Integration von Literaturdatenbanken über das WWW

Diplomarbeit

Universität Rostock, Fachbereich Informatik

vorgelegt von
Weber, Gunnar
geboren am 30.01.1973 in Rostock

Betreuer: Prof. Dr. Andreas Heuer
Prof. Dr. Peter Forbrig
Dipl.-Inf. Jürgen Schlegelmilch

Abgabedatum: 01.06.1998

Zusammenfassung

Diese Arbeit bietet eine Lösung für die Integration der unterschiedlichen Literaturdatenbankformate von BibTeX, *refdbms* und *NCSTRL* in einer Datenbank, auf die über das WWW zugegriffen wird. Dazu werden zunächst die einzelnen Literaturdatenbankformate betrachtet. Im Anschluß daran wird ein objektorientiertes Datenbankschema entwickelt, das die adäquate Speicherung der Einträge aller betrachteten Literaturverwaltungssysteme erlaubt. Danach werden die Strukturkonflikte, die bei der Integration auftreten, diskutiert und Lösungen vorgeschlagen. Aufbauend auf diesem Schema wird ein Datenbanksystem entwickelt, das folgende Möglichkeiten bietet: Import von Einträgen in den einzelnen Formaten, Änderung/Reklassifizierung von Einträgen, Anfragen an die Datenbank, Export von Einträgen in die einzelnen Formate.

Das Ergebnis der Arbeit ist ein auf O_2 basierendes Datenbanksystem mit einer Implementierung der Import-/Exportschnittstelle für das BibTeX-Format. Für die WWW-Präsentation wird das Modul O_2Web genutzt.

Abstract

This paper offers a solution for the integration of the bibliography database formats BibTeX, *refdbms* and *NCSTRL* in one database, which is accessible via the WWW. First the several bibliography formats are analysed. Then an object-oriented database scheme is developed which allows the corresponding storage of entries of all examined formats. After that the structure conflicts arising from the integration are discussed and solutions are suggested. Based on this scheme a database system is developed which offers the following capabilities: import of entries in the formats listed above, change/reclassification of entries, queries to the database, export of entries in several formats.

The result is an O_2-based database system with an implementation of the import/export interface for the BibTeX format. For the WWW representation the module O_2Web is used.

CR-Klassifikation

H.2.5. Heterogeneous Databases
H.2.8. Database applications
J.5 Literature

Key Words

WWW, Schemaintegration, Literaturdatenbanken, objektorientierte Datenbanken, Datenbankanwendung

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Zielsetzung

Eine Literaturdatenbank besteht aus Referenzen, die Artikel, Bücher, technische Berichte usw. beschreiben. Die Referenzen können neben den bibliographischen Angaben weitere Informationen wie einen Abstrakt des Dokumentes oder Stichwörter, die bei der Suche nach dem Dokument hilfreich sind, enthalten. Einige Literaturdatenbanken nehmen nur Referenzen einer bestimmten Klasse (z.B. technische Berichte) auf, bei anderen werden die einzelnen Referenzen einer Klasse (z.B. Artikel oder Buch) zugeordnet. Bei vielen Literaturdatenbanken wird jedem Eintrag ein Schlüsselwort zugewiesen, über das ein Eintrag eindeutig identifiziert werden kann. Eine eindeutige Identifizierung ist vor allem für Literaturdatenbanken wichtig, die von bestimmten Werkzeugen für die automatische Generierung des Literaturverzeichnisses eines Dokumentes genutzt werden.

Literaturdatenbanken werden im allgemeinen von Einzelpersonen gepflegt und ohne Abstimmung mit anderen erweitert. Das führt zu Problemen, sobald die Literaturdatenbanken mehrerer Personen kombiniert werden müssen. Typische Probleme sind:

- verschiedene Schlüssel für dasselbe Dokument,
- gleiche Schlüssel für verschiedene Dokumente,
- unterschiedliche Klassifizierung desselben Dokuments,
- abweichende Attributwerte für dasselbe Dokument sowie
- inkonsistente Schreibweisen bei eigentlich identischen Attributwerten (z.B Autoren-, Verlags-, Zeitschriftennamen).

Oft kommt auch der Fall vor, daß mehrere Literaturdatenbanken, die in verschiedenen Formaten vorliegen, zu einer zusammengefaßt werden sollen. Dazu muß

man das Format der resultierenden Datenbank vorher festlegen und die Literaturdatenbanken, die nicht das gewünschte Format haben, durch entsprechende Werkzeuge transformieren. Anschließend können sie dann kombiniert werden, wobei auch hier die oben erwähnten Probleme auftreten können. Hinzu kommen Konflikte, die bei der Transformation gelöst werden müssen. Solche Konflikte sind z.B.:

- gleiche Attributnamen für semantisch unterschiedliche Attribute,
- unterschiedliche Attributnamen für semantisch gleiche Attribute,
- unterschiedliche Formate von Attributwerten in gleichen Attributen und
- die Darstellung eines Attributs durch mehrere Attribute im anderen Format.

Für die Verwaltung von Literaturangaben werden oft BibTeX-Dateien verwendet. Obwohl es sich hier nur um Dateien in einem bestimmten Format handelt, kann man sie trotzdem als „Literaturdatenbank“ [Kop96] bezeichnen. Diese Dateien haben eine weite Verbreitung, da sie zusammen mit dem Programm BibTeX für die Erstellung von Literaturverzeichnissen in LaTeX-Dokumenten verwendet werden können. *Refdbms* und Networked Computer Science Technical Reports Library (*NCSTRL*) sind weitere häufig verwendete Literaturdatenbanken. *NCSTRL* ist nur für technische Berichte gedacht, während in BibTeX-Dateien und in *refdbms* die einzelnen Einträge bestimmten Klassen wie *Artikel* oder *Buch* zugeordnet werden. Diese Zuordnung ist dem jeweiligen Anwender überlassen. Eine spätere Änderung der Zuordnung eines Eintrags zu einer Klasse wird als *Reklassifizierung* bezeichnet.

