder ändern. Andernfalls nutzt er die Möglichkeit, dem Besitzer der Literaturreferenz eine Nachricht zukommen zu lassen. Nur auf diesem Wege können Änderungen stattfinden.

Ein Benutzer kann anderen Benutzern und Außenstehenden den Zugriff auf eigene Literaturreferenzen verwehren. Annotationen zu Literaturreferenzen können nur von ihrem Besitzer eingesehen werden.

kooperative Zugriffspolitik Zur Unterstützung bei der Arbeit von mehreren Benutzern an einer gemeinsamen Arbeit kann ein Benutzer die Zugriffsrechte für Literaturreferenzen auf mehrere Benutzer zu erweitern. Damit sind alle als Besitzer eingetragenen Benutzer gleichberechtigt. Sie können beliebige Änderungen an den Literaturreferenzen vornehmen. Dem ursprünglichen Besitzer der Literaturreferenz kann jedoch der Zugriff nicht entzogen werden. Ändert ein Benutzer eine Literaturreferenz, werden die weiteren als Besitzer eingetragenen Benutzer davon mit einer Nachricht informiert.

Jeder Benutzer stellt seine Literaturreferenzen allen anderen uneingeschränkt zur Ansicht zur Verfügung. Für Annotationen kann entschieden werden, ob sie anderen Benutzern anzeigt werden sollen.

freizügige Zugriffspolitik Alle Benutzer bringen einander so viel Vertrauen entgegen, daß sie alle eigenen Literaturreferenzen und Annotationen den anderen ohne Einschränkungen zur Verfügung stellen. Es ist möglich, beliebige Änderungen an den Literaturreferenzen vorzunehmen. Eine Nachricht informiert die Benutzer, die eine Literaturreferenz selektiert haben, über Änderungen. Bei dieser Zugriffspolitik hängt die Qualitätssicherung stark von der Disziplin aller Beteiligten ab.

Kapitel 4

Verwendete Komponenten, Werkzeuge und Sprachen

Für die Modellierung der Daten, Prozesse und Datenflüsse stehen neben den älteren, am imperativen oder funktionalen Paradigma orientierten Softwareentwurfsmethoden eine Reihe von objektorientierten Methoden zur Verfügung. In diesem Kapitel wird die *Object Modeling Technique* (OMT), mit der die Modellierung in dieser Arbeit vorgenommen wird, vorgestellt.
Die Modellierung läßt sich für kleinere Projekte mit wenigen Entwicklern und überschaubarem Zeitrahmen noch mit Papier und Bleistift durchführen. Für größere Projekte, an denen Dutzende von Entwicklern monatelang arbeiten, geht kein Weg an einem Werkzeug, das den Einsatz der Softwareentwurfsmethode unterstützt, vorbei. Die Methoden zum Softwareentwurf sollen die Komplexität der modellierten Domäne beherrschbar machen, doch erst mit der Werkzeugunterstützung stehen geeignete Mittel und Wege bereit, größere Projekte zu realisieren. Die in der vorliegenden Arbeit verwendete Softwareentwicklungsumgebung *Software through Pictures* (StP) wird beschrieben.
Die Implementation des Prototypen erfolgt mit der objektorientierten Programmiersprache *Java*, die in Zuge des Internet-Booms eine rasante Entwicklung zeigt. Neben einen kurzen Einführung in die Programmiersprache und ihre besonderen Eigenschaften wird die *Client-Server-Programmierung* mit Java vorgestellt. Aufgrund seiner Portabilität stellt Java eine Programmiersprache und -umgebung dar, die sich für den Einsatz in heterogenen Umgebungen besonders eignet. Die Methoden zur Datenbankanbindung von Java-Applikationen - beginnend mit der Oberfläche über die Anbindung bis zu den verbreiten, relationalen Datenbanken - werden vorgestellt. Den Abschluß dieses Kapitels bildet die Betrachtung der Sicherheitsaspekte in heterogenen, offenen und damit potentiell unsicheren Netzen.

4.1 Methoden und Werkzeuge zur Modellierung

Die Analyse und die Modellierung von Datenstrukturen, Prozessen als dynamischen Abläufen, die den Zustand der Daten verändern sowie Datenflüssen, die durch Prozesse auf den Daten ausgelöst werden, ist ein komplexer Vorgang, der die Möglichkeiten des Menschen schnell übersteigt. Die Softwaretechnik hat zur Beherrschung der Komplexität des Entwurfsprozesses formale Methoden entwickelt. Der Vorteil dieser Methoden liegt darin, daß durch die Formalisierung der Entwurfsvorgang besser nachvollziehbar wird und die Kommunikation unter denen, die die Methode beherrschen, durch eine einheitliche Begriffsbildung und Vorgehensweise bei der Anwendung der Methode erleichtet wird.

Für die Modellierung wird die Object Modeling Technique verwendet. Für die Modellierung wird *Software through Pictures* verwandt, ein Werkzeug, das einen rechnergestützten Entwurfsprozeß unterstützt.

4.1.1 OMT

Die *Object Modeling Technique* (OMT) [Rumbaugh et al. 91] ist historisch gesehen eine der ersten aus einer ganzen Reihe von Methoden zur objektorientierten Entwicklung von Software. Es handelt sich hierbei nicht um eine Programmiertechnik, sondern ein Modellierungskonzept. Dieses entwickelt ein Modell der Anwendungsdomäne und fügt während des Entwurfs Details der Implementierung hinzu. Die Methode umfaßt vier Phasen:

Analyse Ausgehend von einer informalen Problembeschreibung wird in der *Analyse* ein Bereich der realen Welt möglichst kurz, präzise, verständlich und korrekt modelliert. Die Analyse dient dem Verständnis des Problems und stellt fest, was getan werden muß, ohne einzuschränken, wie dies zu tun ist. Als Vorbereitung für den Entwurf wird festgelegt, welche Funktionalität die zu erstellende Software erfüllen soll, ohne irgendwelche Implementationsentscheidungen zu fällen.

Systementwurf Die zu erstellende Software wird im Rahmen der Erstellung einer Systemarchitektur in ihre wichtigsten Komponenten unterteilt. Beim *Systementwurf* werden die grundlegenden konzeptionellen und entwurfspolitischen Entscheidungen gefällt, die den Rahmen für die Detaillierung des Entwurfs im Rahmen des als nächste Phase folgenden Objektentwurfs bilden.

Objektentwurf Der *Objektentwurf* erweitert das Analysemodell um das für die Implementation Nötige und definiert Klassen, Beziehungen zwischen Klassen, Schnittstellen und Algorithmen in vollem Umfang. Beim Objektentwurf werden interne Objekte, die für die Implementation benötigt werden, hinzugefügt und Datenstrukturen und Algorithmen optimiert.

Implementierung Die während des Objektentwurfes entwickelten Klassen und Algorithmen werden während der *Implementierung* realisiert mit den Konzepten von objektorientierten oder nicht-objektorientierten Programmiersprachen und Datenbankmanagementsystemen. Die Programmierung sollte die simple und gerade, damit fast mechanische Übersetzung der Entwurfsentscheidungen in die Eigenheiten einer bestimmten Programmiersprache sein, da alle wichtigen Entscheidungen während des Entwurfs getroffen worden sein sollten.

Die OMT-Methode definiert eine graphische Notation, um die Erstellung kohärenter, präziser und leicht zu formulierender und zu verstehender Modelle zu erreichen. Während des Entwurfs werden in der Notation der OMT-Methode drei Modelle erstellt:

Objektmodell Das *Objektmodell* erfaßt die statische Struktur eines Systems, indem es die Objekte in dem System, die Beziehungen zwischen den Objekten und die Attribute und Operationen, die jede Klasse von Objekten beschreiben, zeigt. Es enthält Objektdiagramme. Ein Objektdiagramm ist ein Graph, dessen Knoten Klassen und dessen Kanten Beziehungen zwischen Klassen sind.

Dynamisches Modell Das *dynamische Modell* beschreibt Veränderungen des Zustandes des Systems über die Zeit. Es enthält Zustandsdiagramme. Ein Zustandsdiagramm ist ein Graph, in dem die Knoten Objektzustände und die Kanten der von Ereignissen ausgelöste Übergang zwischen Objektzuständen sind.

Funktionales Modell Das *funktionale Modell* beschreibt die Transformation von Daten durch Prozesse. Es enthält Datenflußdiagramme. Ein Datenflußdiagramm ist ein Graph, in dem die Knoten Prozesse oder Datenspeicher und die Kanten Kontroll- oder Datenflüsse sind.

Das Objektmodell ist das wichtigste der drei Modelle und die Basis für die anderen beiden, da es definiert, *was* sich ändert. Erst damit kann mit dem dynamischen Modell beschrieben werden, *wann* sich etwas ändert und mit dem funktionalen Modell, *wie* es sich ändert.

4.1.2 StP/OMT

Software through Pictures (StP) ist eine allgemeine Softwareentwicklungsumgebung, die mit der Erweiterung *StP/OMT* [Int 94] die Softwareentwicklung mit der objektorientierten Methode OMT erlaubt. Unter den rechnergestützten Werkzeugen zählt es zur Kategorie der *Computer Aided Software Engineering* (CASE) Werkzeuge.
Die Werkzeuge von StP/OMT unterstützen die Analyse, den System- und Objektentwurf und die Implementierung mit *C++*, *Ada*, *Smalltalk* und *SQL*. Sie bestehen jeweils aus einer Palette graphischer Editoren, Tabelleneditoren und Editoren für textuelle Annotationen. Diese werden in den Phase der Analyse und des System- und Objektentwurfs genutzt. In der Implementierungsphase generieren Werkzeuge automatisch Programmkode, der mit den Ergebnissen der Analyse- und Entwurfsphase kommentiert wird.
Die graphischen Editoren bieten die Möglichkeit, die vordefinierte Symbole der OMT-Methode mit Hilfe der Maus auf einer Arbeitsfläche zu plazieren. Die Symbole können miteinander verknüpft und beschriftet werden, ihre Anordung läßt sich auf eine korrekte Syntax testen. Auf diese Art und Weise werden im Rahmen der Modellierungschritte Systementwurf und Objektentwurf Diagramme für das Objektmodell, das dynamische Modell und das funktionale Modell erstellt.
Mittels der Tabelleneditoren für Klassen und Zustände werden die Diagramme, die mit den graphischen Editoren erstellt wurden, genauer spezifiziert. Dazu können vordefinierte Eigenschaften der auf der Arbeitsfläche plazierten Symbole textuell beschrieben werden.
Die Editoren zur textuellen Annotation dienen ebenso wie die Tabelleneditoren zur Erfassung von Texten. Mit ihnen können im Rahmen der Analyse mit einem größeren Focus die Diagramme und die durch die Tabelleneditoren beschriebenen Eigenschaften ohne die Einschränkung auf vordefinierte Attribute in ihrer Bedeutung festgelegt werden.

4.2 Komponenten und Sprachen zur Implementierung

Die Implementation soll mit Komponenten und einer Programmiersprache durchgeführt werden, die für den Einsatz in heterogenen, verteilten Umgebungen als Client-Server-Lösung besonders geeignet sind. Dies ist eine Folgerung aus dem Katalog der grundsätzlichen Anforderungen.
Mit Java wird eine relativ neue Programmiersprache gewählt, die durch den Boom des World-Wide Web und die Unterstützung der Client-Server-Programmierung in einer heterogenen, verteilten Umgebung sehr rasch an Bedeutung gewinnt.

4.2.1 Java

Die Programmiersprache Java [Sun Microsystems Inc. 96b] wurde laut Sun Microsystems entworfen, um „eine Reihe von Probleme in der modernen Programmierung zu lösen und um eine Programmiersprache für das Internet zu schaffen". Abbildung 4.1 enthält eine Reihe von Schlagworten, mit denen die Entwickler von Java ihre Sprache kennzeichnen. Java entstand im Rahmen eines größeren Projektes, das Software für den Einsatz in integrierten Systemen entwickeln sollte. Die betrachteten Geräte im Bereich der „Consumer Electronics" (elektronische Endkundengeräte) sollten kleine, zuverlässige, portable, verteilte, echtzeit-fähige integrierte Systeme sein.

> Java: A simple, object-oriented, network-savvy, interpreted, robust, secure, architecture neutral, portable, high-performance, multi-threaded, dynamic language.

Abbildung 4.1: Zitat des Java-Entwicklers James Gosling

Um diesen Anforderungen zu genügen, bekam Java eine einfache und bekannte, da an die verbreiteten Programmiersprachen C und C++ angelehnte Syntax, die den Umstieg für die meisten Entwickler erleichtern soll. Einige Eigenschaften und Besonderheiten, die C und C++ bieten, wurden entfernt, um Java einfach und frei von Redundanz zu halten. Ein weiteres Ziel dabei war auch die Verbesserung der Ablaufsicherheit der Programme (z.B. *Pointer*). Die C-Konzepte Präprozessor und Pointer wurden entfernt. Die Mehrfachvererbung auf der Ebene der Implementierung, die C++ bietet, wurde aufgrund ihrer schwierig zu handhabenden Semantik entfernt und durch die Mehrfachvererbung auf der Ebene der Interfaces ersetzt. Das Überladen von Operatoren (Overloading) wird ebenso wie die automatische Konvertierung von Typen (Coercion) nicht unterstützt. Die Robustheit wird durch intensive Tests auf Fehler zur Kompilierzeit und eine zweite Stufe von Tests zur Laufzeit erreicht. Durch den objektorientierten Ansatz entsteht der Vorteil, moderne Softwareentwicklungsmethoden wie z.B. OMT mit ihrer größeren Nähe zur realen Welt zu nutzen und dies ohne den konzeptuellen Bruch zwischen den Phasen Entwurf und Implementation. Mit der Integration von Threads in Java wird ein Betriebssystemkonzept unterstützt, das immer weitere Verbreitung findet, da es Skalierbarkeit beim Einsatz mehrer Prozessoren verspricht.
Ausgelöst durch das massive Wachstum des Internet und des World-Wide Web wurde Java aufgrund seiner Eigenschaften für die Entwicklung sicherer, portabler und robuster Anwendung im Rahmen von heterogenen, verteilten Netzen mit vielen unterschiedlichen Systemplattformen bekannt. Durch den Betrieb auf vielen Plattformen in heterogenen Netzwerken erscheint das traditionelle Schema zur Entwicklung und Verteilung von Software, die Verteilung von binärem Maschinensprachen-Kode für eine Zielplattform, die Definition von Releases und die anschließenden Pflege durch Upgrades und Patches nicht mehr opportun.
Java basiert als architektur-neutrale, portable Programmiersprache auf einer Laufzeitumgebung, die aus der *Java Virtual Machine* (JVM), einem Interpreter für den plattformunabhängigen *Java-Bytecode*, und einer in Java geschriebenen Klassenbibliothek, dem Java *Application Programming Interface* (API), die auf jeder Plattform dieselbe Schnittstelle zur Entwicklung von Anwendungen bereitstellt, besteht. Die Speicherfreigabe geschieht automatisch per *Garbage Collector*. Der Vorteil der Plattformunabhängigkeit steht dem Nachteil mangelnder Effizienz gegenüber, der durch die aufwendige Interpretation des plattformunabhängigen Bytecodes durch eine plattformabhängige virtuelle Maschine entsteht.
Damit ist die Basis vorhanden, um die Software, die die Client-Rechner benötigen, in Form von *Applets* bei jedem Programmstart über das Netzwerk von der

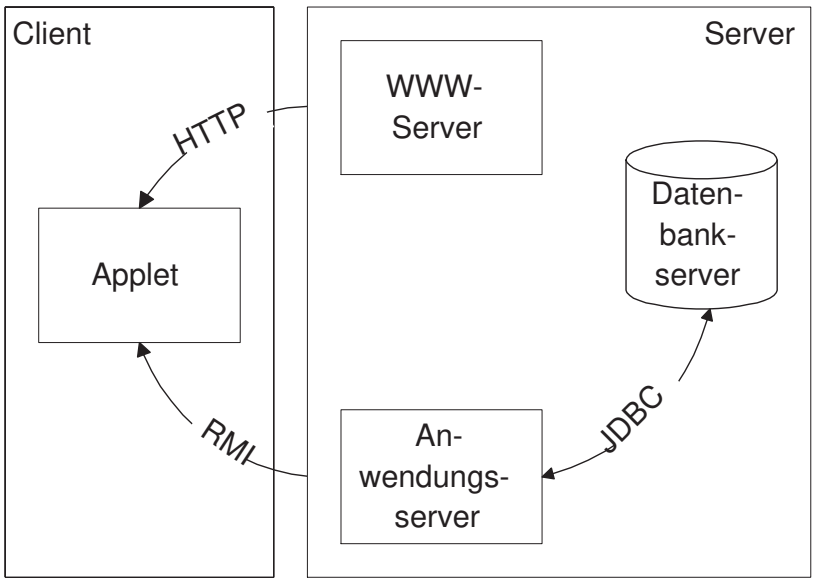


Abbildung 4.2: Client-Server-Modell

Server-Rechnern herunterzuladen. Dieser Aufbau verbindet die Vorteile an Flexibilität einer verteilten Umgebung mit den organisatorischen Vorteilen einer zentralen Administration. Es kann z.B. gewährleistet werden, daß alle Client-Rechner stets denselben Versionstand der Software benutzen; neue Software steht sofort allen Client-Rechern zur Verfügung. Jedoch entfällt die Installation und Konfiguration der Client-Rechner.
Dank all dieser Eigenschaften bietet Java für die Anwendung als Programmiersprache für verteilte Anwendungen in heterogenen Netzen gute Voraussetzungen.

4.2.2 Client-Server Programmierung mit Java

Für die Erstellung von Client-Server-Anwendungen bietet sich die Integration von Java-Bytecode, den *Applets* und *Hypertext Markup Language* (HTML) Seiten im World-Wide Web an. Da HTML ein offener Standard ist und ein Großteil der Benutzer über einen java-fähigen World-Wide Web Browser verfügt, ergibt sich eine weithin verfügbare, offene und plattformunabhängige Basis für verteilte Anwendungen. Die grundlegende Architektur einer solchen verteilten Anwendung zeigt Abbildung 4.2.
Die Anwendung, d.h. der Teil, der als Applet auf dem Client-Rechner abläuft, gelangt über die Einbettung in eine HTML-Seite von dem Web-Server zum Client-Rechner. Dieser übernimmt als sog. *Thin-Client* ausschließlich die Präsentation der Anwendung und besitzt lediglich die Präsentationslogik der Anwendung. Die Programmobjekte des Client-Teils der Anwendung werden über entfernte Referenzen auf den Anwendungs-Server verteilt. Der Prozeß der Anwendung mit der Anwendungslogik läuft auf dem Anwendungs-Server, der mit dem Client über ein Protokoll zum entfernten Methodenaufruf (RMI, s. Abschnitt 4.3.1) kommuniziert. Der Anwendungs-Server wiederum nutzt einen Datenbank-Server, um die Persistenz der Daten zu erreichen. Beide kommunizieren über ein Datenbankprotokoll (JDBC, s. Abschnitt 4.3.3) und die relationale Anfrage- und Manipulationssprache SQL.
Die vorgeschlagene Trennung von Präsentationslogik, Anwendungslogik und Persistenzabstraktion ist Charakteristikum des sog. *Three-Tier-Model*. Es trennt Präsen-

tation, Prozeß und Daten [Orfali et al. 96]. Der Nachteil dieser Lösung ist die gesteigerte Komplexität und der erhöhte Kommunikationsaufwand gegenüber Varianten, die Applikations- und Datenbank-Server oder alle drei Komponenten vereinten. Es gibt jedoch eine Anzahl von Vorteilen:

- Die Aufgaben sind klar getrennt: Der Client präsentiert die Anwendung, er stellt die graphische Oberfläche dar und enthält den Programmkode, um die Oberfläche zu erstellen und für den Benutzer bedienbar zu machen.
- Der Anwendungs-Server beinhaltet die Anwendungslogik und damit den Prozeß der Anwendung. Er enthält den Programmkode, um den Prozeß zu steuern und die Datenbank anzusprechen. Zur Laufzeit existieren die Programmobjekte auf dem Anwendungs-Server.
- Der Datenbank-Server sorgt ausschließlich für die Persistenz der Daten der Anwendung.
- Aus der Aufgabentrennung läßt sich eine Lastverteilung leichter erreichen. Die Skalierung und Austauschbarkeit der Komponenten ist durch die klaren Schnittstellen in einer feineren Granularität möglich. Es ist einfach, neue Clients einzubeziehen, die Server an die zu erwartende Laste anzupassen, zu ersetzen oder an einem anderen Ort aufzustellen. Der Datenbank-Server unterliegt keiner Einschränkung, außer daß er für den Anwendungs-Server erreichbar sein muß.
- Es ist nicht nötig, auf der Clientseite außer der Basistechnik (in Form eines WWW-Browsers mit Java-Unterstützung) weitere Komponenten zu installieren. Die Wartung und Administration vereinfacht sich, da sie stärker zentralisiert werden kann.
- Die Pflege aller Komponenten bleibt übersichtlich. Die Installation neuer Versionen auf dem Client geschieht einfach durch den Austausch des Java-Bytecodes durch eine neue Version. Damit nutzen alle Clients stets die aktuelle Version. Diese Homogenität der Komponenten erleichtert ebenfalls Wartung und Administration.