Ziel der vorliegenden Arbeit ist es, ein Datenbanksystem zu entwerfen, auf das über das WWW zugegriffen werden kann und das die am Anfang beschriebenen Probleme zumindest teilweise löst. Es soll folgende Funktionen bieten:

- die Speicherung beliebiger BibTeX-, *refdbms*- und *NCSTRL*-Einträge,
- die Änderung vorhandener Einträge inklusive Reklassifizierung,
- eine Import-Schnittstelle für Dateien in den verschiedenen Formaten mit
  - Konfliktauflösung,
  - einer Unterscheidung zwischen deutschen und englischen Einträgen und
  - einer Zuordnung der Einträge zum jeweiligen Erfasser,
- Konsistenzprüfungen (Eindeutigkeit der Schlüssel, Zeitschriften, Verlage, Autoren),

- eine Export-Schnittstelle für frei wählbare Referenzen sowie
- mehrere Sichten auf die Datenbank, die die Suche nach
  - Themen,
  - Autoren,
  - den Einträgen eines Erfassers und
  - einzelnen Attributen

  ermöglichen.

Das Datenbanksystem basiert auf O_2, die Implementierung der Funktionalität erfolgt mit O_2C. Für die WWW-Präsentation wird das Modul O_2Web genutzt. Die Import- und die Exportschnittstelle wird für die drei zugrundeliegenden Literaturdatenbanken konzipiert und für BibTeX-Dateien beispielhaft implementiert.

Anwendungsszenario Ein Anwender kann weiterhin seine Literaturreferenzen in lokalen BibTeX-Dateien verwalten. Diese können in die Literaturdatenbank importiert werden. Dadurch ist der Anwender in der Lage, alle Literaturreferenzen, die in verschiedenen lokalen BibTeX-Dateien enthalten sind, in einer Datenbank zu sammeln. Inkonsistenzen, die zwischen den Dateien auftreten können, werden beim Import in die Literaturdatenbank beseitigt. Außerdem erhält der Nutzer des Systems die Möglichkeit, auf Literaturreferenzen, die von anderen Personen angegeben wurden, zuzugreifen. Die lokalen Eigenheiten des Anwenders werden durch eine zu integrierende Nutzerkennung berücksichtigt. Die in der Datenbank vorhandenen Referenzen können in BibTeX-Dateien exportiert werden. Später sollen dann auch Exporte in HTML möglich sein, um Anfrageergebnisse an verschiedenen Stellen in der WWW-Präsentation des Fachbereiches oder in Jahresberichten unterbringen zu können.

Abgrenzung zu anderen Bibliographie-Systemen Es existieren bereits verschiedene Bibliographie-Systeme, beispielsweise OPAC (Online Public Access Catalogue) und ScatMan. OPAC wird von der Universitätsbibliothek Rostock genutzt. Es handelt sich um ein zentralisiert gespeistes System und enthält nur Literatur, die auch in der Universitätsbuchhandlung nachgewiesen werden kann. ScatMan ist ein lokales Datenbanksystem, mit dem BibTeX-Referenzen verwaltet werden können. Das System ermöglicht den Import von BibTeX-Dateien sowie den Export vorhandener Einträge in das BibTeX-Format. Es ist nicht auf die Nutzung von mehreren Anwendern zugeschnitten und berücksichtigt deshalb keine Konfliktauflösung. Beide Systeme stimmen nicht mit dem beschriebenen Anwendungsszenario überein, da es sich bei OPAC um ein zentralisiert gespeistes und bei ScatMan um ein lokales Datenbanksystem handelt.

1.2 Gliederung

Im folgenden wird ein kurzer Überblick über den Aufbau dieser Arbeit gegeben:

Zunächst werden im **Kapitel 2** die Formate der Literaturdatenbanken, die im Datenbanksystem berücksichtigt werden sollen, vorgestellt. Ein Vergleich dieser Formate findet in **Kapitel 3** statt. Für die Konflikte, die zwischen den einzelnen Formaten auftreten, werden Lösungen angegeben. Darauf aufbauend wird dann ein objektorientiertes Datenbankschema entworfen.

Kapitel 4 beschäftigt sich mit den Möglichkeiten der Übertragung von Dateien an den WWW-Server. Anschließend werden die Möglichkeiten hinsichtlich der Nutzbarkeit für den Import von Literaturdateien miteinander verglichen.

In **Kapitel 5** wird das Konzept für das Datenbanksystem vorgestellt. **Kapitel 6** beschreibt die Umsetzung dieses Konzepts in ein konkretes, auf O_2 basierendes Datenbanksystem.

Abschließend erfolgt in **Kapitel 7** eine Zusammenfassung der Arbeit, gefolgt von einem Ausblick auf weitere Arbeiten.

Beispiele für die Literaturdatenbanken, auf denen das entworfene Datenbanksystem basiert, werden im **Anhang A** vorgestellt. **Anhang B** zeigt, wie die Datenbank und die zugehörigen Programme erstellt werden. Im **Anhang C** wird beschrieben, wie das Datenbanksystem zu nutzen ist. Die einzelnen Klassen- und Methodendefinitionen sind im **Anhang D** zu finden.