Bei dieser Auflistung sind Betrachtungen der Qualität der verwendeten Kommunikationsinfrastruktur außen vor gelassen, da diese als eine erforderliche Ressource angenommen wird, die in geeigneter Qualität zur Verfügung steht.

4.3 Methoden zur Erstellung verteilter, persistenter Anwendungen

Um die dauerhaft in einer Datenbank gespeicherten Daten auf der Benutzeroberfläche eines auf einem entfernten Rechner ablaufenden Programms darzustellen und zu bearbeiten, sind Komponenten notwendig, die diese Funktionalität erlauben. Die Methoden zur Erstellung verteilter, persistenter Anwendungen nach dem Client-Server-Modell (s. Abbildung 4.2) in Java unterteilen sich in die Verteilung von Objekten mit *RMI*, die Erstellung plattformunabhängiger Oberflächen mit *AWT*, die Datenbankanbindung mit *JDBC* und relationale Datenbanken zur dauerhaften Speicherung der Daten. Diese Komponenten werden im Folgenden vorgestellt.

4.3.1 Verteilte Objekte mit RMI

Für die Trennung von Präsentations- und Anwendungslogik stellt der entfernte Aufruf von Prozeduren, Funktionen oder Methoden die Basis dar. Seit der Version 1.1

[Sun Microsystems Inc. 97a] des *Java Developer Kit* (JDK) steht mit *Remote Method Invocation* (RMI) [Sun Microsystems Inc. 96d] ein Java API zur Verfügung, das die Verteilung von Java-Objekten erlaubt.
RMI entspricht in seiner Grundfunktion den bekannten *Remote Procedure Call* (RPC) Mechanismen, die den Aufruf entfernter Prozeduren ermöglichen. Der Unterschied besteht darin, daß nicht eine beliebige Prozedur auf dem entfernten Rechner, sondern im Kontext eines Objektes eine Methode aufgerufen wird. RMI gestattet es, Referenzen zu entfernten Objekte transparent zu behandeln.
RMI stützt sich auf ein weiteres API, *Object Serialization* [Sun Microsystems Inc. 96c], das die Funktionalität enthält, um Objekte zu serialisieren und aus einer Serialisierung Objekte zu erzeugen. Dies ist die Basis für das Marshalling und Unmarshalling der Parameter, die für RMI benötigt werden, um die Verteilung von Objekten und den Methodenaufruf über Netzwerke zur Verfügung zu stellen. RMI bietet im Unterschied zu üblichen RPCs nicht nur Referenzparameter (*call by reference*) sondern durch die mögliche Serialisierung der Objekte auch Wertparameter (*call by value*).
Mit RMI ist es möglich, die Anwendung zwischen Client und Anwendungs-Server aufzuteilen. Der Client enthält lediglich den Programmkode zur Darstellung einer Oberfläche und zur Reaktion auf Benutzereingaben. Die Java-Objekte der Anwendung, die nicht ausschließlich für diese Aufgabe benötigt werden, werden als entfernte Referenzen auf Objekte des Anwendungs-Servers realisiert. Die nötige Kommunikation zwischen Client und Anwendungs-Server geschieht nach der Definition der Schnittstelle zwischen Client und Anwendungs-Server transparent.

4.3.2 Plattformunabhänige Oberflächen mit AWT

Durch das *Abstract Window Toolkit* (AWT) des Java API steht eine plattformunabhängige Schnittstelle für den Aufbau graphischer Benutzeroberflächen auf jeder Plattform zur Verfügung, auf die die Java-Laufzeitumgebung portiert wurde. Da dies für alle kommerziell relevanten Plattformen der Fall ist, besteht die Möglichkeit, mit derselben Version des Java-Bytecodes einen Client für alle Client-Plattformen zu erstellen. Da das AWT eine Vielzahl von Plattformen unterstützt und das Nachprogrammieren von Eigenschaften, die nicht alle Plattformen bieten, sehr aufwendig und komplex ist, stellt das AWT jedoch nur den kleinsten gemeinsamen Nenner aller Plattformen dar und nutzt nicht alle Möglichkeiten, die eine Plattform bieten mag. Eine mit dem AWT erstellte Oberfläche fällt im Vergleich zu den Oberflächen plattformspezifischer Programme durch eine gewisse Schlichtheit auf. Bei der Funktionalität einer Benutzungsoberfläche an sich müssen jedoch keine Abstriche in Kauf genommen werden.

4.3.3 Datenbankanbindung mit JDBC

Zur persistenten Speicherung der Daten werden in Java-Anwendungen häufig relative Datenbanksysteme verwendet. Diese Datenbank-Technologie ist derzeit am weitesten verbreitet und befindet sich in einem weit fortgeschrittenen Entwicklungsstadium. Der Anwendungs-Server muß dabei eine Konversion von Java-Objekten zu den Relationen der Datenbank durchführen. Zur Umsetzung von den in Java realisierten Objektstrukturen auf die Tabellen in der relationalen Datenbank wird im Rahmen der OMT Methode eine Vorgehensweise vorgestellt. Der *impedance mismatch* [Manthey 91] von Datenbanken und Programmiersprachen sorgt trotz der o.a. Methode zur Umsetzung von Objektstrukturen auf relationale Datenbanken für viele Fehlerquellen.
Java Database Connectivity (JDBC) [Sun Microsystems Inc. 96a] nennt sich das API, um mit Java relationale Datenbanken anzubinden. Es lehnt sich an den de-

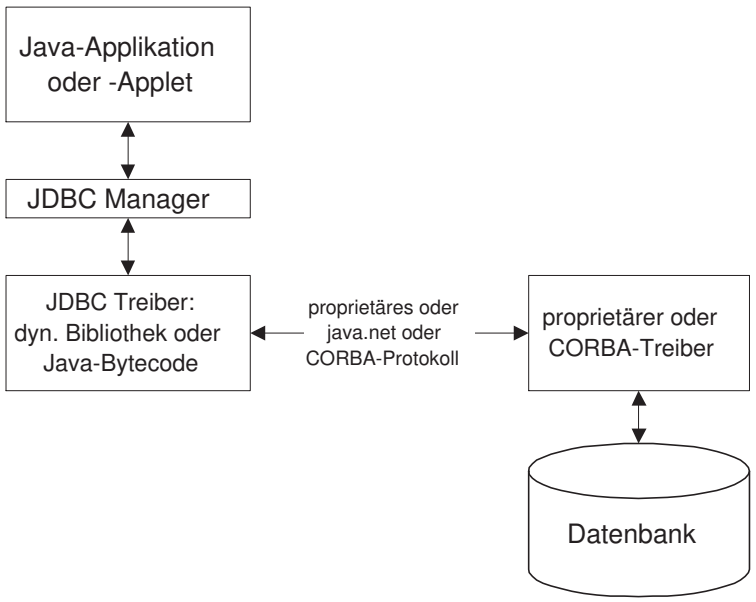


Abbildung 4.3: JDBC-Architektur

fakto Standard für den Zugriff auf relationale Datenbanken, die *Open Database Connectivity* (ODBC) [Simon 96] von Microsoft an. In Abbildung 4.3 ist die Architektur von JDBC zu sehen.
Für Java-Applikationen steht als Schnittstelle zur Nutzung relationaler Datenbanken der JDBC-Manager zur Verfügung. Der JDBC-Manager bildet das JDBC API und benötigt einen JDBC Treiber, um für die Kommunikation mit der Datenbank zu sorgen und die Funktionalität des API zu erbringen. Zu seinen Aufgaben gehört die Auswahl und Aktivierung des zur Datenbank passenden Treibers. Der JDBC-Treiber ist von der Datenbank und dem Betriebssystem des Datenbankrechners abhängig. Üblicherweise wird ein solcher Treiber in Form von dynamischen Bibliotheken oder in Java realisiert. Im letzteren Fall, daß der Treiber mittels Java realisiert wird, besteht der klare Vorteil, die Portabilität von Java auszunutzen und nicht auf plattformspezifischen Programmkode angewiesen zu sein.
Auf der Datenbankseite spricht ein weiterer, datenbank- und plattformabhängiger Treiber die Datenbank über eine proprietäre Aufrufschnittstelle an. Bei der *Oracle* Datenbank geschieht dies beispielsweise über das *Oracle Call Interface* (OCI), das eine Schnittstelle der Programmiersprache C zum Oracle Datenbankmanagementsystem realisiert. Alternativ ist eine Lösung über Komponenten nach der *Common Objekt Request Broker Architecture* (CORBA) [Orfali et al. 96] möglich, die den Datenbankzugriff als CORBA-Dienst kapselt und damit universell für Anwendungen in fast beliebigen Programmiersprachen und auf beliebigen Betriebssystemen zur Verfügung stellt.
Der JDBC Treiber kommuniziert über ein Protokoll mit dem Treiber auf der Datenbankseite. Dieses kann ein proprietäres Protokoll, das von Sun standardisierte *java.net*-Protokoll oder ein *Object Request Broker* (ORB)-Protokoll sein. Ein proprietäres Protokoll bedingt, daß der JDBC Treiber und der Treiber auf der Datenbankseite von demselben Hersteller stammen. Das java.net-Protokoll ermöglicht einen Aufbau, der mit einem einzigen JDBC Treiber in Java und unterschiedlichen datenbank- und plattformspezifischen Treibern arbeitet. Kommt eine neue Plattform für die Datenbank oder eine Datenbank hinzu, so ist lediglich ein Trei-

ber für diese spezielle Variante zu erstellen, der JDBC Treiber bleibt unverändert. Die CORBA-Variante unterscheidet sich von der java.net-Version dadurch, daß es möglich ist, die Datenbank nicht nur von Java, sondern von allen Plattformen, für die es CORBA-Implementationen gibt, und mit allen Programmiersprachen, die von CORBA mit Sprachbindungen unterstützt werden, zu nutzen.

4.3.4 Relationale Datenbanken

Das relationale Datenbankmanagementsystem *ORACLE* geht in der Architektur zurück auf den IBM-Forschungsprototyp *System R* [Heuer 92]. Es unterstützt die Überwachung von Fremdschlüsselbeziehungen und ermöglicht über objektrelationale Erweiterungen die implizite Vergabe von Primärschlüsseln. Als eine weit verbreitete Datenbank wird Oracle von einer Vielzahl von Treibern diverser Hersteller auf allen Plattformen unterstützt und ist daher gut für die Integration in eine verteilte Java-Anwendung geeignet.

4.4 Sicherheit

Da Applets bei jeder Benutzung erneut über das Netzwerk geladen werden, können sie eine Gefahr für die Sicherheit darstellen und die Systemstabilität und -integrität beeinträchtigen. Auf dem Weg vom Server zum Client kann das Applets unsichere Rechner passieren, die das Applet absichtlich verändern und so den Client oder den Server in ihrer Systemstabilität gefähren oder manipulieren. Auch die versehentliche Modifikation des Applets kann Daten zerstören oder in irgendeiner Art Schaden anrichten.
Um die Gefahren dieser Szenarien stark einzuschränken, werden Applets in einer eingeschränkten Umgebung ausgeführt, die schadhafte Auswirkungen verhindert soll. Dieses *sandbox* genannte Sicherheitsmodell wird näher in [Fritzinger, Mueller 96] erläutert. Zum sandbox-Modell gehört zum einen die Überprüfung des Bytecodes auf dem Client, die sicherstellen soll, daß das Applet keine illegalen Methodenaufrufe tätigt und während der Laufzeit nicht durch Laufzeitfehler abstürzt und damit die Systemintegrität des Client stört. Zum anderen gelten für Applets restrikte Einschränkungen, was den Zugriff auf lokale Ressourcen des Clients betrifft. Applets ist unter anderem nicht erlaubt

- auf das lokale Dateisystem zuzugreifen,
- Netzwerkverbindungen zu anderen Rechnern außer dem, von dem sie selbst geladen wurden, aufzubauen,
- dynamische Bibliotheken ins lokale System nachzuladen und
- Klassen aus dem Standard-Java-API, die nicht lokal vorhanden sind, nachzuladen.

Eine vollständige Aufstellung der Sicherheitsrestriktionen findet sich in [Flanagan 96]. Eine Laufzeitumgebung für Applets kann durch die Implementierung einer entsprechenden Subklasse des *SecurityManagers* alle wichtigen, oben genannten und einige weitere Sicherheitsrestriktionen einrichten. Bei allen bisher implementierten Ausführungsumgebungen wie java-fähigen World-Wide Web Browsern und *Appletviewern* wird dies sehr restriktiv gehandhabt. Die World-Wide Web Browser nutzen den vollen Umfang der Sicherheitsrestriktionen, die die SecurityManager Klasse zur Verfügung stellt. Die Appletviewer lassen sich für lokale Netze etwas weniger restriktiv einrichten.

Kapitel 5

Modellierung einer Arbeitsumgebung für die kooperative Bearbeitung von Literaturreferenzen

Die Analyse der Anwendungsszenarien, der Arbeitsmodi und der Benutzergruppen und die sich daraus ergebenden Anforderungen bilden die Grundlage für den Entwurf. Nach der OMT-Methode fallen diese Grundlagen in den ersten Abschnitt der Analysephase, in der von einer informalen Problembeschreibung ausgegangen wird, um einen Bereich der realen Welt zu modellieren. Die Analyse dient dem Verständnis des Problems. Als Vorbereitung für den Entwurf wird festgelegt, welche Funktionalität eine Arbeitsumgebung für die kooperative Bearbeitung von Bibliographien erfüllen soll, ohne irgendwelche Implementationsentscheidungen zu fällen.
Bei der OMT-Methode werden in der Phase des Systementwurfs im Rahmen der Erstellung einer Systemarchitektur die grundlegenden konzeptionellen und entwurfspolitischen Entscheidungen getroffen. Es wird ein Objektmodell, ein dynamisches Modell und ein funktionales Modell entworfen. Dieses Kapitel umfaßt die Phase des Systementwurfs insoweit, daß die grundlegende konzeptionellen und entwurfspolitischen Entscheidungen vorgestellt werden.
Der Entwurf eines Objektmodells, das den aufgestellten Anforderungen entspricht, ist in der Phase des Systementwurfs von großer Bedeutung, da das Objektmodell die Basis für die weiteren Modelle darstellt. Es wurde zunächst ein Objektmodell mit der Software StP/OMT entwickelt und mit dem Anforderungskatalog abgeglichen, bis nach einigen Iterationszyklen ein Stand erreicht war, der als Grundlage für eine prototypische Implementierung dienen konnte. Mit der anschließenden Implementation eines Prototypen werden die Modelle weiter verbessert und verfeinert.
Dieses Kapitel beschreibt die Analyse des BibTeX-Formats, die Modellierung des Objektmodells, des dynamischen Modells und des funktionalen Modells einer Arbeitsumgebung für die kooperative Arbeit mit Bibliographien und erläutert die wichtigsten Entwurfsentscheidungen, die getroffen wurden. Diese Modelle bilden die Grundlage für den in Kapitel 6 beschriebenen Prototyp.

5.1 BibTeX-Analyse

Die Untersuchung des BibTeX-Formats bildet den Ausgangspunkt für den Entwurf eines Objektmodells, das als geeignete Grundlage für eine prototypische Implemen-

tierung dienen kann. Von Interesse für die Untersuchung sind zum einen die Definition der Typen von Literaturreferenzen und deren Attribute sowie andererseits eine empirische Betrachtung der konkreten Inhalte, der Werte der Typen und der Attribute.

5.1.1 Definition des BibTeX-Formats

Das BibTeX-Format wird in einem Kapitel des LaTeX-Buchs [Lamport 85b] definiert durch eine Reihe von Literaturreferenztypen sowie einer Anzahl von erforderlichen und optionalen Attributen für jeden Typ. Für den Entwurf des Objektmodells ist von Interesse, welche Typen von Literaturreferenzen unterschieden werden, mit welchen Attributen die Literaturreferenzen ausgezeichnet werden müssen und können sowie wie der Zugriff auf die Literaturreferenzen geregelt wird.
Die Definition des BibTeX-Formats [Lamport 85a] ist sehr informal verfaßt und wird deshalb durch die Übersetzung vom Englischen ins Deutsche noch unpräziser. Als Beispiel sei hier die Definition des Literaturreferenztyps Buch genannt. Ein Buch (engl. *book*) wird als "book with an explicit publisher" definiert. Zum einen bezieht sich diese Definition auf sich selbst und wird wenig konkret, zum anderen besteht bei der Übersetzung ins Deutsche das Problem, daß der *publisher* ein Herausgeber, ein Verleger oder im amerikanischen Englisch der Besitzer eines Zeitungsverlages sein kann.
Bei der Betrachtung der Typen von Literaturreferenzen fällt auf, daß für die Typen *mastersthesis* und *phdthesis* ein Äquivalent im deutschen Sprachraum schwer zu finden ist. Für eine konsistente Struktur der Typen und eine präzisere Einordnung der Literaturreferenzen fehlt ein Typ *collection*, der einen Sammelband für wissenschaftliche Arbeiten beschreibt, für den mit dem Typ *incollection* eine Literaturreferenz besteht. Dieser Typ wurde analog zu den im BibTeX-Format definierten Paaren *book/inbook* und *proceedings/inproceedings* geschaffen.
Um einen Teil der zusätzlich in der Arbeitsumgebung enthaltenen Information zu den Literaturreferenzen auch nach dem Export aus der Arbeitsumgebung in eine Datei zur Verfügung zu haben, stehen für alle Typen von Literaturreferenzen zusätzlich zu den erforderlichen und den optionalen Attributen die von BibTeX ignorierten Attribute *owner*, *reg-date*, *change-date*, *abstract*, *keywords*, *directory* und *comment* bereit.

5.1.2 Literaturreferenzen im BibTeX-Format

Die Definition des BibTeX-Formats weist Schwächen auf, die im vorigen Abschnitt erläutert wurden. Neben der Analyse des BibTeX-Format erscheint es angebracht zu untersuchen, wie die Definition des BibTeX-Formats in der Praxis von Benutzern interpretiert wird. Dazu sind drei größere Bestände von Literaturreferenzen im BibTeX-Format, die aus der Erstellung umfangreicher wissenschaftlicher Publikationen über einen längeren Zeitraum hervorgegangen sind, zur Analyse des BibTeX-Formats herangezogen worden.
Bei dieser empirischen Untersuchung über die Verwendung der Literaturreferenztypen und deren Attribute für die betrachtete Menge wurde herausgefunden:

- Das Attribut *note* erscheint im Literaturverzeichnis und enthält Texte, die auf die Erscheinung der Literatur hinweisen. Beispiele hierfür sind "erscheint MM/YY", "to appear" und "nicht veröffenlicht".

- Felder wie *comment*, die nicht im Literaturverzeichnis erscheinen, werden eher weggelassen.

- Das Attribut *address* wird bei folgenden Literaturreferenztypen verwendet:

book/inbook als Adresse von Attribut *publisher*,
proceedings/inproceedings als Ort der Proceedings,
manual als Adresse vom Attribut *organization*,
mastersthesis als Adresse vom Attribut *school*,
techreport als Adresse vom Attribut *institution*,
unpublished als Adresse oder URL, über die die Literatur zu beziehen ist.

- Das Attribut *howpublished* wird überwiegend für URLs benutzt.

Eine Reihe von Attributen enthalten oftmals Texte, für die mit dem @STRING-Makro [Lamport 85a] eine Kurzform definiert wurde. In der Literaturreferenz steht dann die Kurzform, die bei der Verarbeitung durch das BibTeX-Programm zur Langform expandiert wird. Neben dem geringeren Schreibaufwand für den Benutzer vereinheitlicht sich dadurch die Schreibweise der Inhalte der Attribute. Die ist ein positiver Nebeneffekt für den Aufbau eines konsistenten Bestandes. Weiter wurde herausgefunden:

- Die Attribute *institution*, *organization* und *school* enthalten Namen von Institutionen, Organisationen und Bildungseinrichtungen. Beispiele sind:

  institution FIDE Project Coordinator, Department of Computing Sciences, University of Glasgow, Glasgow G128QQ; Digital Equipment Corporation, Systems Research Center, Palo Alto, California

  organization IEEE; ISO/IEC JTC 1/SC 21/DIS 13235; Sun Microsystems; Star Division, Software Development

  school Stanford University; Fachbereich Informatik, Universität Hamburg, Germany; Massachusetts Institute of Technology

- Die Attribute *journal* und *series* beschreiben die Namen von Zeitschriften und von Publikationen im Bereich der Informationstechnologie, die periodisch im Rahmen einer Reihe erscheinen. Beispiele sind:

  journal ACM SIGPLAN Notices; ACM Transactions on Database Systems; Informatik Spektrum; BYTE

  series Topics in Information Systems; Lecture Notes in Computer Science; Workshops in Computing; Frontier Series

- Das Attribut *edition* enthält die Ausgabe einer Publikation. Mit dem Attribut *number* wird die Nummer innerhalb einer Reihe von Publikationen angegeben, die eine Publikation eindeutig identifiziert. Beispiele hierfür sind:

  edition 1st; 2nd; 3rd; first; second; third

  series 4,47; FIDE/94/102; 50-90; 91.1.4.9

- Das Attribut *type* ordnet Literaturreferenzen vom Typ *techreport* eine Klasse von Reports zu. Beispiele dazu sind:

  type FIDE Technical Report Series; Technical Report; Research Report; Esprit Project 892

Aus der ursprünglichen Definition des BibTeX-Formats ist so durch die Analyse der Definition des BibTeX-Formats und durch die empirische Untersuchung der Verwendung der Attribute und Typen eine Verfeinerung der ursprünglichen Definition entstanden. Diese Verfeinerung (s. Anhang A.1) soll nicht das BibTeX-Format erweitern, sondern für den Benutzer durch eine präzisere Formulierung der Definitionen eine einheitliche Zuordnung von Literaturreferenzen zu Typen erreichen und darüber die Qualität des Bestandes und damit die Genauigkeit der Recherche verbessern.

5.1.3 Definition der Schlüssel

Die Wahl der Schlüssel ist für den BibTeX-Benutzer völlig frei. Eine einheitliche Auszeichnung der Literaturreferenzen mit Schlüsseln, die dann im LaTeX-Dokument verwendet werden, findet maximal innerhalb einer Arbeitsgruppe statt.
Für eine Arbeitsumgebung, die die kooperative Arbeit mit Literaturreferenzen unterstützt, ist deshalb eine auf klaren, einheitlichen Regeln aufbauende Definition der Schlüssel wichtig. Eine intuitive, schematische Bildungsregel für die Schlüssel, die die zentralen Attribute der Literaturreferenz wie die Namen der Autoren benutzt, bietet die Möglichkeit, bei Erstellung einer Literaturreferenz ohne Zugriff auf den BibTeX-Bestand fortzufahren mit der Erstellung des Dokuments.
Es wurden eine Reihe von Bildungsregeln für Schlüssel untersucht. Es soll genügen, daß der Benutzer die Autoren und das Erscheinungsjahr kennt, um den Schlüssel bilden zu können. Sollten Autoren mehrere Publikationen im Jahr erstellen, sind diese zu unterscheiden. Es wird die Bildung der Schlüssel nach folgendem Schema vorgeschlagen:

- 3 Buchstaben pro Autor, erster Buchstabe vom Nachnamen großgeschrieben, die beiden folgenden klein
- maximal 3 Autoren, ab dem 4. Autor ein Plus hinter dem 9. Buchstaben
- keine Umlaute
- zweistellige Jahreszahl
- bei Gleichheit kleiner Buchstabe als Anhängsel[1], beginnend bei a – Beispiel: Mue94 MueMue94 MueMueMue94 MueMueMue+94 Mue94a Mue94b

5.2 Objektmodell

Mit dem Objektmodell wird die statische Struktur eines Systems festgehalten. Das Objektmodell zeigt die Objekte in dem System, die Beziehungen zwischen Objekten und die Attribute und Operationen, die jede Klasse von Objekten charakterisieren. In der Phase des Systementwurfs kommt dem Entwurf eines Objektmodells für Literaturreferenzen, das den aufgestellten Anforderungen entspricht, die größte Bedeutung zu, da es die Basis für die weiteren Modelle ist.
In den Graphen der Objektmodelle stehen Knoten für Klassen und Kanten für die Beziehungen zwischen Klassen.

5.2.1 Überblick über die Modellierung

Nach der Untersuchung des BibTeX-Formats ist die daraus entstandene, verfeinerte Definition des BibTeX-Formats (s. Abschnitt A.1) die Basis für den Entwurf des Objektmodells. Zusätzlich haben die Zitierregeln der APA (s. Abschnitt 2.2.3) die Modellierung vor allem durch deren Art und Weise mit den Neuerungen umzugehen, die durch das neue Medium Internet entstanden sind, beeinflußt.
Die Entwurfsentscheidungen bei der Modellierung der grundlegenden Klassen waren:

reference Die Klasse *reference* am oberen Ende der Klassenhierarchie beinhaltet die Gemeinsamkeiten aller Literaturreferenztypen sowie die Attribute und Assoziationen, die für die Funktionalität der Arbeitsumgebung notwendig sind.

[1] In diesem Fall ist es allerdings für den Benutzer nicht mehr möglich, die Literaturreferenz nur aus dem Namen der Autoren und dem Erscheinungsjahr herzuleiten.

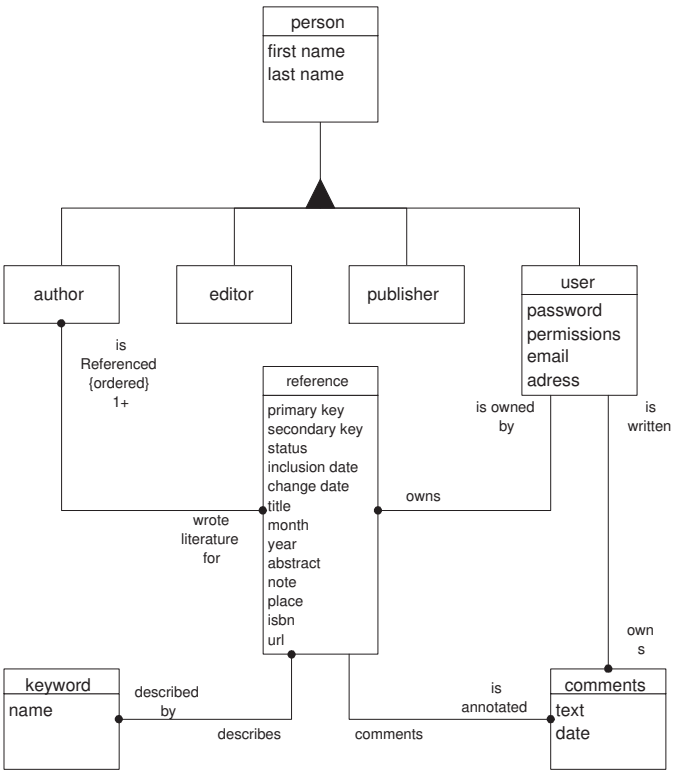


Abbildung 5.1: Klasse reference

Dieser Zusammenhang ist im Objektdiagramm von Abbildung 5.1 gezeigt. Es stellt den Kern des Objektmodells dar. Die Klasse *reference*

- erhält einen Primärschlüssel (*primary key*), der unabhängig macht gegenüber Schlüsseländerungen bei BibTeX-Schlüsseln. Solche Änderungen sollten zwar nicht oder so so gut wie nie auftreten, können aber z.B. bei Fehlern wie falscher Reihenfolge der Autoren (*author*) evtl. nicht verhindert werden.
- bekommt zwei BibTeX-Schlüssel-Attribute: eines (*primary key*), um ein neues Schema mit intuitiver, schematischer Vergabe der BibTeX-Schlüssel (s. Abschnitt 5.1.3) zu ermöglichen; ein weiteres (*secondary key*), um alte Schlüssel übernehmen zu können und kompatibel zu Dokumenten mit den alten Schlüsseln zu bleiben, da eine Änderung der alten Dokumente zuviel Aufwand bedeutet.
- stellt mit dem Attribut *status*, die Zustände der Literaturreferenz wie "unvollständig", "exportiert", "modifiziert" u.Ä. (s. Abschnitt 5.3) dar.
- enthält eine Assoziation zur Klasse *keywords*, um Suchanfragen nach festgelegten Suchbegriffen besser zu unterstützen.

Das Inhaltsverzeichnis wird nicht in die Klasse *reference* aufgenommen. Ein Abstract (*abstract*), der Standord der Literatur (*place*) und Kommentare (*comments*) des Benutzers sind sinnvoller, da ein Inhaltsverzeichnis selten abgetippt werden wird. Selbst wenn dies der Fall sein sollte, dann ist das Inhaltsverzeichnis im Abstract auch gut aufgehoben ist, da dessen Funktion ein Überblick über den Inhalt ist.

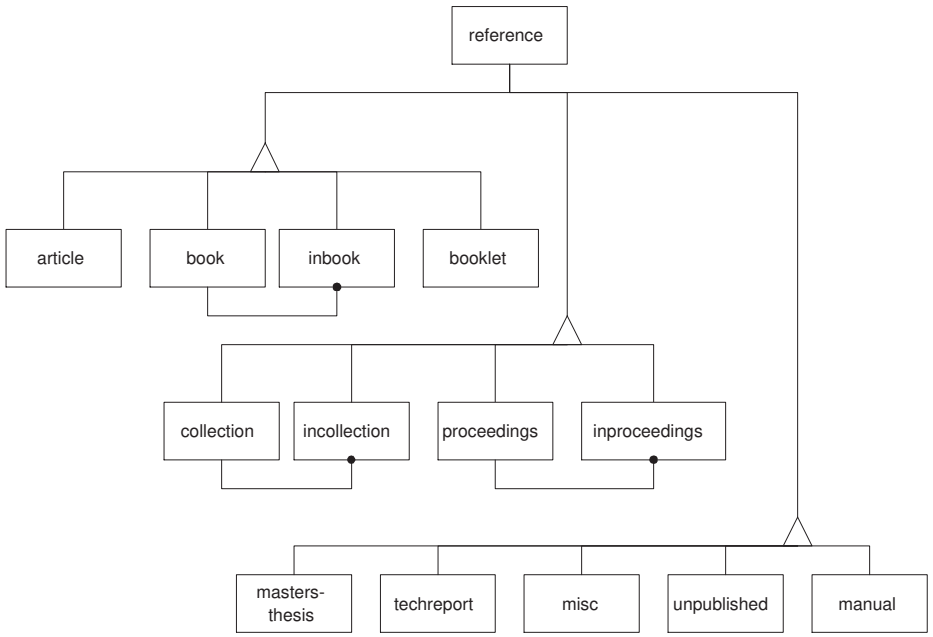


Abbildung 5.2: Hierarchie der Referenzen

person Die Klasse *person* ist die Oberklasse von *author*, *editor*, *publisher* und *user*, damit keine Personen doppelt vorkommen, wenn z.B. ein Autor auch Editor ist.

Die Subklassen der Klasse *reference*, die in Abbildung 5.2 dargestellt sind, stehen für die einzelnen Typen von Literaturreferenzen. Alle Subklassen haben als Superklasse die Klasse *reference* und erben von ihr die Attribute und Assoziationen. Daneben besitzen sie auch eigene Attribute und Assoziationen. Ein Übersicht über alle Klassen findet sich in Anhang B.

5.2.2 Genauere Modellierung

Die Subklassen *booklet* und *misc* von *reference* sind sehr einfach, sie erweitern *reference* um ein oder zwei Attribute. Die Subklasse *unpublished* ist gar identisch mit *reference*.
Im Gegensatz dazu liegen die Klassen *article*, *inproceedings* und *mastersthesis* von der Komplexität, d.h. der Anzahl der zusätzlichen Attribute und Assoziationen, eher in der Mitte der Bandbreite, die die Subklassen von *reference* aufweisen. Klassen wie *book*, *collection*, *proceedings* und *techreport* sind mit drei zusätzlichen Attributen und Assoziationen zu fünf weiteren Klassen recht komplex. Als Beispiel für eine komplexere Klassen sei hier die Klasse *book*, dargestellt als Objektdiagramm in Abbildung 5.3, herausgehoben.
Die Entwurfsentscheidungen für die grundlegenden Klassen sind im Folgenden aufgeführt:

corporation Die Klasse *corporation* ist die Oberklasse von *institution*, *organization* und *school*. Es gibt Überschneidungen zwischen den drei Klassen, deshalb handelt es sich um eine Inklusionsbeziehung. Der Vorteil dieser Modellierung liegt darin, daß die Namen nur einmal und damit konsistent auftreten.

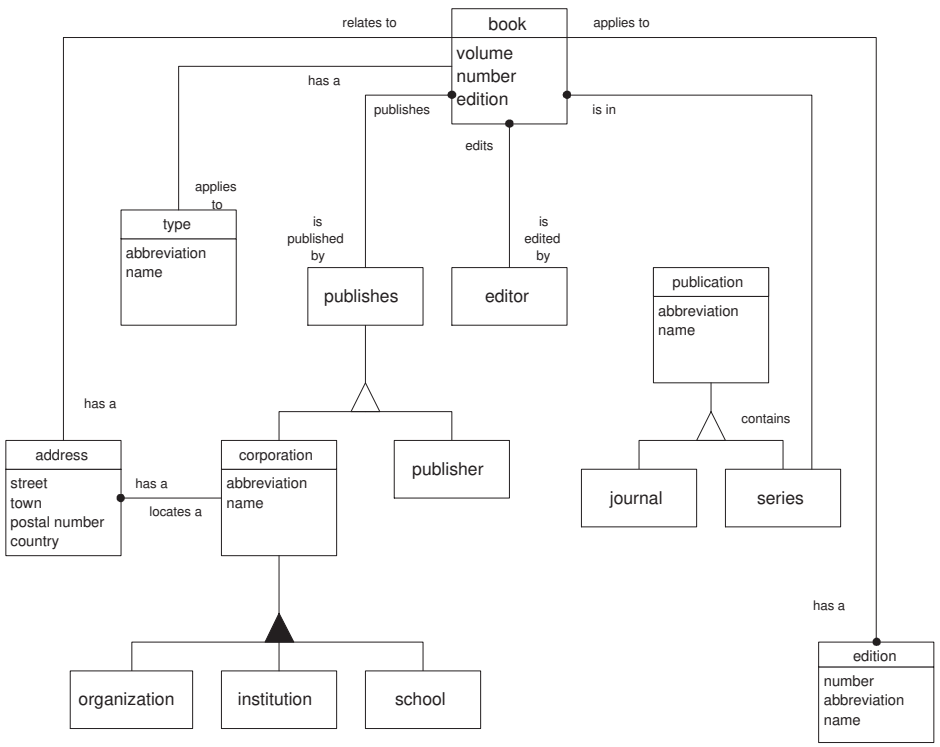


Abbildung 5.3: Klasse book

publishes Mit der Oberklasse *publishes* der Klassen *corporation* und *publisher* besteht die flexible Möglichkeit, den oder die für die Publikation Verantwortlichen unter einer gemeinsamen Oberklasse zu präsentieren. Im Normalfall ist *publishes* eine Subklasse von *corporation*. Für die Ausnahmen, wenn eine Einzelperson eine Publikation vornimmt, steht die Klasse *publisher* bereit.

publications Die Klasse *publications* ist die Oberklasse von *journal* und *series*, da der Titel eines Journals, eines Magazins, einer Zeitschrift, einer Zeitung, wo ein Artikel erscheint bzw. Name einer Reihe von Bücher, eines Journals häufig ähnlich ist. Es bietet sich wieder der Vorteil einer vereinheitlichten Schreibweise und den daraus folgenden Vorteilen.

edition Die Klasse *edition* ist eigene Klasse mit Nummer (1,2,3..), Abkürzung (1st, 2nd..) und ausgeschriebenem Namen (first, second..) der Ausgaben. Damit werden uneinheitliche Schreibweisen wie z.B. "1st" vs. "first" vermieden, indem eine *reference* nur eines der Objekte der Klasse *edition* referenzieren kann.

type Die Klasse *type* enthält die Klassifikation eines technischen Reports, um den Inhalt des Reports besser abschätzen zu können. Durch eine eigene Klasse wird die Anzahl der Klassifikationen beschränkt. Gleichzeitig vereinfacht sich die Einordung und das Ergebnis der Recherche wird verbessert.

address Die Klasse *address* hat sowohl eine Assoziation mit *publishes*, woraus die aktuelle Adresse hervorgeht als auch mit der Klasse *reference* selbst, was auf die Adresse zum Zeitpunkt der Erstellung der Literaturreferenz hindeutet. Es ist damit möglich, Adreßänderungen zu berücksichtigen, ohne Literaturreferenzen fälschlich ändern zu müssen.

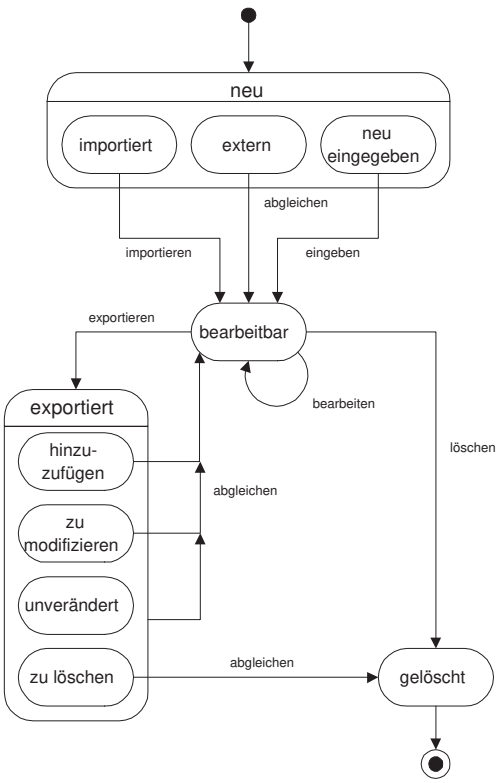


Abbildung 5.4: Zustände einer Literaturreferenz

Darüber hinaus gibt es noch eine neue Klasse *conference*, die für die Klasse *proceedings* benötigt wird, um eine Konferenz eindeutig zu bezeichnen.

conference Die Klasse *conference* wird von der Klasse *proceedings* benutzt, um Konferenzen eindeutig zu bezeichnen und auch die Zuordnung von Kurzformen und Langformen der Konferenztitel zu unterstützen.

Durch die Assoziationen mit einer Vielzahl weiterer Klassen können gemeinsame Daten zwischen verschiedenen Objekten der Klasse *book* geteilt werden. Dadurch liegt diese Modellierung näher an der Realität und es ist möglich, eine höhere Konsistenz der Daten zu erreichen als bei einer einfachen BibTeX-Datei. Ein Objekt der Klasse *publishes* beispielsweise kann über die Subklassenbeziehung eine Firma (*corporation*) oder eine Person (*publisher*) in der Realität repräsentieren. Wenn die Firma oder die Person mehr als ein Buch herausgibt, kann dies entsprechend dargestellt werden.

5.3 Dynamisches Modell

Bei der dynamischen Modellierung werden Veränderungen des Zustandes des Systems, d.h. seiner Objekte, über die Zeit betrachtet. Die Graphen der Zustandsdiagramme enthalten als Knoten Objektzustände. Die Kanten stellen den von Ereignissen ausgelösten Übergang zwischen Objektzuständen dar.
Eine Literaturreferenz befindet sich in einem der Zustände, die zusammen mit den Übergängen zwischen den Zuständen im Zustandsdiagramm von Abbildung 5.4 beschrieben sind. Die Zustände stellen sich wie folgt dar:

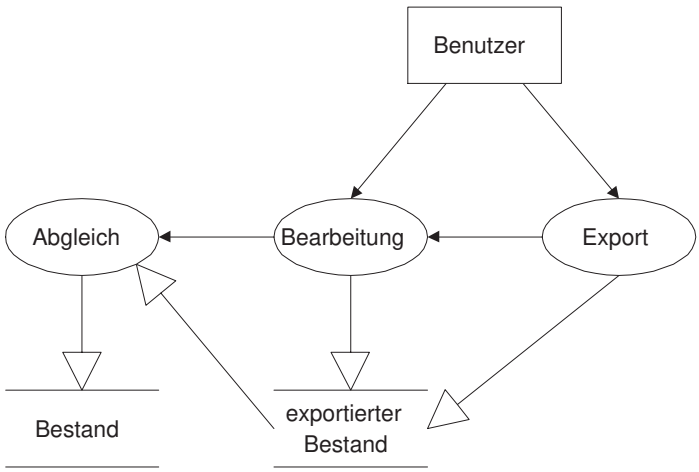


Abbildung 5.5: Arbeiten mit einem exportierten Bestand

Wenn eine Literaturreferenz neu in den Bestand aufgenommen wird, kann dies durch den Import aus einem externen Bestand, durch den Abgleich mit einem externen Bestand oder durch die komplette Neuerstellung geschehen. Im Unterschied zum Import bleibt wird einem Abgleich der externe Bestand weiter gepflegt. Ist eine Literaturreferenz in dem Bestand, dann ist sie bearbeitbar. Sie kann editiert werden, woraufhin sie wieder bearbeitbar ist. Wird die Literaturreferenz entfernt, ist sie gelöscht.
Beim Export in einen Bestand, beispielsweise, um autonom zu arbeiten, bestehen ebenfalls die Möglichkeiten, neue Literaturreferenzen anzulegen, bestehende zu bearbeiten oder zu löschen. Durch einen Import wird der exportierte Bestand mit dem Bestand abgeglichen. Die im exportierten Bestand gelöschten Literaturreferenzen werden im Bestand gelöscht, die im exportierten Bestand modifizierten Literaturreferenzen werden im Bestand geändert, die im exportierten Bestand neu erstellten Literaturreferenzen werden dem Bestand hinzugefügt.