Kapitel 2

Formate für Literaturangaben

Dieses Kapitel beschäftigt sich mit den Formaten der Literaturdatenbanken, die im Datenbanksystem berücksichtigt werden sollen. Die Eintragstypen, denen die Literaturangaben in BibTeX-und *refdbms*-Dateien zugeordnet sind, werden detailliert beschrieben. Für *NCSTRL* ist dies nicht nötig, da dieses System nur technische Berichte verwaltet. Außerdem erfolgt eine genaue Darstellung, welche Angaben für die Einträge benötigt und wie diese formatiert werden. Bei der Beschreibung der Literaturdatenbanken werden Grammatiken in der erweiterten Backus-Naur-Form (EBNF) verwendet. Terminalsymbole, also feststehender Text, der nur auf der rechten Seite einer Regel vorkommen kann, sind **fett** hervorgehoben. Um Terminale, die nur aus einem einzigen Zeichen bestehen, besser erkennen zu können, werden diese zusätzlich in Hochkommata eingeschlossen. Nichtterminale sind *kursiv* angegeben. Die in eckige Klammern [ ] eingeschlossenen Ausdrücke sind optional. Steht ein Ausdruck in geschweiften Klammern {}, kann er null- oder mehrmalig auftreten. Mit einem senkrechten Strich | werden Alternativen getrennt, wobei zu beachten ist, daß Verkettungen Vorrang gegenüber | haben. Im Anhang A sind für die einzelnen Formate Beispieleinträge enthalten.

2.1 Das BibTeX-Format für Literaturdatenbanken

BibTeX-Literaturdatenbanken [GMS94, Kop96, Pat88] sind Dateien, in denen Literaturinformationen in einem bestimmten Format abgelegt werden. Sie werden in Verbindung mit dem Programm BibTeX genutzt, um Literaturverzeichnisse in LaTeX-Dokumenten zu erzeugen. Das Aussehen eines solchen Literaturverzeichnisses wird durch die BibTeX-Stildateien beeinflußt, die individuell geändert werden können. In den nächsten Abschnitten erfolgt eine Vorstellung des Aufbaus einer BibTeX-Datei, der möglichen Eintragstypen sowie der Standardfelder und ihrer Formate.

2.1.1 Aufbau

Eine BibTeX-Literaturdatenbank ist wie folgt aufgebaut:

BibTeX-Datei	::=	{'@' *Eintrag* | *Kommentar*}
Eintrag	::=	**preamble** *Präambel-Eintrag*
	|	**string** *String-Eintrag*
	|	*Literatureintrag*
Präambel-Eintrag	::=	'**{**' *Wert* '**}**'
	|	'(' *Wert* ')'
String-Eintrag	::=	'**{**' *Zuweisung* '**}**'
	|	'(' *Zuweisung* ')'
Literatureintrag	::=	*Eintragstyp Angaben*
Angaben	::=	'**{**' [*Schlüsselwort*] [',' *Felder*] '**}**'
	|	'(' [*Schlüsselwort*] [',' *Felder*] ')'
Felder	::=	*Zuweisung* {',' [*Zuweisung*]}
Zuweisung	::=	*Name* '=' *Wert*
Wert	::=	*Text* {'#' *Text*}
Text	::=	*Abkürzung*
	|	*Zeichenkette*
	|	*Nummer*
Zeichenkette	::=	'"' {*Zeichen*} '"'
	|	'**{**' {*Zeichen*} '**}**'

Das Zeichen `@` leitet einen Literatureintrag ein. Danach folgt der *Eintragstyp*. Die möglichen Eintragstypen werden im Abschnitt 2.1.5 beschrieben. Das Klammerpaar `{}` bzw. `()` schließt die Literaturangabe ein. Das erste Wort innerhalb des Klammerpaares ist das *Schlüsselwort*. Es besteht aus einer beliebigen Folge von Buchstaben, Zahlen und Zeichen, mit Ausnahme des Kommas. Die einzelnen Eintragungen werden als *Felder* bezeichnet, die durch Kommata voneinander getrennt sind. Sie enthalten jeweils Teilinformationen und bestehen aus dem *Namen*, einem Gleichheitszeichen und dem *Wert*. Leerzeichen vor oder nach dem Gleichheitszeichen bzw. Komma werden ignoriert. Ein *Text* wird entweder in Anführungsstrichen oder in geschweiften Klammern eingeschlossen, wobei darauf zu achten ist, daß die gleiche Anzahl von öffnenden und schließenden geschweiften Klammern vorhanden ist. Solche Texte werden als *Zeichenkette* bezeichnet. Besteht der *Text* nur aus Ziffern, können die Klammern bzw. Anführungsstriche weggelassen werden. Er wird in diesem Fall als *Nummer* interpretiert. Soll der *Text* eine *Abkürzung* darstellen, entfällt ebenfalls der Einschluß in Klammern oder Anführungsstriche. Mehrere Zeichenketten, Nummern und Abkürzungen können durch den Verknüpfungsoperator `#` zu einem *Wert* verbunden werden. Über den `STRING`-Befehl kann man Abkürzungen definieren. Die Nutzung von Abkürzun-

gen und der Verwendungszweck des `PREAMBLE`-Befehls werden in den nächsten Abschnitten erläutert.

Beim Eingabetyp, beim Schlüsselwort und bei den Feldnamen wird nicht zwischen Groß- und Kleinbuchstaben unterschieden, d.h. der Eintragstyp `BOOK` kann z.B. als `Book`, `book` oder `bOOk` angegeben werden.