5.4 Funktionales Modell

Die funktionale Modellierung befaßt sich mit der Transformation von Daten durch Prozesse. In den Graphen der Datenflußdiagramme sind die Knoten Prozesse oder Datenspeicher und die Kanten stellen Kontroll- oder Datenflüsse dar.
Das Datenflußdiagramm von Abbildung 5.5 zeigt das funktionale Modell für den Export von Literaturreferenz aus dem Bestand in einen externen Bestand und den anschließenden Abgleich. Dies geschieht so: Der Benutzer stößt den Export von Literaturreferenzen in den externen Bestand an und bearbeitet diese. Unter Bearbeiten ist dabei das Hinzufügen, Verändern und Löschen zu verstehen, wie dies aus Abbildung 5.4 im vorigen Abschnitt hervorgeht. Nach diesen Modifikationen führt ein Abgleich den exportierten Bestand mit dem Bestand zusammen.
Im Datenflußdiagramm von Abbildung 5.6 ist das Arbeiten auf dem Bestand mit Selektion und Bearbeitung von Literaturreferenzen sowie Abgleich und Import von externen Beständen dargestellt. Dies stellt sich wie folgt dar: Externe Bestände können durch einen Abgleich in den Bestand eingebracht werden. Wird der externe

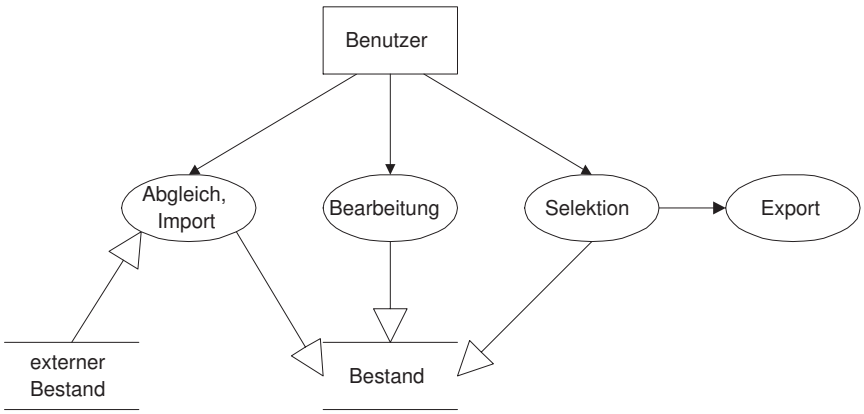


Abbildung 5.6: Arbeiten mit dem Bestand

Bestand nach dem Abgleich entfernt oder nicht mehr genutzt, entspricht dies einem Import. Dies ist wieder analog zu Abbildung 5.4, wo Literaturreferenzen entweder durch einen Abgleich oder einen Import neu in den Bestand kommen konnten, sonst entspricht es dem dort erwähnten Abgleich.
Der Bestand kann durch Benutzer bearbeitet werden, Literaturreferenzen können hinzugefügt, modifiziert und gelöscht werden. Schließlich können ausgewählte Literaturreferenzen des Bestandes selektiert werden, um Publikationen, die der Benutzer erstellt, zuzuordnen. Die selektierten Literaturreferenzen können wieder exportiert werden.

Kapitel 6

CobWeb

Der Prototyp basiert auf den in der Phase des Systementwurfs getroffenen, grundlegenden konzeptionellen und entwurfspolitischen Entscheidungen. Die in Kapitel 5 vorgestellen Modelle - Objektmodell, dynamisches Modell und funktionales Modell - bilden die Grundlage für den Prototyp. Vor der Implementierung des Prototypen wird überlappend mit der letzten Phase des Systementwurfs in der Phase des Objektentwurfs eine Systemarchitektur mitsamt der benötigten Komponenten und Klassen entworfen.
Nach Floyd [Floyd 84] und Grønbæk [Grønbæk 88] werden in der Softwareentwicklung zwei Arten von Prototypen unterschieden:

Horizontaler Prototyp Bei einem *horizontalen Prototypen* sind alle sichtbaren Teile der Benutzerschnittstelle eines neuen Computersystems implementiert, d.h. es können Bildschirmdialoge simuliert werden, ohne tatsächlich Daten zu berechnen.

Vertikaler Prototyp Bei einem *vertikalen Prototypen* sind einige der Systemfunktionen in ihrer beabsichtigten endgültigen Form implementiert, so daß eine in der Realität vorkommende Arbeitsaufgabe bearbeitet werden kann.

Mit *CobWeb* wird ein Prototyp implementiert, der nach der obigen Definition in die Kategorie der vertikalen Prototypen eingeordnet werden kann. So kann einerseits das verwendete Datenmodell validiert und andererseits die Realisierbarkeit trotz des sehr frühen Stadiums der benutzten Komponenten überprüft werden. Dabei besteht die Gelegenheit, einen Einblick in die Komplexität des Zusammenwirkens der Komponenten zu erhalten.
CobWeb stützt sich auf zwei Java Application Programming Interfaces, *Remote Method Invocation* und *Java Database Connectivity*, die in der Version 1.1 [Sun Microsystems Inc. 97a] des *Java Developer Kit* in ihrer endgültigen Form zur Verfügung stehen werden. Zum Zeitpunkt des Objektentwurfs und der Implementation des Prototypen standen sie jedoch erst in vorläufigen, zum Testen freigegeben Versionen zur Verfügung.
Dieses Kapitel gibt einen Einblick in die konkrete Implementation. Es werden die Architektur und die Komponenten von CobWeb vorgestellt. Interessante Aspekte, die sich durch die Verteilung der Komponenten, die unterschiedliche Darstellung und gleichzeitige Verteilung der Daten ergeben sowie Erfahrungen, Erkenntnisse und Schwierigkeiten bei der Realisierung werden beschrieben.

6.1 Architekturentwurf

Die Architektur von CobWeb und ihre Aufteilung in Komponenten zeigt Abbildung 6.1. Der Anforderungskatalog an eine Arbeitsumgebung für die kooperative Arbeit mit Bibliographien bestimmt den Entwurf und die Abgrenzung der einzelnen Komponenten. Im folgenden sind die Komponenten und die Anforderungen, die sie erfüllen, kurz beschrieben.

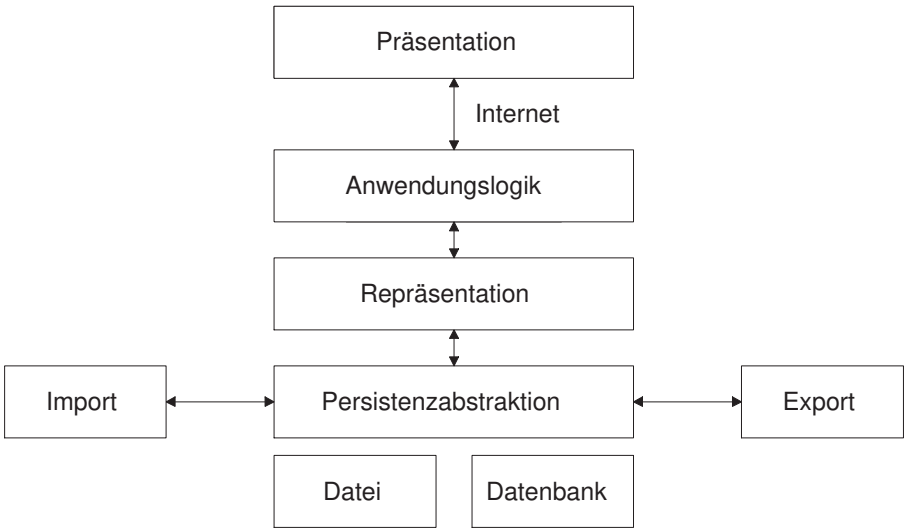


Abbildung 6.1: Architekturüberblick

Die Komponente *Präsentation* stellt eine Benutzungsoberfläche für die Funktionalität, die CobWeb bietet, bereit. Dazu gehören der Aufbau der Elemente der Benutzungsoberfläche und die Abwicklung der Interaktion mit dem Benutzer. In der Komponente *Anwendungslogik* stehen die Verwaltung der Literaturreferenzen und der geordnete Zugriff mehrerer Benutzer auf die Literaturreferenzen für die Präsentation zur Verfügung.
Die Trennung der Komponenten Präsentation und Anwendungslogik mit der Möglichkeit zur Verteilung über das Internet genügt den Anforderungen nach der Verfügbarkeit und der Flexibilität der Arbeitsumgebung. Ein Internetzugang genügt, um an einem beliebigen Ort zu einer beliebigen Zeit die Arbeitsumgebung nutzen zu können.
Die Anwendungslogik-Komponente setzt die grundsätzlichen Anforderung nach einem Mehrwert durch die Erfüllung konkreter Anforderungen nach Personalisierung, Benutzerverwaltung, Zugriffsschutz und -rechte, Sichten sowie Eingabehilfen und -unterstützung, Schlüsselgenerierung und Versionsverwaltung um.
Die Komponente *Repräsentation* stellt ein Datenmodell für die Anwendungslogik bereit, auf der die Anwendung ihre Funktionalität für die Komponente Repräsentation erbringen können. Über eine *Persistenzabstraktion* können sowohl Dateien als auch eine Datenbank zur Speicherung genutzt werden. Die *Datei*-Komponente erfüllt die Anforderung nach Autonomie, die *Datenbank*-Komponente dient zur Erfüllung der grundsätzlichen Anforderung nach Qualität, denn sie bietet Fehlererholung, Konsistenzsicherung, Sichten und Zugriffsrechte.
Die Komponenten *Import* und *Export* überführen Daten aus externen Formaten in die Repräsentation und zurück. Sie erfüllen die grundsätzlichen Anforderungen nach Kontinuität und Kompatibilität sowie die konkreten Anforderungen nach Im- und Exportfiltern und Offenheit. Für das BibTeX-Format stehen Im- und Exportfilter

bereit, damit weiter mit dem TeX-System gearbeitet werden kann. Weitere Formate für Literaturreferenzen lassen sich durch die Persistenzabstraktion und die Erweiterung vorhandener Filter leicht ergänzen. Mittels der *Datei* Komponente ist auch die Anforderung nach Autonomie erfüllt, denn sie ermöglicht das lokale Arbeiten, erfordert jedoch einen Abgleichvorgang, um den Datenbestand aktuell und konsistent zu halten.

6.2 Implementierungsdetails der Architektur

Die im vorigen Abschnitt beschriebene Architektur definiert abstrakt einzelne Komponenten innerhalb der Architektur. Eine mögliche Realisierung einer solchen Architektur mit den in Kapitel 4 beschriebenen Komponenten, Werkzeugen und Sprachen, sieht man in Abbildung 6.2 und Abbildung 6.3. In den folgenden Abschnitten sind der Client und der Server der Systemarchitektur beschrieben.

6.2.1 Client

Wenn der Benutzer CobWeb starten möchte, sucht er eine HTML-Seite auf, die das CobWeb-Applet enthält. Dazu ist ein World-Wide Web Browser mit Java-Unterstützung notwendig (s. Abschnitt 4.2.2). Die HTML-Seite und das Applet werden durch das *Hypertext Transfer Protokoll* (HTTP) von einem Web-Server zum Client übertragen, wodurch die Verteilung der Anwendung und damit deren Verfügbarkeit erreicht wird. Der Browser stellt die HTML-Seite dar und überläßt dem Applet, das in der virtuellen Maschine läuft, einen Rahmen auf der HTML-Seite, um dort z.B. die Anwendung darzustellen.

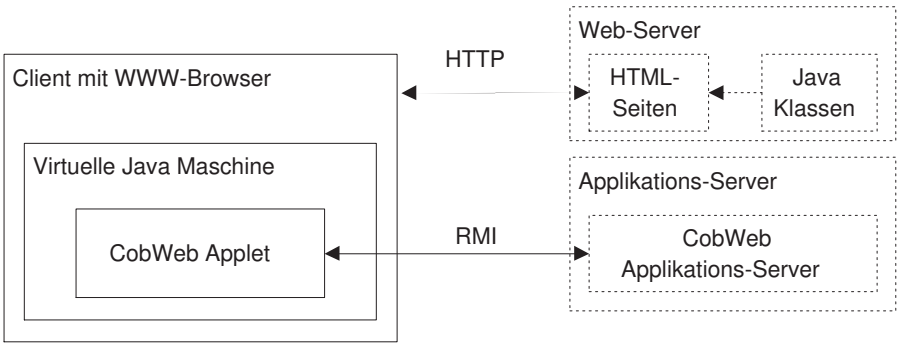


Abbildung 6.2: Technische Komponenten des Clients

Das CobWeb-Applet übernimmt die Präsentation der Anwendung. Der Anwendungsprozeß CobWeb Applikations-Server läuft auf dem Anwendungs-Server und stellt die Anwendungslogik (und die Repräsentation) für die Präsentation zur Verfügung. Applet und Applikations-Server kommunizieren über Remote Method Invocation, das dem Applet entfernte Objektreferenzen und Methodenaufrufe an Objekte auf dem Applikations-Server ermöglicht. Dieser Mechanismus wird bereits in Abschnitt 4.3.1 beschrieben.
Aufgrund des Java Sicherheitsmodells (s. Abschnitt 4.4) gilt die Einschränkung daß der Web-Server und der Anwendungs-Server auf demselben Rechner laufen müssen.

6.2.2 Server

Der Server der Systemarchitektur ist auf Abbildung 6.3 dargestellt. Der CobWeb Applikations-Server stellt nicht nur die Anwendungslogik für die Präsentation zur Verfügung, sondern leistet auch die Persistenzabstraktion zur Datenbank.

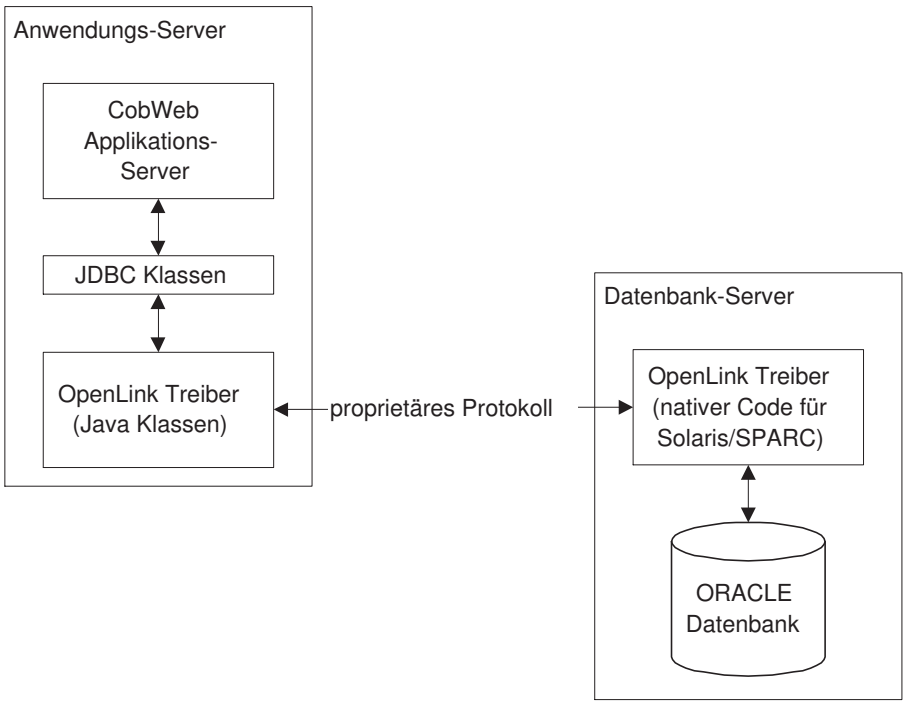


Abbildung 6.3: Technische Komponenten des Servers

Der Zugriff auf die relationale Datenbank über JDBC wurde in Abschnitt 4.3.3 beschrieben. Über die JDBC Klassen und den darunterliegenden Treiber wird mit dem Treiber auf dem Datenbank-Server ein proprietäres Protokoll gefahren, um die relationale Datenbank Oracle auf dem Datenbank-Server anzusprechen und zur dauerhaften Speicherung der Daten zu nutzen. Die verwendeten JDBC-Treiber stammen von OpenLink. Der Treiber auf dem Client ist in Java geschrieben und plattformunabhängig. Der Treiber auf dem Anwendungs-Server ist in C geschrieben und unter diversen Unix-Betriebssystem als Dämon realisiert. Damit ist er sowohl vom Betriebssystem des Anwendungs-Servers, in diesem Fall Solaris, als auch von der verwendeten Oracle-Datenbank abhängig, da er mangels Standards eine proprietäre Datenbankaufrufschnittstelle nutzt. Bei der CobWeb Systemarchitektur ist dies das Oracle Call Interface.

6.2.3 Datensichten

Bei der Implementation der Persistenzabstraktion zeigt sich, daß die Sicht auf die Objekte der Anwendung abhängig von der Komponente Unterschiede aufweist und damit nicht vollständig transparent ist. In Abschnitt 4.3.3 wurde angesprochen, daß sich das OMT-Modell in einigem Punkten vom Java-Modell und grundlegend vom relationalen Modell unterscheidet. Die unterschiedlichen Sichten auf die Daten sind in Abbildung 6.4 in der jeweiligen Notation des Modell dargestellt. Ausgehend von der Assoziation zwischen den Klassen *Reference* und *Author* im OMT-Modell

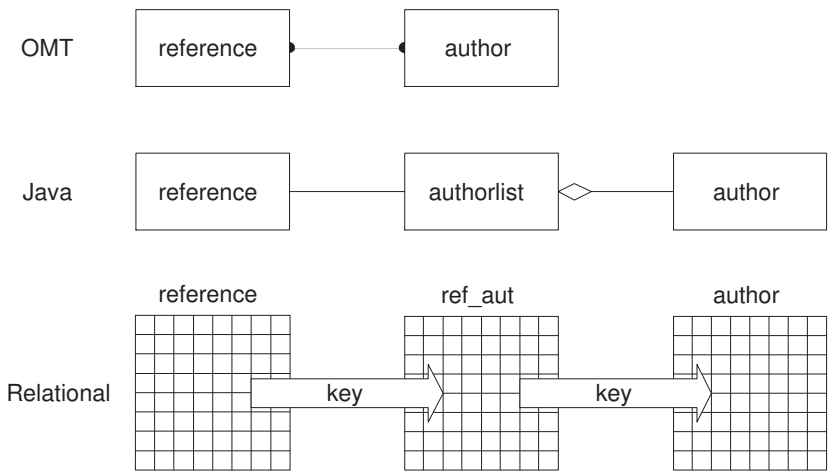


Abbildung 6.4: Datensichten

wird eine Implementation dieser Assoziation in Java und die Umsetzung der in Java implementierten Objektstruktur im relationalen Modell gezeigt.
Während die Modellierung mit OMT noch weitgehend der Struktur der Daten in der Realität folgt, zeigt sich bei der Implementierung, daß die Einschränkung von Java, keine Assoziationen wie mit OMT realisieren zu können, zu einer Modifikation und Erweiterung der ursprünglich modellierten Klassen führt. In Java stehen drei gangbare Wege zur Verfügung, um die Assoziationen von OMT zu simulieren:

- Man realisiert die Assoziation über die *java.util.Vector*. Diese aggregiert über die zu speichernden Objekte und stellt so die Assoziation dar. Dadurch wird auf die Typisierung verzichtet, da *java.util.Vector* nicht mit dem Typ der zu verwaltenden Objekte parametrisiert werden kann und mit Objekten der allgemeinen Klasse *java.lang.Object* arbeitet.

- Man erstellt eine eigene Klasse für Assoziationen, deren Interface der Klasse *java.util.Vector* aus dem Java API stark ähnelt. Diese eigene Klasse arbeitet intern wie *java.util.Vector* mit Objekten der Klasse *java.lang.Object*, ist also ein Adapter [Gamma et al. 95], der die Methodenaufrufe an ein Objekt der Klasse *java.util.Vector* weiterleitet. Im Unterschied zu *java.util.Vector* sorgt die eigene Klasse durch Abfragen zur Laufzeit für die Typsicherheit. Dies ist in folgendem Programmkodeausschnitt angedeutet.

```
class Association {
  Class typ;

  void insert (Objekt object) {
    if (classof(object) == typ) {
      ... insert operation ...
    }
    else
      raise Exception();
  }
}
```

  Beim Aufruf des Konstruktors der Klasse *Association* wird die Klasse der Objekte, die zu verwalten sind, in dem Attribut *typ* gespeichert. Bei jeder

Operation auf der Liste wird die Klasse des übergebenen Objekts mit der in *typ* gespeicherten verglichen. Gehören beide zur gleichen Klasse, dann wird die Operation ausgeführt, sonst wird eine Ausnahme ausgelöst.

- Man realisiert zu jeder Klasse X eine Klasse $XAssociation$, die über die Objekte von X aggregiert. In der Implementation heißt dies, daß $XAssociation$ Objekte der Klasse X verwaltet. $XAssociation$ selbst ist gekapselt und bietet Methoden zum Hinzufügen, Entfernen und Suchen von Objekten der Klasse X bietet. Weiterhin kann nach dem Entwurfsmuster *Iterator* [Gamma et al. 95] auf die Objekte zugriffen werden, ohne die konkrete Implementation der Verwaltung offenlegen zu müssen.

Die letzte Variante ist sicherlich die aufwendigste der drei, da für jede Assoziation eine weitere Klasse implementiert werden muß. Dafür bleibt der Programmkode leichter lesbar, es sind keine Type-Casts zur Laufzeit erforderlich und es ist nicht nötig, auf die strenge Typisierung zu verzichten. Da der Umfang des Prototypen CobWeb nicht so groß ist, kam die aufwendige letzte Variante zum Einsatz.
Zur Umsetzung von den in Java realisierten Objektstrukturen auf die Tabellen in der relationalen Datenbank wurde in Abschnitt 4.3.3 eine Vorgehensweise im Rahmen der OMT-Methode vorgestellt.

6.2.4 Objektverteilung

Die im vorigen Abschnitt beschriebenen Datensichten werden nun unter dem Aspekt der Verteilung auf unterschiedliche Komponenten dargestellt. Durch die Verteilung der Komponenten über Rechnergrenzen und Adreßräume hinweg kommt eine zusätzliche Komplexität in die Sicht der Daten hinein. in Abbildung 6.5 sind die Daten, um die es im vorigen Abschnitt ging als Objekte im Applikations-Server zu sehen.

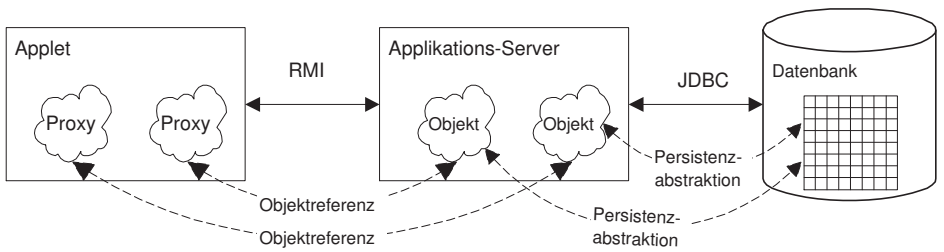


Abbildung 6.5: Objektverteilung

Im Applet, das die Präsentation der Objekte übernimmt, sind nicht die Java-Objekte selbst, sondern Stellvertreter für die Objekte, sog. *Proxys*, repräsentiert. Die Proxys verfügen über alle Attribute der Objekte und stellen alle Methoden der Objekt bereit. Zur Laufzeit verhalten sie sich wie die Objekte selbst, stellen aber nur Objektreferenzen auf entfernte Objekte dar und leiten alle Aufrufe an diese Objekte weiter. Dieses Verhalten wird durch das RMI API, das in Abschnitt 4.3.1 beschrieben wird, bereitgestellt.
Der Applikations-Server enthält die Java-Objekte, auf die die Proxys im Applet verweisen. Auf bzw. mit den Objekten läuft der Anwendungsprozeß ab und über die Persistenzabstraktion zur Datenbank wird die Persistenz der Objekte erreicht. Hier findet sich schon bei den Modellen ein Bruch, der bei der Implementation noch deutlicher wird. Die Komponente, die die Persistenzabstraktion der Objekte auf

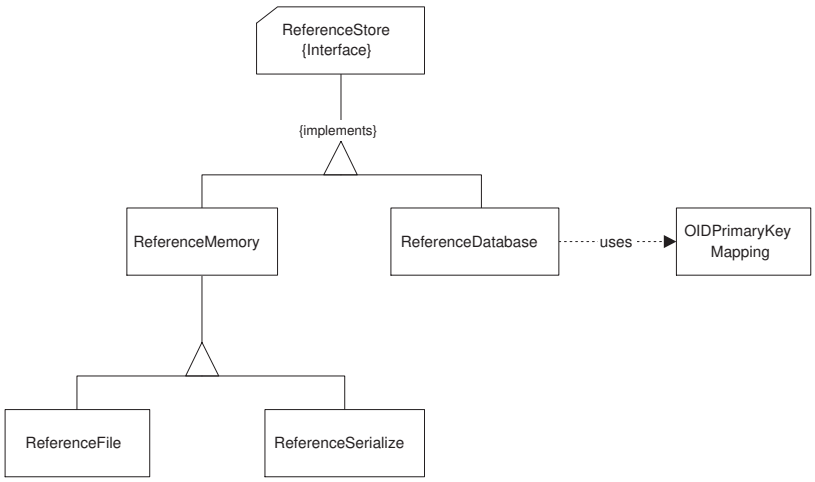


Abbildung 6.6: Hierarchie der Persistenzklassen

Basis einer relationalen Datenbank bietet, muß mit großen Aufwand eine eindeutige Abbildung zwischen den Java-Objekten und den Relationen in der Datenbank leisten. Für die Kommunikation mit der Datenbank wird das JDBC API, das in Abschnitt 4.3.3 vorgestellt wird, benutzt.
Die Datenbank ermöglichst die Persistenz von Daten, die in Relationen abgespeichert werden. Die Java-Objekte werden nach einer Abbildung auf das Relationenmodell durch die Datenbank dauerhaft gemacht. In Abschnitt 4.3.3 wird erklärt, wie diese Abbildung erreicht werden kann.

6.3 Modellierung der Implementation

Für die Implementation von entscheidender Bedeutung ist die Modellierung der Persistenzklassen, d.h. der Klassen die in der Systemarchitektur die Komponente *Persistenzabstraktion* realisieren. Sie bilden eine einheitliche Schnittstelle zur dauerhaften Speicherung der in der Komponente *Repräsentation* dargestellten Daten. Die Speicherung der Daten übernehmen die Komponenten *Datei* und *Datenbank*. Nur ein geeigneter Entwurf der Persistenzabstraktion bietet die Möglichkeit, die Objekte in ganz unterschiedlichen Weisen zu speichern. Einen Überblick über die Hierarchie der Persistenzklassen gibt Abbildung 6.6.
ReferenceStore ist die Schnittstelle, die die Persistenzabstraktion, eine Menge von Methoden zum Erzeugen, Abrufen, Löschen und Verändern von Objekten beinhaltet. In Anhang C.2 ist eine Version der Schnittstelle zu finden, die auf das Wesentliche verkürzt ist. Es bietet nach dem Entwurfsmuster *Facade* [Gamma et al. 95] eine einheitliche Schnittstelle für alle Persistenzklassen, die auf unterschiedliche Arten und Weisen eine Variante der Persistenz der Daten implementieren. Alle Persistenzklassen implementieren diese Schnittstelle, d.h. sie müssen zumindest die in der Schnittstelle definierten Methoden implementieren.
Die Klasse *ReferenceMemory* implementiert die Schittstelle *ReferenceStore* und führt alle Operationen auf den Objekten, die über Methodenaufrufe angestoßen werden, auf Java-Objekten im Hauptspeicher aus.
Die Klasse *ReferenceFile* wiederum ist eine Subklasse von *ReferenceMemory*. Beim Aufruf des Konstruktors liest sie eine BibTeX-Datei ein und erzeugt mit Hilfe von *ReferenceMemory* Java-Objekte im Hauptspeicher. Dann benutzt sie nach dem Entwurfsmuster *Proxy* [Gamma et al. 95] ihre Superklasse *ReferenceMemory*, um die

Operationen auf den Objekten im Hauptspeicher auszuführen. Beim Aufruf des Destruktors werden die Java-Objekte aus dem Hauptspeicher wieder in eine BibTeX-Datei geschrieben.
Die Klasse *ReferenceDatabase* implementiert wie *ReferenceMemory* die Schnittstelle *ReferenceStore*. Sie bietet eine eindeutige Abbildung von den Java-Objektstrukturen auf die Tabellen der Datenbank, wie in Abschnitt 6.2.4 erläutert, und führt alle Operationen auf Objekten der relationalen Datenbank aus. Nach dem Entwurfsmuster *Adapter* [Gamma et al. 95] erledigt *ReferenceDatabase* mit Hilfe der Klasse *OIDPrimaryKeyMapping* die Zuordnung der Objekt-Identifikatoren (OID) der Java-Objekte zu den Primärschlüsseln der entsprechenden Tabellenzeilen in der relationalen Datenbank.
Denkbar wäre noch eine Klasse *ReferenceSerialize*, die beim Aufruf ihres Konstruktors die gespeicherten Objekte in den Hauptspeicher lädt und sich desselben Mechanismus bedient wie *ReferenceFile*. Die Java-Objekte liegen bei dieser Variante durch das API zur Objektserialisierung, beschrieben in Abschnitt 4.3.1, serialisiert in einer Datei vor. Beim Aufruf des Destruktors werden alle Java-Objekte wieder in eine Datei serialisiert. Die Klasse *ReferenceSerialize* wurde im Prototypen nicht implementiert.

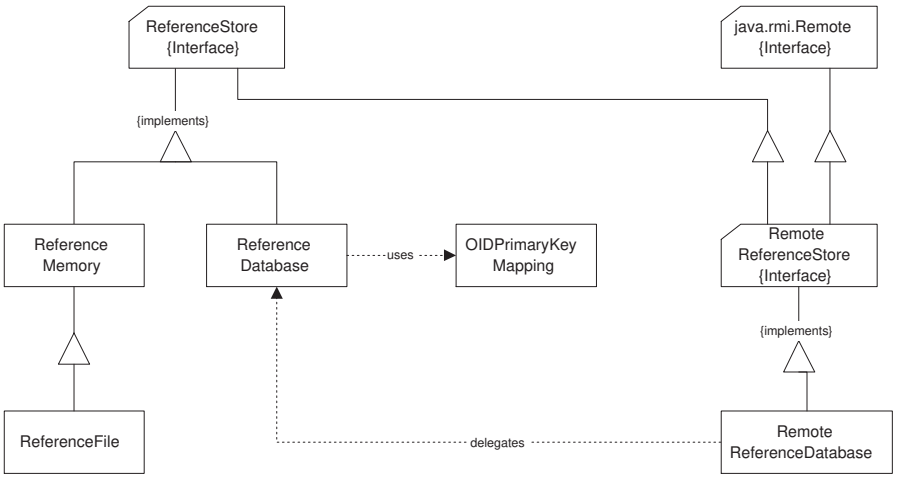


Abbildung 6.7: Erster Ansatz für die Hierarchie der Persistenzklassen

Die in Abbildung 6.6 gezeigte Hierarchie der Persistenzklassen ist als erster Ansatz zu verstehen. Gilt es, die Verteilung der Komponenten zu berücksichtigen, dann ergeben sich durch den Einsatz des RMI API einige Einschränkungen, die eine andere Modellierung der Persistenzklassen erfordern. Damit Objekte verteilt werden können, müssen sie eine Schnittstelle implementieren, die eine Subklasse der Schnittstelle *java.rmi.Remote* ist.
Wie in Abbildung 6.7 zu sehen, wäre es naheliegend, eines Schnittstelle *RemoteReferenceStore* zu definieren, die von der Schnittstelle *ReferenceStore* die Methoden erbt, die zur Persistenzabstraktion nötig sind. Gleichzeitig wäre es Subklasse von der Schnittstelle *java.rmi.Remote*, um die Voraussetzungen zur Verteilung von Objekten per RMI zu erfüllen. Eine Klasse *RemoteReferenceDatabase* implementierte dann diese Schnittstelle *RemoteReferenceStore* und bediente sich seinerseits nach dem Entwurfsmuster *Adapter* [Gamma et al. 95] der Klasse *ReferenceDatabase*, der lokalen Variante, um Persistenz zu erreichen.
Bei dieser Variante wären die zur Verteilung nötigen Klassen und Interface klar herausgestellt und getrennt vom Rest der Klassenhierarchie. Dem entgegen steht,

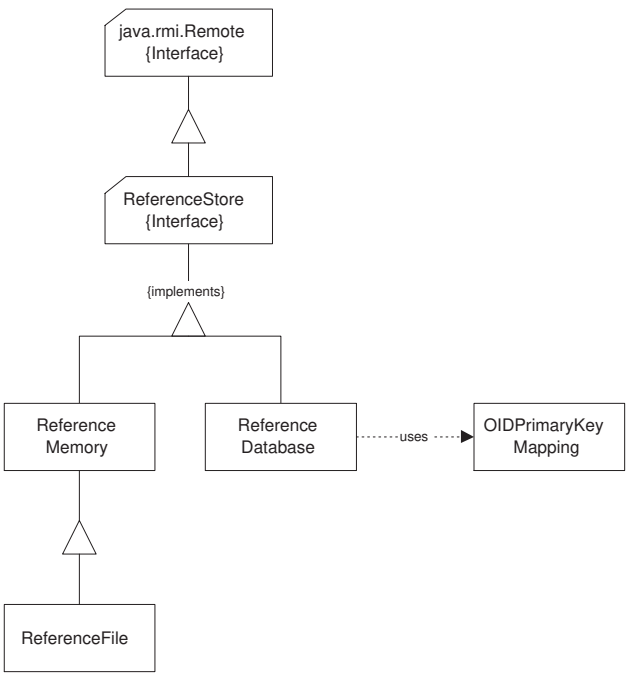


Abbildung 6.8: Realisierte Hierarchie der Persistenzklassen

daß alle Methoden einer Schnittstelle, das *java.rmi.Remote* erweitert, explizit das Auslösen von Ausnahmen der Klasse *java.rmi.RemoteException* deklarieren müssen. Dies erklärt sich daraus, daß *java.rmi.RemoteException* aus unbekanntem Grund *keine* Subklasse von *java.lang.RuntimeException* oder einer der *java.lang.Errors*-Klassen ist und somit deklariert werden *muß*. Der erste Ausweg, die zu implementierende Schnittstelle nur von *java.rmi.Remote* erben zu lassen und alle Methoden von *ReferenceStore* hinüberzukopieren und mit passenden *throws*-Klausel zu versehen, ist wenig elegent und zudem aufwendig. Vertretbarer erscheint die zweite Variante, die Schnittstelle zur Persistenzabstraktion, *ReferenceStore*, von *java.rmi.Remote* erben zu lassen und dort die, für die lokale Version überflüssigen, throws-Klauseln zu deklarieren. Damit sind die Beweggründe für die gewählte Modellierung der Persistenzklassen nach Abbildung 6.8 erklärt.

6.4 Details der Implementierung

Im folgenden Abschnitt sind einige Details der Implementierung beschrieben, die interessant sind, weil sie in ähnlicher Form in anderen Java-Anwendungen nützlich und angebracht sind.

6.4.1 Frame-Klassen

Zur Trennung von Repräsentation der Daten und ihrer Darstellung besitzt jede Datenklasse eine Präsentationsklasse, die die Daten präsentieren kann und ihre Bearbeitung ermöglicht. Beispielweise sind dies die Klasse *Reference*, die Bibliographie-Objekte repräsentiert und die Klasse *ReferenceFrame*, die Objekte der Klasse *Reference* anzeigen und bearbeiten lassen kann. Die Präsentationsklassen sind Subklassen der Klasse *MyFrame*, die von *java.awt.Frame* erbt und einige *constrain()* Methoden mitbringt. Diese erleichtern den Umgang mit dem Gridbag-Layoutmanager *ja-*

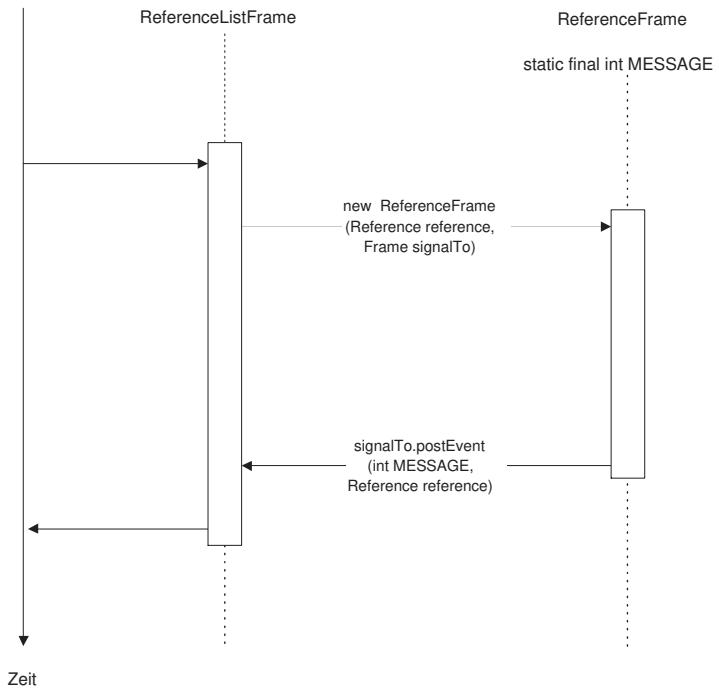


Abbildung 6.9: Kommunikation zwischen Daten- und Präsentationsklassen

va.awt.GridBagLayout, indem sie eine einfache und einheitliche Angabe der Ränder (*java.awt.Insets*) und der Einschränkungen (*java.awt.GridBagConstraints*) für die Elemente der Oberfläche erlauben. Der Quelltext der Klasse *MyFrame* steht in Abbildung C.1.

6.4.2 Ereignisbehandlung

Die Datenklassen und die Präsentationsklassen sind gekapselt gegenüber der jeweils anderen Klasse. Soll ein Objekt einer Datenklasse angezeigt und bearbeitet werden, so wird ein neues Objekt der Präsentationsklasse erzeugt. Mit dem Konstruktor wird dem Präsentationsobjekt das zu bearbeitende Datenobjekt und eine Referenz auf das Objekt, das das Präsentationsobjekt erzeugt, übergeben. Das Präsentationsobjekt handhabt die Anzeige und die Bearbeitung der Daten. Über Nachrichten, die in der Präsentationsklasse festgelegt sind, wird das Objekt, das das Präsentationsobjekt erzeugt hat, über Manipulationen unterrichtet. Mit den Nachrichten wird das veränderte Datenobjekt zurückgegeben. Die Abarbeitung der Abläufe geschieht mit Hilfe der Ereignisbehandlung des AWT. Das aufrufende Objekt behandelt diese Nachrichten wie Ereignisse zur Behandlung der Oberfläche und kann entsprechend der Nachricht Veränderungen der Datenobjekte berücksichtigen.
In Abbildung 6.9 ist dieser Mechanismus am Beispiel der Präsentationsklassen *ReferenceFrame* und der Datenklasse *Reference* als Interaktionsdiagramm zu sehen. Ein Objekt der Klasse *ReferenceListFrame* erzeugt z.B. aufgrund einer Benutzereingabe ein neues Objekt *ReferenceFrame* und übergibt ihm das zu bearbeitende Objekt der Klasse *Reference*. *ReferenceFrame* erzeugt ein neues Fenster und erlaubt das Bearbeiten der Attribute vom *Reference*. Die Bearbeitung geschieht innerhalb von *ReferenceFrame* und unabhängig von *ReferenceListFrame*. Wenn die Bearbeitung beendet ist, sendet *ReferenceFrame* das *Reference* Objekt mitsamt des Status, ob

sie bearbeitet wurde oder nicht, zurück an *ReferenceListFrame*. Diese hat dafür zu sorgen, daß die eventuellen Änderungen in der Anzeige der Liste der *References* berücksichtigt werden.