2.1.2 Abkürzungen

In jedem Feld kann der *Text* durch eine Abkürzung ersetzt werden. Dies ist sinnvoll, wenn dieser Text in mehreren Einträgen vorkommt, da so eine einheitliche Schreibweise garantiert wird. Einige häufig verwendete Abkürzungen sind in den BIBTEX-Stildateien vordefiniert. Die Abkürzungen werden bei der Erzeugung des Literaturverzeichnisses durch den zugehörigen *Wert* ersetzt. Der *Name* einer Abkürzung kann aus Buchstaben, Ziffern und Zeichen außer `" # % ' ( ) , = { }` bestehen. Beim Namen wird ebenso wie beim `String`-Befehl die Groß-/Kleinschreibung nicht berücksichtigt. Eine Abkürzung darf nicht in Anführungsstriche oder geschweifte Klammern eingeschlossen sein und muß vor ihrer ersten Verwendung definiert werden.

2.1.3 Präambel

Mit dem `PREAMBLE`-Befehl lassen sich Befehle definieren, die in den Feldtexten verwendet werden können. Die Ausführung erfolgt dann bei der Erstellung des Literaturverzeichnisses.

Die einzelnen Befehlsdefinitionen, die in geschweifte Klammern bzw. Anführungszeichen einzuschließen sind, werden durch das `#`-Symbol miteinander verknüpft.

2.1.4 Felder

In der folgenden Liste sind alle Felder angegeben, die in den Standard-BIBTEX-Stilen vorhanden sind. In einigen erweiterten BIBTEX-Stilen können weitere Felder, wie z.B. *abstract*, definiert sein. Felder, die im verwendeten Stil nicht bekannt sind und somit bei der Erstellung eines Literaturverzeichnisses ignoriert werden, können dazu verwendet werden, um weitere Angaben in einen Eintrag einzufügen.

address Verlagsanschrift: bei bekannten Verlagen genügt die Angabe des Verlagsortes; bei kleineren, wenig bekannten Verlagen sollte die gesamte Anschrift angegeben werden

annote Kommentar: wird in Stilen verwendet, die ein Literaturverzeichnis mit Kommentaren erzeugen

author der oder die Autorennamen; das Format dieses Feldes wird in den Abschnitten 2.1.7.1 und 2.1.7.2 beschrieben

booktitle Name des Buches, in dem der Eintrag enthalten ist (als Kapitel oder Ausschnitt) bzw. dessen Teile eigene Titel haben

chapter Kapitelnummer bzw. Abschnittsnummer

crossref Datenbankschlüsselwort des Eintrags, auf den verwiesen wird; eine genauere Beschreibung erfolgt in Abschnitt 2.1.8

edition Auflage eines Buches

editor der oder die Namen des/der Herausgeber(s); das Format dieses Feldes wird in den Abschnitten 2.1.7.1 und 2.1.7.2 beschrieben

howpublished für Buchveröffentlichungen außerhalb eines Verlages wie z.B. `Selbstveröffentlichung` u.ä.

institution Institution, durch die eine verlagsfreie Veröffentlichung erfolgte

journal Name der Zeitschrift: für einige Journale gibt es Kurzformen

key bestimmt bei Fehlen eines *author*- oder *editor*-Feldes die alphabetische Einordnung ins Literaturverzeichnis

month Monat, in dem die Arbeit veröffentlicht wurde; bei einer unveröffentlichten Arbeit der Monat, in dem sie geschrieben wurde

note zusätzliche Hinweise, die im Literaturverzeichnis nützlich sein können

number laufende Nummer eines Journals, einer Zeitschrift oder eines technischen Berichts

organization Name der Organisation, die die Tagung oder Konferenz ausgerichtet oder finanziert hat

pages Angabe von einer oder mehreren Seiten bzw. Seitenbereichen

publisher Name des Verlages

school Name der Hochschule oder Universität, an der die Diplom- oder Doktorarbeit geschrieben wurde

series Name eines Satzes zusammengehöriger Bücher oder einer Buchreihe

title Titel des Werkes

type Typ des Berichtes, z.B. `Forschungsbericht`; als Standard wird `Technical Report` verwendet

volume Band einer Zeitschrift oder eines Buches, das zu einer mehrbändigen Reihe gehört

year Erscheinungsjahr bei veröffentlichten oder Entstehungsjahr bei unveröffentlichten Werken

2.1.5 Eintragstypen

Für unterschiedliche Arten von Veröffentlichungen sind verschiedene Informationen erforderlich. Abhängig vom Eintragstyp sind deshalb bestimmte Felder *zwingend* notwendig. *Optional* können weitere Angaben hinzugefügt werden. Felder, die weder zwingend noch optional sind, werden als *überflüssig* bezeichnet. Diese Felder werden bei der Erzeugung eines Literaturverzeichnisses nicht verwendet. Sie können aber sinnvolle Informationen wie eine inhaltliche Zusammenfassung des Literatureintrags enthalten, die in anderen Datenbankprogrammen genutzt werden können.

Die möglichen Eintragstypen hängen von der verwendeten BIBTEX-Stilvorlage ab. Die meisten BIBTEX-Stile stellen vierzehn Eintragstypen bereit, die im folgenden vorgestellt werden. Dabei wird auch angegeben, welche Felder für einen Typ *zwingend* bzw. *optional* sind.

article Artikel aus einer Zeitung oder einem wissenschaftlichen Journal
zwingend: author, title, journal, year
optional: volume, number, pages, month, note

book Buch, in dem explizit der Verlag angegeben ist
zwingend: author oder editor, title, publisher, year
optional: volume oder number, series, address, edition, month, note

booklet gedrucktes oder gebundenes Werk ohne Angabe eines Verlages oder einer Institution
zwingend: title
optional: author, howpublished, address, month, year, note

conference entspricht dem Eintragstyp *inproceedings* und wird aus Kompa-

tibilitätsgründen bereitgestellt

inbook Teil eines Buches, z.B. Kapitel, Abschnitt und/oder ein Seitenbereich
zwingend: author oder editor, title, chapter und/oder pages, publisher, year
optional: volume oder number, series, type, address, edition, month, note

incollection Teil eines Buches mit eigenem Titel
zwingend: author, title, booktitle, publisher, year
optional: editor, volume oder number, series, type, chapter, pages, address, edition, month, note

inproceedings Artikel in einem Konferenzband
zwingend: author, title, booktitle, year
optional: editor, volume oder number, series, pages, address, month, organization, publisher, note

manual technische Dokumentation
zwingend: title
optional: author, organization, address, edition, month, year, note

mastersthesis Diplomarbeit
zwingend: author, title, school, year
optional: type, address, month, note

misc Dokument, das keinem anderen Eintragstyp zugeordnet werden kann
zwingend: eines der optionalen Felder
optional: author, title, howpublished, month, year, note

phdthesis Doktorarbeit
zwingend: author, title, school, year
optional: type, address, month, note

proceedings Konferenzbericht
zwingend: title, year
optional: editor, volume oder number, series, address, publisher, note, month, organization

techreport Bericht, der von einer Hochschule oder Institution veröffentlicht wurde, normalerweise numerierte Ausgabe in einer Reihe
zwingend: author, title, institution, year
optional: type, number, address, month, note

unpublished unveröffentlichtes Dokument, das Autor und Titel hat
zwingend: author, title, note
optional: month, year

Bei allen Typen ist ein optionales *key*-Feld erlaubt. Dieses Feld sollte dann eingesetzt werden, wenn für den Eintrag weder ein *author*- noch ein *editor*-Feld angegeben werden kann. In solchen Fällen ermöglicht das *key*-Feld die alphabetische Einordnung des Eintrags in ein Literaturverzeichnis.

2.1.6 Sonderzeichen

Zu den Sonderzeichen zählen spezielle Buchstaben wie ß und Akzente. Für BIBTEX ist ein Sonderzeichen ein Konstrukt, das auf der linken Seite auf oberster Ebene mit einer öffnenden geschweiften Klammer beginnt, unmittelbar gefolgt von einem Backslash, und das mit der zugehörigen schließenden geschweiften Klammer endet. Oberste Ebene bedeutet, daß dieses Konstrukt nicht in weitere Klammern eingefaßt werden darf, ausgenommen der Klammern, die den gesamten Feldtext begrenzen.

Der Umlaut ö ist z.B. als `{\"o}` oder `{\"{o}}` anzugeben. Für den Namen `Gödel` ist z.B. die Schreibweise `G{\"o}del` korrekt. Dagegen sind `{G{\"{o}}del}` oder `{G\"{o}}del` falsch.

2.1.7 Spezielle Feldformate

Die Felder *author* und *editor* enthalten Autoren- bzw. Herausgeberangaben. Diese Angaben müssen einem bestimmten Format genügen. In den nächsten beiden Abschnitten wird beschrieben, in welcher Form der Name einer Person angegeben werden muß und wie die Verknüpfung mehrerer Personen erfolgt.

Für die Angaben in den Feldern *title* und *booktitle* gelten ebenfalls besondere Regeln, nach denen diese formatiert werden. Eine Darstellung dieser Regeln erfolgt in Abschnitt 2.1.7.3.

2.1.7.1 Namen

Namen können in verschiedenen Formen angegeben werden:

```
" Vorname von Nachname "
" von Nachname, Vorname "
" von Nachname, Jr, Vorname "
```

Sie können also aus vier Teilen bestehen, die mit *Vorname*, *von*, *Nachname* und *Jr* bezeichnet werden. Kleingeschriebene Wörter in einer Namensangabe werden dem *von*-Teil zugeordnet. Bei amerikanischen Angaben erscheint häufig ein `Jr.` für `Junior`. Diese Information wird *Jr*-Teil genannt. Jeder Teil besteht aus einer Liste von Namensangaben, die bis auf die Nachnamensliste leer sein können. Teile eines Namens werden als Einheit angesehen, wenn sie in geschweifte Klammern eingefaßt sind.

In der ersten Form bewirkt der *von*-Teil eine Trennung der Vor- und Nachnamen. Ist er nicht vorhanden, wird das letzte Wort als *Nachname* angesehen. Bei der zweiten Form werden die Nachnamen durch ein Komma von den Vornamen getrennt. Für die Angabe eines *Jr*-Teils gibt es zwei Möglichkeiten. Entweder wird für die Namensangabe die dritte Form gewählt (dem *Jr*-Teil wird ein Komma vorangestellt), oder er wird als Teil des Nachnamens betrachtet. In diesem Fall sollte der Nachname zusammen mit dem *Jr*-Teil in geschweifte Klammern eingefaßt werden.

In den meisten Fällen kann die erste Form verwendet werden. Sie sollte jedoch nicht angewendet werden, wenn ein *Jr*-Teil vorhanden ist oder der *Nachname* aus mehreren Namen besteht und kein *von*-Teil vorhanden ist.

Die Autoren- oder Herausgebernamen sollten immer so angegeben werden, wie sie in dem Werk, auf das man sich bezieht, erscheinen. BibTeX stellt eine Möglichkeit bereit, um verschiedene Schreibweisen bei Vornamen in Übereinstimmung zu bringen. Sie wird im folgenden Beispiel vorgestellt: Es soll ein Dokument, dessen Autor als `D. E. Knuth` angegeben ist, in eine BibTeX-Datei einfügt werden. In dieser Datei existiert ein Eintrag mit dem Autoren `Donald E. Knuth`. Handelt es sich in beiden Fällen um dieselbe Person, so sollte für die Autorenangabe die Schreibweise

```
author = {D[onald] E. Knuth}
```

verwendet werden. Dadurch wird deutlich, daß es sich bei beiden Angaben um denselben Autoren handelt, obwohl in der Orginalveröffentlichung eine andere Form gewählt wurde.