6.4.3 Layout-Manager

Das Java AWT biete eine Reihe von Layout-Managern, um zur Gestaltung der Oberfläche Komponenten-Objekte (*java.awt.Component*) in Behältern (*java.awt.Container*) zu arrangieren:

BorderLayout arrangiert Komponenten nach den vier Himmelsrichtungen sowie zentriert in der Mitte des Behälters.

CardLayout stellt sicher, daß zur Zeit nur eine aus einer Menge von Komponenten oder Behältern zur Zeit angezeigt wird.

FlowLayout legt Komponenten von links nach rechts aus, solange noch Platz vorhanden ist. Danach wird in die nächste Zeile gewechselt.

GridBagLayout ist der komplizierteste und leistungsfähigste Layout-Manager. Er teilt den Behälter in ein Gitter von Zeilen und Spalten und plaziert die Komponenten in diesem Gitter, wobei die Größe der Gitterzellen angepaßt wird, damit sich die Komponenten nicht überschneiden. Jede Komponenten kontrolliert ihre Positionierung im Gitter durch eine Menge von Restriktionen, die per *GridBagConstraints* und *Insets* festgelegt werden.

GridLayout teilt den Behälter ebenfalls in eine Anzahl von Zeilen und Spalten. Die Komponenten werden von links nach rechts, von oben nach unten in die Zellen des Gitters gelegt. Jede Zelle ist so hoch wie die höchste Zelle in derselben Zeile und so breit wie die breiteste Zelle in derselben Spalte.

6.4.4 GridBagLayout-Manager

Der *GridBagLayout*-Manager bietet nicht nur den Vorteil, daß die Komponenten am Flexibelsten in den Behältern arrangiert werden können. Durch die relative Positionierung in einem Gitter und die Möglichkeit, das Verhalten der Komponenten beim Vergrößern und Verkleinern des Behälters zu steuern, ist der *GridBagLayout*-Manager vor allem beim Oberflächen, die auf unterschiedlichen Plattformen gut aussehen sollen, von Vorteil. Beim Java Developement Kit in der Version 1.02 weichen die Java-Implementationen in Details voneinander ab. Die Abweichungen sind nicht groß, aber groß genug, um bei absoluter Positionierung der Komponenten zu unschönen Ergebnissen zu führen.
Bei CobWeb nutzen sowohl die Import-/Exportkomponente als auch die Verteilungskomponente zur Darstellung ihrer Oberflächenelemente den *GridBagLayout*-Manager. Die Präsentationskomponente nutzt wie die Import-/Exportkomponente eine gemeinsame Basisklasse *ReferenceListFrame*, die einen Rahmen für die beiden Komponenten bildet. Die Klasse *ReferenceListFrame* ist eine Subklasse von *MyFrame* und stellt eine Liste von Literaturreferenzen, drei *Buttons*, um die Einträge der Liste zu manipulieren, sowie ein *TextField* für Informationen über den derzeitigen Zustand zur Verfügung. Subklassen von *ReferenceListFrame* wie die Import-/Export- und die Präsentationskomponente verbinden diese Oberflächenelemente mit eigenen Oberflächenelementen, um zu einer Benutzungsoberfläche zu gelangen. Zur Verwendung dieser Klasse folgt ein Beispiel. Es werden die Überschrift "Gefundene Verweise", das Listenfeld und die Reihe mit den Buttons "Neu", "Ändern" und "Löschen" gestaltet. Dazu wird als Behälter für alle Komponenten ein *Panel*

erzeugt und bestimmt, daß der *GridBagLayout*-Manager das Layout des *Panel* übernimmt. Die Benutzungsoberfläche läßt sich leichter gestalten, wenn man auf Papier eine Einteilung der Oberflächenelemente auf ein Gitter vornimmt. Wird schon dann festgelegt, in welchen Zellen die Oberflächenelemente plaziert werden und welche Oberflächenelemente beim Vergrößern mitwachsen und welche eine feste Größe behalten, geht die Gestaltung der Oberfläche schnell voran. Für das Beispiel sieht der Java Programmkode wie folgt aus:

```
GridBagLayout gridbag = new GridBagLayout();

Panel refPanel = new Panel();
refPanel.setLayout(gridbag);

constrain(refPanel, new Label("Gefundene Verweise:"), 0, 2, 3, 1);
constrain(refPanel, new List(13, false), 0, 3, 3, 1,
          GridBagConstraints.BOTH, GridBagConstraints.CENTER,
          4.0, 1.0, 0, 0, 20, 20);
constrain(refPanel, new Button("Neu"), 0, 4, 1, 1,
          GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTH,
          1.0, 0.0, 0, 0, 0, 0);
constrain(refPanel, new Button("Ändern"), 1, 4, 1, 1,
          GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTH,
          1.0, 0.0, 0, 0, 0, 0);
constrain(refPanel, new Button("Löschen"), 2, 4, 1, 1,
          GridBagConstraints.HORIZONTAL, GridBagConstraints.NORTH,
          1.0, 0.0, 0, 0, 0, 0);
```

Über die *constrain()* Methoden wird das Verhalten der Komponenten festgelegt. Sie kapseln das Anlegen von *GridBagConstraints* und *Insets*, die zur Festlegung des Verhaltens für den *GridBagLayout*-Manager angelegt werden müssen. Die Varianten der Methoden unterscheiden sich darin, inwieweit das Verhalten beeinflußt werden kann. Nicht angegebene Werte werden dann mit Default-Vorgaben belegt.
Die Parameter für die einfachste Methode, die im obigem Beispiel für die Label Komponente gelten, sind wie folgt anzugeben: Der Behälter, in dem die Überschrift erscheinen soll, ist *refPanel*. Dann wird als Referenz auf die Komponente an Ort und Stelle in diesem Fall ein Label-Objekt mit der Überschrift erzeugt. Die Komponente soll in der Zelle erscheinen, die in Spalte 0, Zeile 2 im Gitter beginnt. Die Zelle ist drei Zellen breit und eine Zelle hoch. Die Breite der Zelle beträgt drei, da sie so breit sein soll, wie die später definierten drei Buttons zusammen.
Für die komplizierteste Methode, in dem Beispiel für die Liste benutzt, kommen zu den Objekt-Referenzen auf Behälter und Komponente, sowie der Position und Größe der Komponente noch weitere Parameter hinzu: Die Komponente soll sich, wenn die Zelle größer wird, in beide Richtung ausdehnen (*GridBagConstraints.BOTH*) und zentriert in der Zelle ausgerichtet werden (*GridBagConstraints.CENTER*). Die Ausdehung der Komponente ist in horizontaler Richtung mit 4.0 und in vertikaler Richtung mit 1.0. Damit wird angegeben, wie das *List*-Objekt, das sich in der Zelle befindet, beim Vergrößern des Anzeigebereichs Platz in horizontaler und vertikaler Richtung an andere Zellen abgibt. Die Gewichtung der Ausdehung ist immer im Verhältnis zu den anderen Zellen zu sehen. Für die horizontale Richtung bedeutet dies, daß die Zelle mit dem *List*-Objekt beim Vergrößern in horizontaler Richtung viermal mehr Platz erhält als die anderen Zellen, die mit dem Faktor 1.0 aber immer noch am Verteilen des Platzes beteiligt werden. Bei einer vertikalen Vergrößerung des Anzeigebereichs bekommt die Zelle mit dem *List*-Objekt sogar jeden neu hinzukommenden Platz und muß beim Verkleinern den freizugebenden Platz abgeben,

da die anderen Komponenten mit Faktor 0.0 sich nicht vergrößern oder verkleinern dürfen. Die letzten vier Parameter bestimmen für die einzelnen Seiten den Platz, der um die Komponente, das *List*-Objekt, in der Zelle freibleiben soll.

6.4.5 Filterung, Matching

Der Im- und Export von Literaturreferenzen gehört zu den konkreteren Anforderungen, die in Kapitel 3 aufgestellt wurden. Der Export von Literaturreferenzen ist problemlos, da lediglich das interne Datenmodell auf das BibTeX Datenformat abzubilden ist.
Der Import hat eine Doppelfunktion. Zum einen soll der Abgleich und das Zusammenführen separat gewachsener Bestände möglich sein. Zum anderen sollen aus der Arbeitsumgebung exportierte und autonom bearbeitete Bestände mit dem gemeinsamen Bestand an Literaturreferenzen abgeglichen werden, um die Änderungen am autonom bearbeiteten Bestand in den gemeinsamen Bestand einzupflegen.
Um einen Abgleich von Beständen durchführen zu können, muß ein geeigneter Algorithmus gefunden werden, um für zwei Literaturreferenzen entscheiden zu können, ob sie gleich oder ähnlich sind. Ein Abgleich von zwei Objekten der Klasse *Reference* anhand der Objekt-Identifikatoren (OID) und der Objekte, zu denen Assoziationen bestehen, ist nicht anwendbar. Vielmehr muß für jeweils zwei Objekte einer Klasse entschieden werden, ob sie gleiche oder ähnliche Werte der Attribute haben und damit als gleich gelten. Zu diesem Zweck haben die Klasse Reference und ihre Subklassen sowie alle Klassen, zu denen Beziehungen von diesen Klassen ausgehen, einen Methode *matches()*, die dies entscheidet. Wenn für zwei Literaturreferenzen, die durch Objekte der Klasse Reference oder einer Subklasse von Reference repräsentiert werden, entschieden werden soll, ob die Literaturreferenz gleich ist, werden die Objekte und dann rekursiv alle eingebetteten Objekte überprüft.
In der derzeitigen Implementation wird dieser Mechanismus von den Klassen genutzt, die das Interface ReferenceStore implementieren. Die Methoden der Objekte vergleichen die Attribute auf Gleichheit und vergleichen anschließend die eingebetteten Objekte. Als Voraussetzung für einen effektiven Abgleich müßte der Vergleich nicht auf vollständige Übereinstimmung aller Attribute überprüfen, sondern erweitert werden, um zu erreichen, daß die weitgehende, eindeutige Ähnlichkeit als identisch erkannt wird. Entsprechende Algorithmen wurden bereits erfolgreich angewandt [French et al. 97]. Eine solche Lösung wäre noch flexibler, wenn man die Algorithmen zum Filtern und für das Matching nach dem Entwurfsmuster *Strategy* [Gamma et al. 95] aus den Datenklassen herausfaktorisierte.

Kapitel 7

Zusammenfassung und Ausblick

Diese Arbeit befaßte sich mit einer Arbeitsumgebung für die werkzeuggestützte, kooperative Arbeit mit Literaturreferenzen. Zunächst wurden die Methoden und Werkzeuge zur Literaturverwaltung vorgestellt, die derzeit zur Arbeit mit Literaturreferenzen eingesetzt werden. Danach wurden charakteristische Arbeitsszenarien mit besonderem Hinblick auf den Aspekt der Kooperation erarbeitet. Diese Arbeitsszenarien waren die Grundlage für einen Katalog von Anforderungen an eine Arbeitsumgebung für die kooperative Arbeit mit Literaturreferenzen. Als Grundlage für die folgenden Kapitel wurden die verwendeten Komponenten, Werkzeuge und Sprachen beschrieben. Es wurden Modelle für die Verwaltung von Literaturreferenzen entworfen, wobei auf die Erstellung geeigneter Datenmodelle der größte Anteil entfiel. Zur Evaluation der entworfenen Modelle wurde eine prototypische Anwendung entwickelt. Die Erfahrungen, Erkenntnisse und Schwierigkeiten dabei bildeten den letzten Teil der Arbeit.
Diese Kapitel geht näher auf die Erfahrungen beim Bearbeiten des Themas ein, bewertet die eingesetzten Werkzeuge und gibt einen Ausblick auf mögliche Erweiterungen und Ansätze für weitere Arbeiten.

7.1 Zusammenfassung

Für die Erstellung dieser Arbeit war die Beschäftigung mit den Grundlagen von besonderer Bedeutung, da Arbeitsumgebungen für die werkzeuggestützte, kooperative Arbeit mit Literaturreferenzen häufig in einem Umfeld eingesetzt werden, in der seit längerer Zeit mit Literaturreferenzen gearbeitet wurde. Die existierenden Bestände und Arbeitsweisen müssen bei der Einführung eines neuen Werkzeuges berücksichtigt werden.
Die Betrachtung der existierenden Methoden und Werkzeuge zur Literaturverwaltung und die Arbeit damit war der Ausgangspunkt für diese Arbeit. Die relativ aufwendige Erstellung der Arbeitsszenarien, die in einem Katalog von Anforderungen münden, war der geeignete Ansatz für die Modellierung. Die Anforderungen konnten so frühzeitig berücksichtigt und systematisch in die Modelle integriert werden. Zur Evaluation wurde ein Prototyp erstellt, dessen Modellierung und Aufteilung der Komponenten ebenso von den Anforderungen profitieren konnten. Es zeigte sich, daß vor allem das Objektmodell, das die Basis für die anderen Modelle darstellt und dem deshalb ein besonderer Augenmerk galt, während der Implementation nahezu unverändert blieb.
Damit bietet sich der Prototyp, der als vertikaler Prototyp realisiert wurde, als eine

geeignete Basis für die Implementation einer einsetzbaren Lösung an. Die wesentlichen Entwurfsentscheidungen auf dem unterschiedlichen Entwurfsebenen wurden getroffen, so daß für den Ausbau zu einer für Benutzer effektiven Arbeitsumgebung "lediglich" das Ausprogrammieren weiterer Funktionalität notwendig ist.
Besondere Aufmerksamkeit sollte bei dem angedachten Ausbau die Relationen zwischen dem Aufwand für die grafische Benutzeroberfläche, der Behandlung von Ereignissen an der Benutzeroberfläche und der Persistenz der Daten gelten. Bei dem in Java realisierten Prototypen entfiel mit ungefähr zwei Dritteln des Programmkodes der Großteil auf die Realisierung der Persistenz der Daten. Von dem verbleibenden Drittel beschäftigten sich wieder ungefähr zwei Drittel des Programmkodes mit der Ausprogrammierung der Elemente für die grafische Benutzeroberfläche. Der Rest des Programmkodes behandelt die Ereignisse der Benutzeroberfläche und ruft die Methoden für die Persistenz der Daten an.
Die Erstellung der Benutzeroberfläche erfolgte beim Prototypen noch per Hand. Das Ausprogrammieren könnte durch spezielle Editoren für die Gestaltung von Benutzeroberflächen, wie sie in den neuesten Java-Entwicklungsumgebungen wie z.B. IBMs *VisualAge for Java* enthalten sind, nahezu entfallen. Es verbleibt als größter Teil der Programmkode zur Persistenz der Daten. Der Aufwand dafür läßt sich, da zum einen eine Transformation vom objektorientieren ins relationale Modell nötig ist und zum anderen Java keine geeigneten Abstraktionen und Konzepte für diese Transformation mitbringt, prinzipiell nicht durch die geeignete Modellierung der Klassen reduzieren. Die Zahl der Klassen zur Unterstützung der Persistenzabstraktion wächst mit jeder Klasse und jeder Assoziation zwischen Klassen um eins, so daß die Komplexität der Persistenzabstraktion bei komplexen Objektstrukturen zum entscheidenden Hindernis für die Implementation einer einsetzbaren Lösung werden wird.
Sollte der in dieser Arbeit vorgestellte Prototyp CobWeb zu einer einsetzbaren Lösung ausgebaut werden, ergibt sich eine Liste mit Funktionalitäten, die für die Effektivität einer Arbeitsumgebung notwendig sind, beim Prototypen jedoch nicht berücksichtigt wurden. Der Prototyp CobWeb wäre noch hinsichtlich der folgenden Aspekte erweiterbar:

kompletter Umfang an Literaturreferenztypen Bisher ist von dem in Abschnitt 5.2 vorgestellten Objektmodell für die Repräsentation der Literaturreferenzen nur eine Untermenge realisiert. Diese ist auf den kompletten Umfang an Literaturreferenztypen auszudehnen.

Benutzerverwaltung Obwohl in der Modellierung vorgesehen, ist beim Prototypen CobWeb keine Verwaltung der Benutzer realisiert.

Personalisierung Dadurch, daß in dem Prototypen keine Benutzerverwaltung implementiert wurde, ist die Implementation der Komponenten zur Personalisierung überflüssig und wurde daher nicht realisiert.

fremde BibTeX-Attribute importieren Der im Prototypen realisierte Import ignoriert alle Attribute, die nicht denen in Anhang A.2 entsprechen. Es ist eine geeignete Lösung zu finden, wie "fremde" BibTeX-Attribute, also Attribute, die BibTeX ignoriert aber die für den Benutzer bedeutsam sind, übernommen werden können und beim Export wieder zur Verfügung stehen.

Unterstützung für kooperative Arbeit Ohne die Benutzerverwaltung und die Personalisierung ist die Unterstützung kooperativer Arbeit nicht zu realisierung und wurde deshalb nicht implementiert.

7.2 Bewertung der eingesetzten Werkzeuge

Die für die Modellierung eingesetzte Object Modeling Technique (s. Abschnitt 4.1.1) diente zur Visualisierung der statischen, dynamischen und funktionalen Struktur des Prototypen. Sie war die Grundlage für das Verständnis der Materie und die Modellierung geeigneter Objektmodelle, auf die hier besonderen Wert gelegt wurde. Da OMT eine objektorientierte Methode ist und mit Java eine objektorientierte Programmiersprache eingesetzt wurde, gab es keinen Bruch zwischen den Phasen Entwurf und Implementation. Insgesamt erschien die OMT-Methode geeignet für diese Arbeit.
Im Gegensatz dazu ist der Einsatz der Softwareentwicklungsumgebung Software through Pictures (s. Abschnitt 4.1.2) in dieser Arbeit kritisch zu beurteilen. Der Einarbeitungsaufwand und der tatsächliche Nutzen standen in einem ungünstigen Verhältnis zueinander. Für das Verständnis des Themenkomplexes der Literaturreferenzen und den Entwurf der Modelle hätten auch auf Papier durchgeführt werden können. Die mit StP erstellten Modelle konnten zudem nicht direkt in diese Arbeit übernommen werden, da die Qualität der exportieren Zeichnungen unbefriedigend war. Deshalb mußten alle Modelle mit *Visio*, das zum Erstellen von Diagrammen ohne weitere Semantik deutlich besser als StP geeignet ist, erneut gezeichnet werden. In größeren Projekten, die so komplex sind, daß mehrere Entwickler daran arbeiten und alle vier Phasen der OMT-Methode komplett durchlaufen werden, kann StP sicherlich nützlich sein. Besonders die automatische Erstellung und Kommentierung von Quelltexten in der Phase der Implementation ist dann ein wesentliches Argument für den Einsatz von StP.
Die Programmiersprache Java (s. Abschnitt 4.2.1) konnte bei der Implementation des Prototypen überzeugen. Die Syntax ist sehr leicht zu erlernen, wenn Vorkenntnisse in den Programmiersprachen C und C++ bestehen. Besonders im Vergleich zu den genannten Programmiersprachen fallen die Prinzipien Einfachheit und Robustheit der erstellten Programme auf. Ein Java-Programm, das vom Compiler anstandslos übersetzt wurde, stürzt nicht ab. Während des Testens des Prototypen war zu bemerken, daß typischerweise lediglich zwei Typen von Laufzeitfehlern auftreten: Im Fall der *NullPointerException* wurde die korrekte Initialisierung einer Referenz durch den Programmierer vergessen. Es handelt sich also um einen Programmierfehler. Bei der *SQLException* besteht keine Netzanbindung und es kann deshalb keine Verbindung zur Datenbank aufgebaut werden. Natürlich lassen sich beide Exceptions abfangen und führen nicht zum Abbruch des Programms.
Die Plattformunabhängigkeit von Java erwies sich als wesentlicher Vorzug, da die Programmierung an drei unterschiedlichen Orten mit heterogenen Rechnerumgebungen stattfand und dennoch keine Rücksicht auf Unterschiede der Plattformen genommen werden mußte. Die Geschwindigkeit von Java-Programmen ist für interaktive, benutzergesteuerte Anwendungen vollkommen ausreichend. Engpässe ergeben sich hier höchstens bei besonders rechenintensiven Anwendungen, zu denen leider auch der Java-Compiler gehört. Hier könnte eine native Anwendung die Wartezeiten für den Programmierer deutlich reduzieren.
In dieser Arbeit wurde Java für die Client-Server Programmierung eingesetzt. Auch hier konnten die beteiligten Komponenten überzeugen, da sie trotz teilweise recht früher Entwicklungsstadien von erstaunlicher Reife waren. Dies konnte von den eingesetzten Treibern für das Ansprechen der Datenbank nicht behauptet werden. Die Installation und Konfiguration der Treiber und das anschließende Testen bis eine vorhandene Datenbank angesprochen werden konnte, benötigte einen ganzen Tag.
Obwohl Java also den selbstgestellten Anforderungen (s. Abschnitt 4.2.1) bei der praktischen Anwendung durchaus gerecht wird, bleiben zwei Punkte kritisch anzumerken. Erstens bringen beim Zugriff auf relationale Datenbanken die konzeptbe-

dingten Differenzen zwischen dem relationalen und dem objektorientierten Modell einen erheblichen Aufwand bei der Implementation mit sich. Dieser ist jedoch nicht Java selbst, sondern dem Fehlen einer überzeugenden Alternative und dem verbreiteten Einsatz relationaler Datenbanken, für die Schnittstellen zu Java geschaffen werden müssen, zuzuschreiben. Wenn eine Datenbank neu erstellt wird, könnte eine objektorientierte Datenbank oder gar eine persistente Version von Java, einen Prototypen *PJava* [Atkinson et al. 96] gibt es bereits, den allseits gerühmten "Ease-of-use" von Java auch in diesem Bereich bieten. Der zweite Punkt hat indirekt mit dem ersten zu tun. Für die typsichere Übergabe von Mengen von Objekten wie sie bei der Persistenzabstraktion des Prototypen auftrat, fehlt Java das Konzept des parametrischen Polymorphismus. Hier existiert bereits ein Prototyp namens *Pizza* [Odersky, Wadler 97], mit dem der Aufwand durch die Typisierung von Klassen deutlich reduziert werden könnte.