2.1.7.2 Mehrteilige Autoren- und Herausgeberangaben

Besteht die Autoren- oder Herausgeberangabe aus verschiedenen Personen, so sind die einzelnen Namen durch das Wort `and` zu trennen. Die Autoren „Michael Gossens / Frank Mittelbach / Alexander Samarin“ sind somit in der Form

```
author = {Michael Gossens and Frank Mittelbach and
          Alexander Samarin}
```

anzugeben. Die Autoren- oder Herausgeberliste kann man mit `and others` beenden, wenn sehr viele Autoren oder Herausgeber vorhanden sind, die nicht alle eingegeben werden sollen. Bei der Erstellung des Literaturverzeichnisse wird `and others` in die gebräuchliche Form `et al` umgewandelt. Die Wörter `and` und `others` sind in geschweifte Klammern einzufassen, wenn sie als Teil einer Namensangabe angesehen werden sollen.

2.1.7.3 Überschriften

Bei englischen Buchtiteln werden alle Wörter außer Konjunktionen und Präpositionen groß geschrieben. Derselbe Titel wird bei einem englischen Artikel bis auf Eigennamen und das erste Wort in Kleinbuchstaben angegeben. In den meisten BibTeX-Stilen wird eine Artikelüberschrift auch dann in Kleinbuchstaben gedruckt, wenn diese groß geschrieben wurde. Dies ist bei deutschen Literaturangaben nicht üblich. Wenn der Titel im Literaturverzeichnis unverändert ausgegeben werden soll, dann müssen die entsprechenden Wörter bzw. Großbuchstaben zusätzlich in geschweifte Klammern eingeschlossen werden.

2.1.8 Querverweise

Einträge können Verweise auf andere enthalten. Enthält ein Literatureintrag ein *crossref*-Feld, dann übernimmt BibTeX nicht spezifizierte Felder von dem Eintrag, auf den *crossref* zeigt. Ein Eintrag, auf den verwiesen wird, muß in der Literaturdatenbank hinter dem letzten Eintrag stehen, der auf ihn zeigt, und er darf selbst keine Verweise enthalten. Sinnvoll sind solche Querverweise zum Beispiel, wenn man mehrere Artikel aus einem Konferenzband zitiert. Der Konferenzband sollte dann als eigenständiger Eintrag in die Literaturdatenbank aufgenommen werden, auf den in jedem Artikel verwiesen wird.

2.2 Die *refdbms*-Literaturdatenbank

Refdbms [RG94] ist ein System, daß zur Erstellung von Literaturverzeichnissen genutzt werden kann. Es bietet die Möglichkeit, Einträge in der Datenbank zu suchen und sie in andere Literaturdatenbankformate wie BibTeX umzuwandeln. Im Gegensatz zu vielen anderen Systemen ist *refdbms* in der Lage, die eigene lokale Datenbank mit anderen Nutzern über das Internet zu teilen. In den nächsten Abschnitten werden der Aufbau einer *refdbms*-Datei, die möglichen Felder und

Eintragstypen sowie die Verwendung von Abkürzungen vorgestellt. Im Abschnitt 2.2.2 wird auf einige Besonderheiten bei der Formatierung der Feldinhalte hingewiesen.

2.2.1 Aufbau

Refdbms nutzt ein Format, das dem von *refer* [Les78] ähnelt. Eine *refdbms*-Datei ist folgendermaßen aufgebaut:

refdbms-Datei	::=	*Eintrag Leerzeile* {*Eintrag Leerzeile*}
Eintrag	::=	**%z** *Eintragstyp Zeilenumbruch*
		%K *Schlüsselwort Zeilenumbruch*
		{*Zeile*}
Zeile	::=	'**%**' *Buchstabe Freiraum Wert Zeilenumbruch*
Freiraum	::=	*Leerzeichen* | *Tabulator* {*Leerzeichen* | *Tabulator*}

Alle *Zeilen* mit dem gleichen *Buchstaben* werden einem Typ zugeordnet. Aufeinanderfolgende Zeilen desselben Typs werden als Feld bezeichnet, wobei zu beachten ist, daß viele Felder nur eine Zeile enthalten dürfen. Jeder Eintrag einschließlich des Letzten muß mit einer Leerzeile abgeschlossen werden. In der ersten Zeile ist immer der *Eintragstyp* anzugeben, z.B. `TechReport` oder `Article`. Das *Schlüsselwort* des Eintrags muß in der zweiten Zeile stehen. Die Felder, die danach folgen können, hängen vom Eintragstyp ab und werden im Abschnitt 2.2.3 beschrieben.

2.2.2 Formatierungshinweise

Die Formatierungen für die Angaben in den einzelnen Feldern sind von BibTeX beeinflußt. Folgende Formatierungshinweise sollten befolgt werden:

1. Am Zeilenende sollten bis auf den nachfolgenden Punkt bei einer Abkürzung keine Satzzeichen verwendet werden. Ausgenommen von dieser Regelung sind Felder, die Anmerkungen bzw. inhaltliche Zusammenfassungen beinhalten. Außerdem sollten Details nicht durch zusätzliche Satzzeichen hervorgehoben werden.

2. Akzente und außergewöhnliche Satzzeichen sind in der LaTeX-Form anzugeben. Hervorhebungen sollten durch den Befehl `\em` und nicht durch `\it` angegeben werden. Ansonsten sollte man keine Formatbefehle verwenden, da sie später eingefügt werden.

3. Es werden Striche unterschiedlicher Länge verwendet:

(a) Der *Gedankenstrich* --- ohne voran- und nachgestellten Leerraum wird gewöhnlich genutzt, um eine geklammerte Bemerkung einzuleiten.