7.3 Ausblick

Die in dieser Arbeit vorgestellte Modellierung stellt eine solide Grundlage für einen Arbeitsumgebung dar, die die kooperative Arbeit mit Literaturreferenzen mit einem Werkzeug unterstützen soll. Es sind eine Reihe von Erweiterungen denkbar, die die Effektivität einer solchen Arbeitsumgebung für den Benutzer noch weiter steigern können. Sie wurden hier nicht berücksichtigt, weil der Arbeitsaufwand für die Modellierung den Umfang der Arbeit überstiegen hätte. Die Modellierung wäre noch hinsichtlich der folgenden Aspekte erweiterbar:

Versionierung Durch das Anlegen von Versionen für alle Objekte, also die Literaturreferenzen und die mit ihnen assozierten Objekte, stände dem Benutzer die komplette Historie der Literaturreferenz zur Verfügung.

Strategien für den Zugriff auf die Kommentare Wenn die Arbeitsumgebung von mehr Benutzern als in einer überschaubar kleinen Arbeitsgruppe, die sich persönlich kennt und zusammen arbeitet, benutzt wird, werden Strategien für den Zugriff auf die Kommentare notwendig, um die Privatsphäre des Einzelnen zu schützen. Der Schutz könnte in der Vergabe von Rechten für das Lesen und Verändern bestehen.

strukturierte Kommentare Bisher sind die Kommentare strukturlos und nicht dazu geeignet, die Recherche nach mehr als Stichwörtern zu unterstützen. Eine Strukturierung der Kommentare bietet den Vorteil, nach Kriterien, die die Benutzer bestimmen können, differenzieren zu können und den Gehalt an Information für die Recherche besser aufzubereiten. Eine Strukturierung könnte z.B. die Arbeiten, für die die referenzierte Literatur geeignet ist, den Standort der Literatur in der Bibliothek oder die Eignung als Quelle für Zitate aus bestimmten Themengebieten umfassen.

physische Ausprägung, Format, Ansprechpartner, Standort Zu den weiteren für die Recherche der Benutzer interessanten Daten gehören das Format von elektronischen Dokumenten (z.B. Postscript, TeX- oder Word-Dokument) und die physische Ausprägung (z.B. WWW-Dokument, CD-ROM, Papier, Buch oder Paperback) einer Literatur. Sie enthalten Informationen für den Benutzer, wie er eine Literatur erhalten kann. In diesselbe Kategorie fallen Informationen über Ansprechpartner wie z.B. der Autor oder frühere Leser und den Standort, denn sie ermöglichen, die Literatur zu beziehen und mit Lesern der Literatur in Kontakt zu treten. Diese Daten könnte in einer erweiterten Version mitgespeichert werden.

Referenzen zwischen Literaturreferenzen Es ist keine Möglichkeit, Referenzen zwischen, d.h. zu anderen, Referenzen zu realisieren. Diese könnten genutzt werden, um eine Reihe von Literaturreferenzen, die eng zusammengehören, im Objektmodell entsprechend zu modellieren. In diesem Zusammenhang ist auch eine Aggregation einer neuen Klasse *Journal* über *Article* und von *Series* über *Book* denkbar, um eine realitätsgetreuere Modellierung zu erhalten.

Zusammenführung von ähnlichen Attributen Bislang werden Literaturreferenzen von der Import-Komponente als identisch eingestuft, wenn sie in den erforderlichen Attributen übereinstimmen. Durch das Anwachsen der Menge an Literaturreferenzen und den zunehmenden Austausch von Informationen bekommen Mechanismen zum automatischen Zusammenführung von Zeichenketten zunehmend Bedeutung [Siegfried, Bernstein 91]. In [French et al. 97] ist beschrieben, wie anhand einer großen Bibliographie-Datenbank aus dem Astronomie-Bereich Algorithmen für die Zusammenführung von Zeichenketten entwickelt und ihre Wirksamkeit, Varianten zu reduzieren analysiert wurde. In der ersten Phase wird dazu eine lexikalische Bereinigung der Daten vorgenommen. Häufig vorkommende Abkürzungen und empirisch ermittelte Bestandteile der Attribute werden vereinheitlicht. Für die Definition der Abkürzungen und der zu eliminierenden Bestandteile ist der Eingriff eines fachkundigen Benutzers erforderlich. In einem zweiten Schritt werden über die Ermittlung der Bearbeitungsdistanz, die die Anzahl der Bearbeitungsschritte angibt, die nötig sind, um eine Zeichenkette in eine andere zu transformieren, Äquivalenzklassen gebildet und so die Anzahl der Varianten weiter reduziert. Die Wahl des Schwellwertes, ab welcher Bearbeitungsdistanz zwei Wörter in dieselbe Äquivalenzklasse fallen, ist ausschlaggebend für die Wirksamkeit und das korrekte Ergebnis dieses Schrittes. In einem letzten Schritt werden die Wörter der äquivalenten Zeichenketten isoliert, danach sortiert und Varianten mit übereinstimmenden Wörtern in dieselbe Äquivalenzklasse eingeordnet.

Information Retrieval auf Abstracts Ab einer kritischen Menge an Abstracts zu den Literaturreferenzen lohnt es sich, die Methoden des Information Retrieval bei der Recherche auf Abstracts einzusetzen, um die Qualität der Recherche zu verbessern.

Die Komponenten, die in dieser Arbeit zur Implemenation verwandt wurden, sind im Hinblick sowohl auf die zur Zeit der Implementation auf dem Markt befindlichen Komponenten als auch auf die Beschaffungsmöglichkeiten gewählt worden. Eine objektorientierte Datenbank mit einer Schnittstelle zu Java könnte den Aufwand für die Persistenz erheblich reduzieren. Idealerweise könnte eine Version von Java eingesetzt werden, die die Vorteile, die durch den Einsatz einer Datenbank entstehen, mit der direkten Integration in die Programmiersprache kombinieren. Ein Prototyp einer Java-Laufzeitumgebung, die auf Persistenz durch transitive Erreichbarkeit beruht und keinerlei Änderungen an der Sprachsyntax vornimmt, ist mit *PJava* [Atkinson et al. 96] entwickelt worden.
Mit der prognostizierten Verbreitung von CORBA Lösungen drängt sich eine Implementation der verteilten Komponenten nicht mit RMI, sondern mit Java IDL [Sun Microsystems Inc. 97b], einem API zur Interface Definition Language (IDL) von CORBA, auf. Damit wäre auch für Clients, die nicht aus der Java-Welt stammen, der Weg zur kooperativen Arbeit mit einer Arbeitsumgebung für Literaturreferenzen frei.

Anhang A

BibTeX-Definition

A.1 Typen von Literaturreferenzen in BibTeX

Dies ist die Verfeinerung der Literaturreferenztypen, die zunächst die Typen und dann die erforderlichen und die optionalen Attribute der Typen definiert. Für alle Typen von Literaturreferenzen stehen zusätzlich die ignorierten Attribute `owner`, `reg-date`, `change-date`, `abstract`, `keywords`, `directory` und `comment` zur Verfügung. Diese enthalten einen Teil der zusätzlich in der Arbeitsumgebung enthaltenen Information zu den Literaturreferenzen. Damit stehen diese Informationen auch nach dem Export ins BibTeX-Format zur Verfügung.

article ein Artikel aus einem Journal oder einem Magazin; erforderliche Attribute: `author`, `title`, `journal`, `year`; optionale Attribute: `volume`, `number`, `pages`, `month`, `note`

book ein Buch, bei dem der Verleger oder Herausgeber explizit genannt ist; erforderliche Attribute: `author` oder `editor`, `title`, `publisher`, `year`; optionale Attribute: `volume`, `series`, `address`, `edition`, `month`, `note`

inbook ein Teil eines Buches, beschrieben durch ein Kapitel, einen Abschnitt und/oder einen Bereich von Seiten; erforderliche Attribute: `author` oder `editor`, `title`, `chapter` und/oder `pages`, `publisher`, `year`; optionale Attribute: `volume`, `series`, `address`, `editon`, `month`, `note`

booklet eine gedruckte und gebundene Arbeit, bei der Verleger oder Herausgeber oder die unterstützende Institution nicht genannt werden; erforderliche Attribute: `title`; optionale Attribute: `author`, `howpublished`, `address`, `month`, `year`, `note`

collection ein Sammelband wissenschaftlicher Arbeiten, der eigene Titel für die Kapitel hat, jedoch kein Konferenzband (nicht in der ursprünglichen BibTeX-Definition enthalten); erforderliche Attribute: `title`, `year`, `publisher`; optionale Attribute: `editor`, `organization`, `address`, `month`, `note`

incollection ein Teil eines Sammelbandes, der einen eigenen Titel hat; erforderliche Attribute: `author`, `title`, `booktitle`, `publisher`, `year`; optionale Attribute: `editor`, `chapter`, `pages`, `address`, `month`, `note`

proceedings ein Konferenzband mit einer Sammlung von Ergebnissen und Inhalten einer Konferenz; erforderliche Attribute: `title`, `year`; optionale Attribute: `editor`, `publisher`, `organization`, `address`, `month`, `note`

inproceedings ein Artikel in einem Konferenzband; erforderliche Attribute: `author`, `title`, `booktitle`, `year`; optionale Attribute: `editor`, `pages`, `organization`, `publisher`, `address`, `month`, `note`

conference ist dasselbe wie proceedings und zur Kompatibilität mit dem Scribe-System [Digital Equipment Corp. 96] vorhanden

mastersthesis eine Arbeit zur Erlangung eines *master degree*, im deutschen Sprachraum entsprechend einer Diplomarbeit; erforderliche Attribute: `author`, `title`, `school`, `year`; optionale Attribute: `address`, `month`, `note`

phdthesis eine Arbeit zur Erlangung eines *physical degree*, im deutschen Sprachraum entsprechend einer Doktorarbeit; erforderliche Attribute: `author`, `title`, `school`, `year`; optionale Attribute: `address`, `month`, `note`

manual eine technische Dokumentation; erforderliche Attribute: `title`; optionale Attribute: `author`, `organization`, `address`, `edition`, `month`, `year`, `note`

techreport ein Bericht, der von einer Bildungseinrichtung oder einer Institution veröffentlicht wird und üblicherweise eine Nummer im Rahmen einer Serie erhält; erforderliche Attribute: `author`, `title`, `institution`, `year`; optionale Attribute: `type`, `number`, `address`, `month`, `note`

unpublished ein Dokument mit Titel und Autoren, das jedoch formell nicht veröffentlicht ist; erforderliche Attribute: `author`, `title`, `note`; optionale Attribute: `month`, `year`

misc wenn kein anderer Typ paßt; erforderliche Attribute: keine; optionale Attribute: `author`, `title`, `howpublished`, `month`, `year`, `note`

A.2 Attribute von Literaturreferenzen in BibTeX

Dies ist die Verfeinerung der Attribute der Literaturreferenzen. Nach den von BibTeX als erforderlich und optional unterstützten Attributen folgen einige von BibTeX ignorierte Attribute, die für CobWeb benötigt werden.

key wird benutzt für eine alphabetische Ordnung, wenn `author` und `title` fehlen; dieses Feld enthält *nicht* den Schlüssel, der am Beginn einer Literaturreferenzen steht und zum Zitieren benutzt wird.

author der oder die Namen des oder der Autoren; Autoren werden per "and" getrennt. Stehen zwischen dem Vor- und dem Nachnamen eines Autoren ein oder zwei Kommata, so wird erst der Nachname, ein Komma, evtl. eine Namenserweiterung wie "Junior", ein zweites Komma, wenn eine Namenserweiterung angegeben wurde, sowie der oder die Vornamen benannt. Werden Kommata oder mehrere Namen in geschwungenen Klammern geschrieben, so wird der Inhalt innerhalb der Klammern wie ein Wort betrachtet.

editor der oder die Namen des oder der Herausgeber; gibt es auch ein *author* Feld, dann ist der Herausgeber des Buches oder der Sammlung angegeben.

publisher der Verlages, die herausgebenden Körperschaft oder der Herausgeber

title der Titel einer Arbeit; wenn aus einem Sammelband zitiert wird, dann der eigene Titel des Abschnitts, aus dem zitiert wird

booktitle der Titel eines Sammelbandes, aus dem ein Teil zitiert wird

journal der Name eines Journals oder Magazins

chapter eine Kapitelnummer

pages eine oder mehreren Seitenzahlen oder Bereiche von Seiten

number die Nummer eines Journals, Magazins oder technischen Reports; eine Ausgabe eines Journals oder Magazins wird üblicherweise über den Band und die Nummer beschrieben; die Organisation, die einen technischen Report ausgibt, vergibt normalerweise eine Nummer.

edition die Ausgabe eines Buches

series der Name einer Serie oder einer Menge von Büchern; wenn ein ganzes Buch zitiert wird, gibt das `title` Feld den Titel und das optionale `series` Feld den Namen einer Serie, in der das Buch veröffentlicht wurde, an.

volume der Band eines Journals oder eines mehrbändigen Buches

type der Typ eines technischen Reports

organization die Organisation, die eine Konferenz sponsort

institution die Institution, die die Arbeit veröffentlicht hat

school die Bildungseinrichtung, an der die Arbeit geschrieben wurde

address die Adresse einer Körperschaft

month der Monat, in dem eine Arbeit veröffentlicht wurde; bei unveröffentlichten Arbeiten der Monat, in dem die Arbeit vollendet wurde

year das Jahr der Veröffentlichung; bei unveröffentlichten Arbeiten das Jahr, in dem die Arbeit vollendet wurde

howpublished gibt an, wie etwas auf außergewöhnliche Art und Weise veröffentlicht wurde.

annote ein Kommentar; wird üblicherweise nicht in das Literaturverzeichnis mit aufgenommen, kann aber für die Kommentierung des Literaturverzeichnisses genutzt werden

note zusätzliche Informationen, die dem Benutzer helfen können

Die folgenden Attribute sind nicht in der ursprünglichen BibTeX-Definition enthalten. Es sind von BibTeX ignorierte Attribute, die einen Teil der zusätzlich in der Arbeitsumgebung vorhandenen Informationen zu den Literaturreferenzen enthalten, damit diese Informationen auch nach dem Export ins BibTeX-Format zur Verfügung stehen.

owner der Name der Person, die die Literaturreferenz erstellt hat

owner-key der Schlüssel, den die Person, die die Literaturreferenz erstellt hat, benutzt

reg-date das Datum, an dem die Literaturreferenz erstellt wurde

change-date das Datum, an dem die Literaturreferenz zuletzt verändert wurde

abstract das Abstract, die kurze Zusammenfassung des Inhalts

keywords eine Reihe von Schlüsselworten, die der Literatur zugeordnet werden und den Inhalt der Literatur einordnen

directory der Universal Ressource Locator (URL), unter der die Literatur zu finden ist

comment ein Kommmentar, der der Person, die die Literaturreferenz erstellt hat, hilft, die Literatur einzuordnen