(b) Der *Streckenstrich* -- wird für Zahlenbereiche (z.B. 6--9), zwischen Teilen der Nummer eines Technischen Berichts (z.B. HPL--90--20), oder mit voran- und nachgestellten Leerraum auch als Abtrennung von geklammerten Bemerkungen genutzt.

(c) Der *Bindestrich* - wird als Bindezeichen in zusammengesetzten Worten benutzt.

2.2.3 Felder

Die folgende Liste enthält alle Felder, die für einen *refdbms*-Eintrag angegeben werden können. Felder, denen nach dem Buchstaben ein Pluszeichen (+) folgt, können sich über mehrere Zeilen erstrecken. Ein nachfolgender Stern (∗) bedeutet, daß dieses Feld mehrmals als gesondertes Feld auftreten kann. Die anderen Felder dürfen höchstens einmal in einem Eintrag vorkommen und können auch nur eine Zeile umfassen.

%A ∗ **author** - *Autor*: Jeder Autor hat ein separates *%A*-Feld. Der Name sollte in der längsten Form angegeben werden. Leerzeichen, die einen mehrteiligen Nachnamen trennen, ist ein Backslash voranzustellen. Enthält der Name ein `Jr`, so sollte diese Angabe durch ein Komma vom Nachnamen abgetrennt werden. Zum Familiennamen werden `von`, `de`, `van` etc. nur gezählt, wenn sie klein geschrieben sind. Ansonsten werden sie wie Vornamen behandelt. Spezielle Anmerkungen zum Autor gehören in eine gesonderte *%a*-Zeile, die direkt der entsprechenden *%A*-Zeile folgen muß. Die Autorenliste sollte nicht abgekürzt werden, da die Anmerkung `et al` für dieses Feld nicht geeignet ist (alle Autoren müssen somit angegeben werden).

%a ∗ **author note** - *Anmerkung zum Autor*: Dieses Feld dient für Anmerkungen zum/zu unmittelbar davor angegebenen Autor/Autoren, z.B. `Übersetzer`.

%B **booktitle** - *Titel des Buches*, in dem ein Kapitel oder Ausschnitt enthalten ist: Ist der Eintrag ein komplettes Buch, dann muß der Titel im *%T*-Feld angegeben werden.

%b BibTeX **key** - BibTeX-*Schlüssel*: Der angegebene Schlüssel wird statt des Schlüsselwortes des Eintrags genutzt. Dieses Feld ist nur im Zusammenhang mit automatisch erzeugten Datenbanken sinnvoll. Bei deren Erzeugung werden den Einträgen spezielle Nummern als Schlüsselwörter zugeordnet, um die Eindeutigkeit dieser Schlüssel zu gewährleisten. Wenn

man beim Zitieren von Einträgen aber ein Schlüsselwort bevorzugt, das z.B. vom Autorennamen abgeleitet ist, dann muß dieser Schlüssel hier angegeben werden.

%C **conference name** - *Konferenzname*: Der Name der Konferenz, in der das zugehörige Dokument vorgelegt wurde, ist hier anzugeben. Die Nummern sollten nicht ausgeschrieben, sondern in der numerischen Form (z.B. `14th`) angegeben werden.

%c **conference location** - *Konferenzort/-datum*: Für den Konferenzort sind die wichtigsten Teile einer Adresse, gewöhnlich Stadt und Staat/-Land, anzugeben. Das Datumsformat wird im *%D*-Feld beschrieben.

%D **date** - *Datum der Veröffentlichung*: Monatsnamen werden immer durch die ersten drei Buchstaben und einen nachfolgenden Punkt angegeben (Ausnahmen: Mai, Juni u. Juli). Jahreszahlen sind voll auszuschreiben. Die Angabe des Datums erfolgt im Europäischen Stil (Tag Monat Jahr) ohne Kommata. Wenn die Quelle ein Dokumententwurf ist, dann muß dies ebenfalls in diesem Feld gekennzeichnet werden (`draft of ...`). Zeiträume werden durch den Streckenstrich -- angegeben.

%E ∗ **editor** - *Herausgeber*: Jeder Herausgeber wird in einem eigenen *%E*-Feld angegeben (Formatierungen wie im *%A*-Feld).

%e ∗ **editor note** - *Anmerkungen zum Herausgeber*: Die Anmerkungen gelten für den/die unmittelbar davor angegebenen Herausgeber (Formatierungen wie im *%a*-Feld).

%I + **ISBN/ISSN** - *ISBN-/ISSN-Nummer* des Dokuments: Zwischen den Komponenten der Nummer ist ein Streckenstrich -- zu verwenden. Vor der Nummer muß entweder `ISBN` oder `ISSN` stehen. Mehrere Nummern (z.B. eine für Paperback, eine für gebundene Ausgabe) werden in separaten Zeilen angegeben.

%J **journal** - *Name des Journals*, in dem der Artikel erschienen ist: Da Journalnamen Eigennamen sind, werden die ersten Buchstaben der signifikanten Wörter groß geschrieben.

%K **tag** - *Schlüsselwort*, durch das ein Eintrag eindeutig identifiziert werden kann. Es muß immer in der zweiten Zeile eines Eintrags angegeben werden. Das Schlüsselwort sollte aus dem Nachnamen des ersten Autors, verbunden mit den letzten 2 Ziffern der Jahreszahl der Veröffentlichung, gebildet werden. Existiert bereits ein Eintrag mit demselben Schlüssel, dann wird dieser Konflikt durch das Anhängen eines Buchstaben (`a`, `b`, ...) gelöst. Ein Schlüsselwort darf nur Buchstaben und Ziffern enthalten. Es sollte mit einem Großbuchstaben beginnen, außer wenn dies beim

Hauptautor nicht der Fall ist. Freiräume in mehrteiligen Nachnamen sind wegzulassen (z.B. wird `van Jacobsen` zu `vanJacobsen`). Nachfolgende Angaben wie `Jr` sollten nicht in das Schlüsselwort aufgenommen werden. Existiert kein Autorenname, ist der Name der ersten Herausgebers (wenn vorhanden) oder der Verlagsname zu nutzen.