Anhang B

OMT-Diagramme

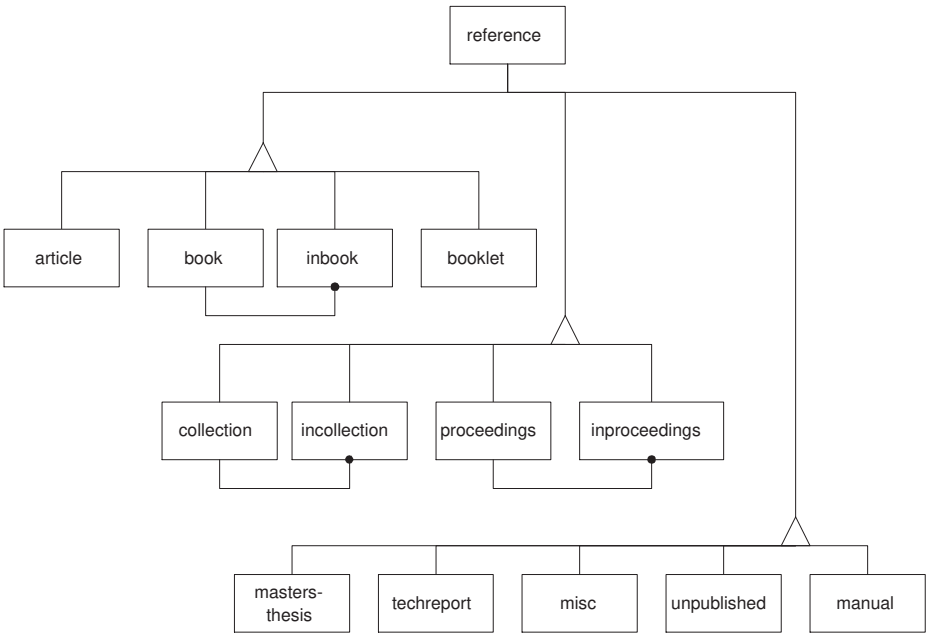


Abbildung B.1: Klassenhierarchie

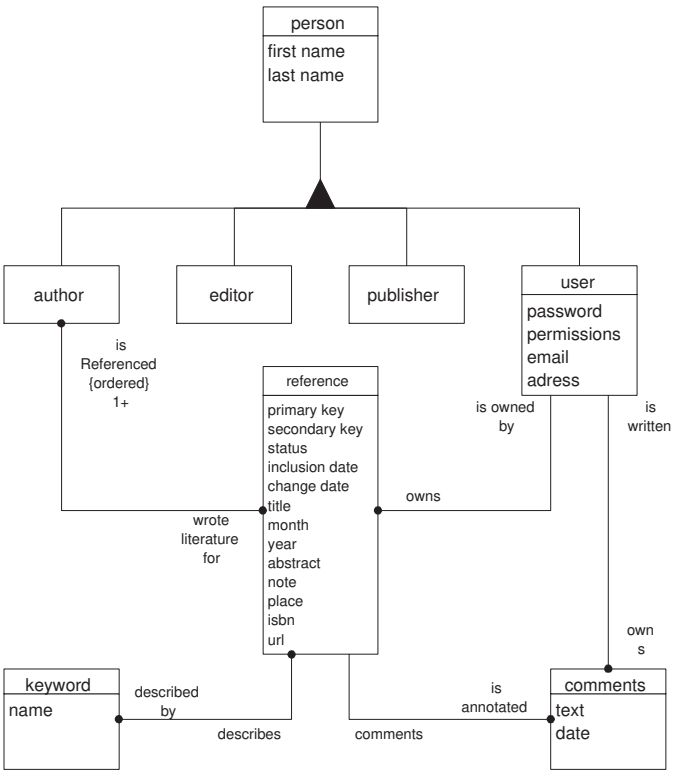


Abbildung B.2: Klasse reference

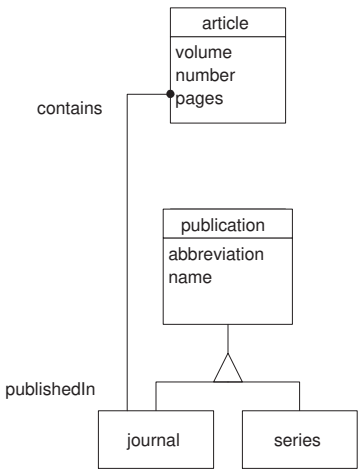


Abbildung B.3: Klasse article

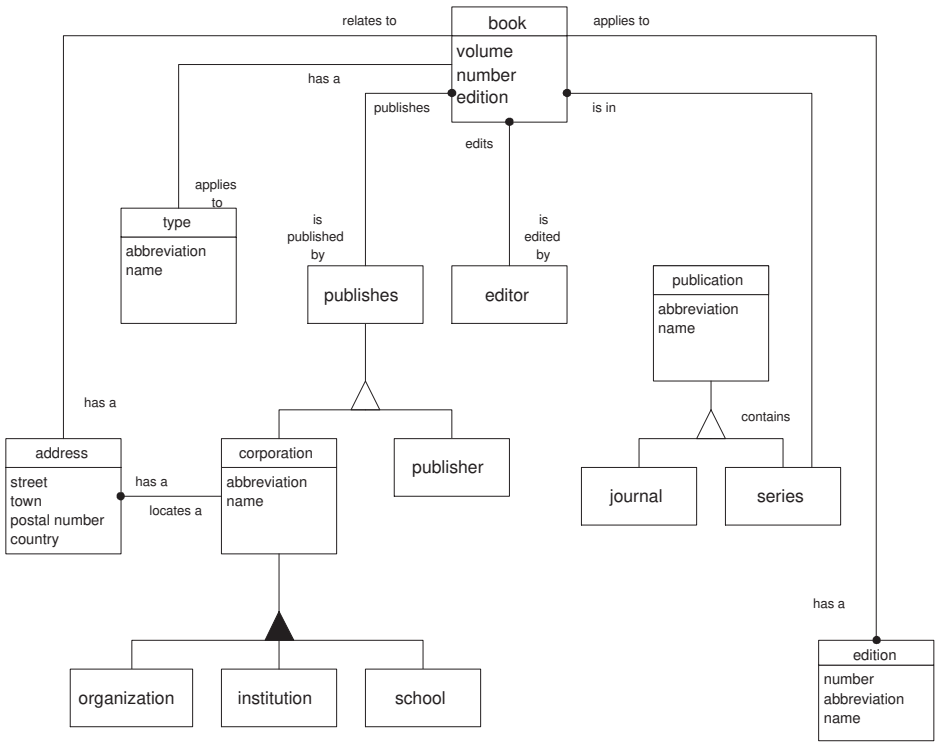


Abbildung B.4: Klasse book

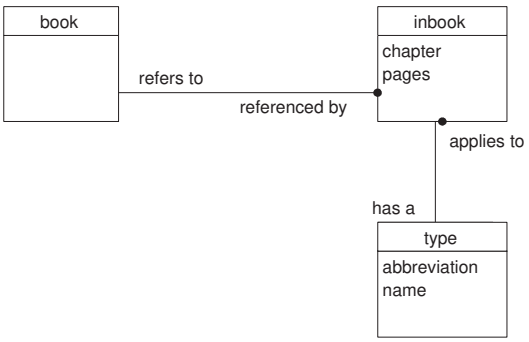


Abbildung B.5: Klasse inbook

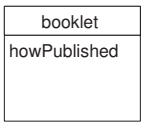


Abbildung B.6: Klasse booklet

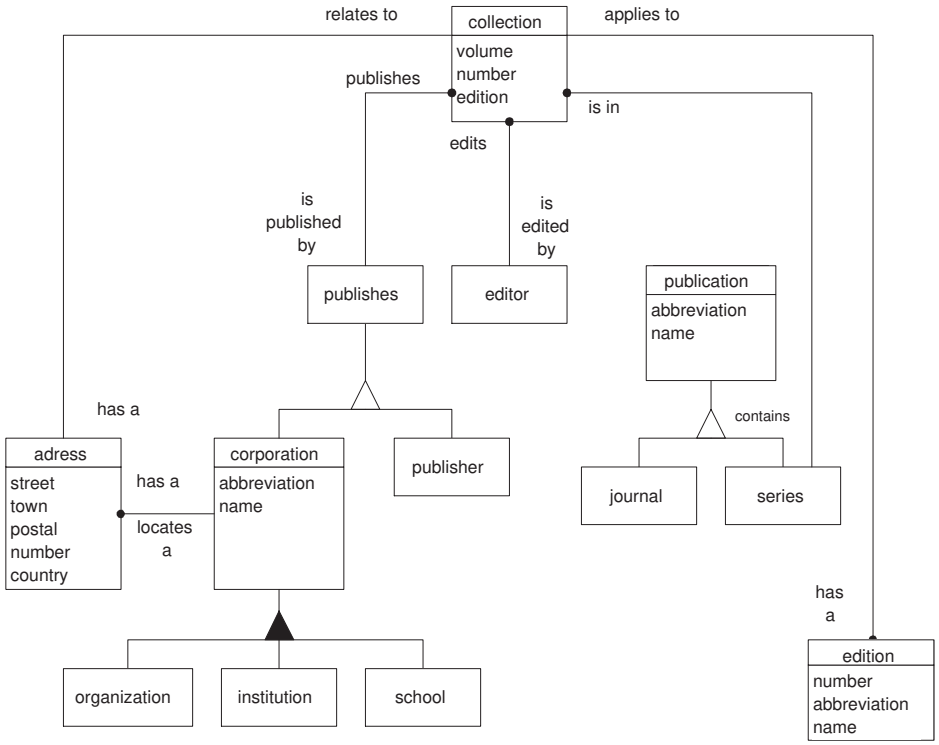


Abbildung B.7: Klasse collection

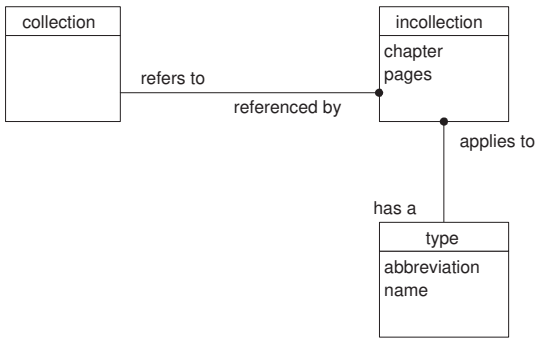


Abbildung B.8: Klasse incollection

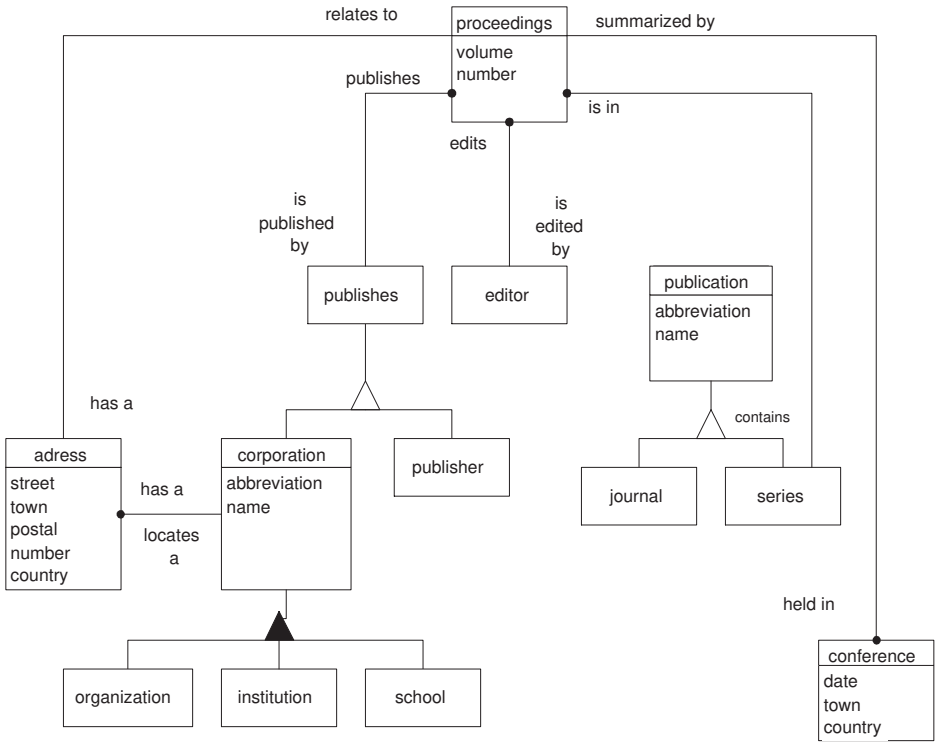


Abbildung B.9: Klasse proceedings

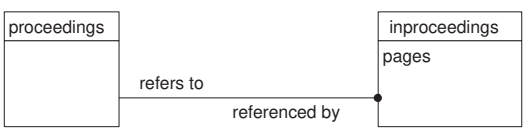


Abbildung B.10: Klasse inproceedings

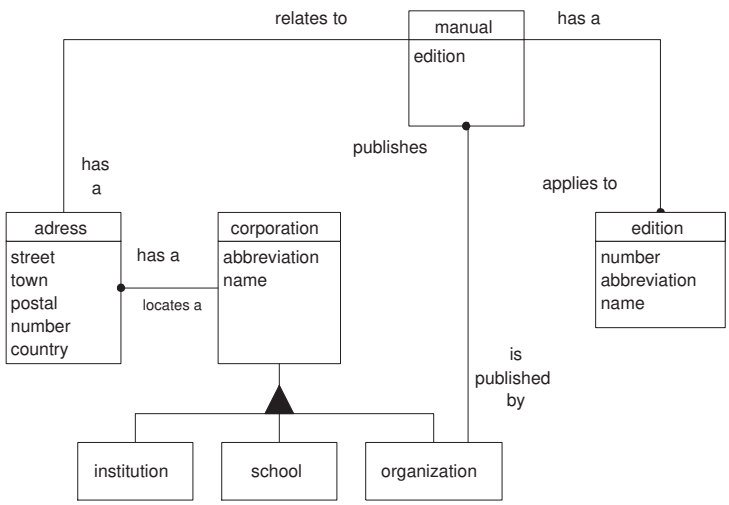


Abbildung B.11: Klasse manual

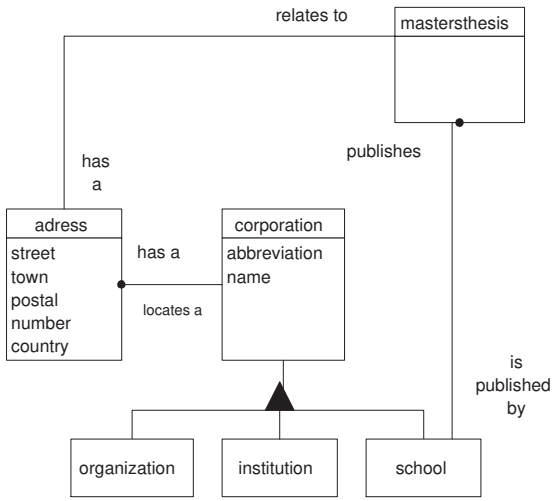
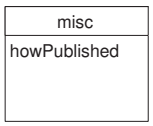


Abbildung B.12: Klasse mastersthesis



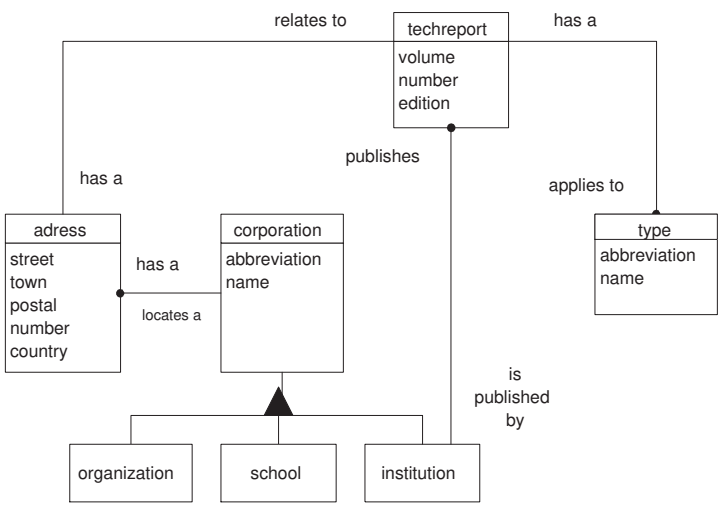


Abbildung B.13: Klasse techreport

Abbildung B.14: Klasse unpublished

Anhang C

Quelltexte

C.1 Klasse MyFrame

```
public class MyFrame extends Frame {
  protected GridBagLayout gridbag = new GridBagLayout();

  public void constrain (Container container, Component component,
                         int grid_x, int grid_y,
                         int grid_width, int grid_height,
                         int fill, int anchor,
                         double weight_x, double weight_y,
                         int top, int left, int bottom, int right)
  {
    GridBagConstraints c = new GridBagConstraints();
    c.gridx = grid_x; c.gridy = grid_y;
    c.gridwidth = grid_width; c.gridheight = grid_height;
    c.fill = fill; c.anchor = anchor;
    c.weightx = weight_x; c.weighty = weight_y;

    if (top + bottom + left + right > 0)
      c.insets = new Insets(top, left, bottom, right);

    ((GridBagLayout)
     container.getLayout()).setConstraints(component,c);

    container.add(component);
  }

  public void constrain (Container container, Component component,
                         int grid_x, int grid_y,
                         int grid_width, int grid_height,
                         int top, int left, int bottom, int right)
  { .. }

  public void constrain (Container container, Component component,
                         int grid_x, int grid_y,
                         int grid_width, int grid_height)
  { .. }
}
```

C.2 Interface ReferenceStore

```
public interface ReferenceStore
{
  // Queries

  public ReferenceList getReferenceList ();
  public ReferenceList getReferenceList (Reference pattern);

  public AuthorList getAuthorList ();
  public AuthorList getAuthorList (Author pattern);

  [..]

  // Actions

  public boolean insert (Reference reference);
  public boolean insert (Author author);
  [..]

  public boolean update (Reference reference);
  public boolean update (Author author);
  [..]

  public boolean delete (Reference reference);
  public boolean delete (Author author);
  [..]

  public boolean exists (Reference reference);
  public boolean exists (Author author);
  [..]
}
```

Literaturverzeichnis

Atkinson et al. 96: Atkinson, M.P., Jordan, M.J., Daynes, L., und Spence, S. *Design Issues for Persistent Java: a type-safe, object-oriented, orthogonally persistent system.* Design and implementation report, Department of Computing Science, University of Glasgow, Februar 1996.

Breu, Brüggemann-Klein 96: Breu, Michael und Brüggemann-Klein, Anne. *Der MeDoc-Dienst: Ein Überblick.* Überblick, Forschungsinstitut für angewandte Softwaretechnik e.V. und Technische Universität München, Dezember 1996.

Crane, Li 96: Crane, Nancy und Li, Xia. *APA Style of Citation.* Diplomarbeit, University of Vermont, Juli 1996.

Dale 94: Dale, Robert. *BibEdit.* Human Communication Research Center, Juli 1994.

Davis 98: Davis, John E. *The JED Editor.* University of Michigan, Mai 1998.

Digital Equipment Corp. 96: Digital Equipment Corp. *Scribe to HTML: Producing HTML Documents with Scribe.* Web Publishing, 1996.

Diverse 97a: Diverse. „BibNet - A Bibliography Network Project“, 1997.

Diverse 97b: Diverse. „The Collection of Computer Science Bibliographies“, 1997.

Doron 98: Doron, Eyal. *BibDB - A BibTeX Database Manager.* Nazareth Illit, Israel, Mai 1998.

Etchinson 98: Etchinson, Roald. *Text Editing and Formatting.* University of Kansas, März 1998.

Flanagan 96: Flanagan, David. *Java in a Nutshell.* O'Reilly & Associates, Inc., Februar 1996.

Floyd 84: Floyd, C. *A systematic look at prototyping.* In: Budde, R., Kuhlenkamp, K., Mathiassen, L., und Züllighoven, H. (Hrsg.). *Approaches to Protoyping.* Springer-Verlag, Berlin u.a., 1984, S. 1–18.

French et al. 97: French, James C., Powell, Allison L., Schulman, Eric, und Pfaltz, John L. *Automating the Construction of Authority Files in Digital Libraries: A Case Study.* In: Peters, Carol und Thanos, Costantino (Hrsg.). *Proceedings of the First European Conference on Research and Advanced Technology for Digital Libraries*, Lecture Notes in Computer Science, Bd. 1324, Pisa, Italy, September 1997, S. 55–71.

Fritzinger, Mueller 96: Fritzinger, J. Steven und Mueller, Marianne. *Java Security.* Whitepaper, Sun Microsystems Inc., November 1996.

Gamma et al. 95: Gamma, E., Helm, R., Johnson, R., und Vlissades, J. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Publishing Company, 1995.

Grønbæk 88: Grønbæk, K. *Rapid Prototyping with Fourth Generation Systems.* Technical Report DAIMI PB - 270, Computer Science Department, Aarhus University, 1988.

Heuer 92: Heuer, Andreas. *Objektorientierte Datenbanken.* Addison-Wesley Publishing Company, Juli 1992.

Int 94: Interactive Development Environments. *Software through Pictures / Object Modeling Technique - Creating OMT Models, Release 2*, 1994.

Jacobsen 95: Jacobsen, Dana. *Canonical Bibliography Format.* Diplomarbeit, Department of Computer Science, UCHICO, Januar 1995.

Jones 97: Jones, D. M. „Hypertext Bibliography Project", 1997.

Kathy 98: Kathy, C. C. *Using the vi editor.* University of Mississippi, April 1998.

Kopka 92: Kopka, Helmut. *LaTeX : Eine Einführung.* Addison-Wesley Publishing Company, Juli 1992.

Lamport 85a: Lamport, Leslie. *The Bibliography Database*, Ch. Appendix B, S. 139–147. Addison-Wesley Publishing Company, Juli 1985.

Lamport 85b: Lamport, Leslie. *LaTeX - A Document Preparation System.* Addison-Wesley Publishing Company, Juli 1985.

Ley 97: Ley. „Bibliography Server", 1997.

Manthey 91: Manthey, R. *Declarative Languages – Paradigm of the Past or Challenge of the Future?* In: *Proceedings of the Kiev East/West Workshop on Next Generation Database Technology*, Lecture Notes in Computer Science, Bd. 504, April 1991.

Odersky, Wadler 97: Odersky, Martin und Wadler, Philip. *Pizza into Java: Translating theory into practice.* In: *Proceedings of the 24th ACM Symposium on Principles of Programming Languages*, Paris, France, Januar 1997.

Orfali et al. 96: Orfali, Robert, Harkey, Dan, und Edwards, Jeri. *The Essential Distributed Objects Survival Guide.* John Wiley & Sons, Inc., Januar 1996.

Rumbaugh et al. 91: Rumbaugh, James, Blaha, M., Premerlani, W., Eddy, F., und Lorensen, W. (Hrsg.). *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, New Jersey, 1991.

Siegfried, Bernstein 91: Siegfried, S. L. und Bernstein, J. *Synoname: The Getty's New Approach to Pattern Matching for Personal Names.* In: *Computers and the Humanities*, Bd. 25(4), 1991, S. 211–226.

Simon 96: Simon, Richard J. *Windows 95 Multimedia & ODBC API Bible.* Prentice, Juli 1996.

Sun Microsystems Inc. 96a: Sun Microsystems Inc. *Java Database Connectivity.* Sun Microsystems Inc., 2550 Garcia Ave, Mtn. View, CA 94043-1100 USA, version 1.10 from 21st october 1996 edition, October 1996.

Sun Microsystems Inc. 96b: Sun Microsystems Inc. *Java Overview.* Whitepaper, Sun Microsystems Inc., 2550 Garcia Ave, Mtn. View, CA 94043-1100 USA, Januar 1996.

Sun Microsystems Inc. 96c: Sun Microsystems Inc. *Object Serialization for Java.* Sun Microsystems Inc., 2550 Garcia Ave, Mtn. View, CA 94043-1100 USA, version Beta 3 edition, November 1996.

Sun Microsystems Inc. 96d: Sun Microsystems Inc. *Remote Method Invocation for Java.* Sun Microsystems Inc., 2550 Garcia Ave, Mtn. View, CA 94043-1100 USA, version Beta 3 edition, October 1996.

Sun Microsystems Inc. 97a: Sun Microsystems Inc. *Java Developer Kit.* Sun Microsystems Inc., 2550 Garcia Ave, Mtn. View, CA 94043-1100 USA, version 1.1 from january 1997 edition, jan 1997.

Sun Microsystems Inc. 97b: Sun Microsystems Inc. *Java Interface Definition Language.* Sun Microsystems Inc., 2550 Garcia Ave, Mtn. View, CA 94043-1100 USA, version 1.1 from 1st february 1997 edition, Januar 1997.

Weizmann 94: Weizmann, A. C. *bibview 2.0*, Juni 1994.

XEmacs Organisation 98: XEmacs Organisation. *XEmacs – Emacs: The Next Generation.* XEmacs Organisation, Februar 1998.