%k + **keywords** - *Stichwörter*, die hilfreich sind, um den Eintrag später wiederzufinden: Die Stichwörter müssen nicht in separaten Zeilen stehen. Sie können durch Leerzeichen und/oder Freiräume getrennt werden. Die Wörter im Titel und die Autorennamen werden automatisch hinzugefügt und müssen somit nicht noch einmal angeführt werden. Bei Stichwörtern ist die Schreibweise zu beachten: Groß- und Kleinbuchstaben sind nicht äquivalent.

%L + **location** - *Orte, an denen das Dokument zu finden ist*: Bei Dokumenten, die elektronisch verfügbar sind, sollten Standardadressen wie URLs verwendet werden.

%N **number** - *Nummer*: Hier wird der Teil eines Buches, eines Bandes oder einer Serie angegeben, in dem das Dokument erschienen ist. Dieses Feld sollte auch genutzt werden, um die Auflage oder Version anzugeben (bevorzugt in numerischer Form mit der Angabe `edition` bzw. `version`). Die einzelnen Angaben werden durch Kommata voneinander getrennt.

%O + **public note** - *öffentliche Anmerkung*: Diese Anmerkung erscheint in einem Literaturverzeichnis. Typische Anmerkungen sind: ein `\cite{...}`-Verweis auf die Quelle, wenn das Dokument eine Übersetzung ist, oder eine andere Sprache als Englisch. Bemerkungen oder Zusammenfassungen sind im *%o*-Feld anzugeben. Bei unveröffentlichten Artikeln steht in diesem Feld der Zweck, für den sie gedacht sind.

%o + **private note** - *private Anmerkung*: Diese Anmerkung wird in den normalen Stilen nicht in das Literaturverzeichnis aufgenommen. Die sinnvollste Nutzung dieses Feldes ist eine kurze Zusammenfassung der wichtigsten Erkenntnisse und Ergebnisse des Dokumentes. Es gelten die LaTeX-Formatierungsvorschriften und es darf keine Leerzeile verwendet werden, um Absätze zu trennen: Stattdessen ist der Befehl `\par` zu benutzen.

%P **pages** - *Seiten*, über die sich die Veröffentlichung erstreckt: Zwischen den Zahlen muß ein Leerzeichen angegeben werden. Ein `+` nach einer Seitenzahl bedeutet, daß die folgenden Seiten nicht benachbart sind. Gibt es mehrere Seitenzahlenbereiche, dann sind diese durch ein Komma und ein Leerzeichen zu trennen.

%p **publisher** - *Verlag*: In diesem Feld ist der Name des Verlags oder der Institution anzugeben, die das Dokument herausgebracht hat.

%R **report number** - *Nummer des Berichts*: Dieses Feld enthält die Nummer eines Technischen Reports, eine Ordnungsnummer oder eine Katalognummer. Dabei ist der Typ der Nummer, z.B. `Technical report`, anzugeben, außer wenn das Feld nur eine technische Berichtsnummer enthält, die ausschließlich aus Ziffern und Großbuchstaben zusammensetzt ist. Zwischen Nummern und Paaren von Großbuchstaben ist ein Streckenstrich (--) zu verwenden.

%S **series** - *Name der Serie*, zu der ein Eintrag gehört: Für den Namen gelten die gleichen Formatierungsvorschriften wie im *%T*-Feld.

%s + **submitter** - *Übermittler*: Hier wird die Mail-Adresse der Person, die den Literatureintrag übermittelt hat, festgehalten. Stammen die Daten der Literaturangabe von einer anderen Person, sollten Orginalautor und die eigene Adresse separat in einem Feld angegeben werden. Der Eintrag sollte dann auch ein *%O*-Feld enthalten, in dem auf die zitierte Quelle verwiesen wird.

%T **titel** - *Titel* des Eintrags: Besteht ein Titel aus zwei Teilen, die nicht durch ein Satzzeichen getrennt sind, dann sollte ein Komma zwischen diese eingefügt werden.

%V **volume** - *Band*: In diesem Feld steht der Name oder die Nummer des Bandes, in dem der Eintrag erschienen ist. Dies kann ein Journalband oder ein Band in einer Serie von Büchern sein. Hat die Angabe zwei oder mehr Teile, dann sind diese durch Kommata zu trennen.

%x + **extract** - *Auszug*: Der Auszug besteht aus Zitaten, die direkt aus dem Abstrakt der Literaturangabe stammen.

%y * **organizational affiliation** - *Organisationen*, denen der Autor zugehört bzw. die Autoren angehören: Diese Angaben sind dem Dokument zu entnehmen.

%z **reftype** - *Eintragstyp*: Dieses Feld muß immer in der ersten Zeile einer Literaturangabe stehen. Die Groß- und Kleinschreibung der Eintragstypen ist signifikant.

Die Felder, die mehrfach auftreten können (Autor, Anmerkung zum Autoren, Herausgeber, Anmerkung zum Herausgeber und organisatorische Zugehörigkeit), ermöglichen es, Informationen über Autoren und Herausgebern mit ihren Namen zu verschachteln. Die Reihenfolge dieser Felder ist somit von Bedeutung.

2.2.4 Eintragstypen

Für jeden Eintragstyp ist eine bestimmte Menge von Informationen *zwingend* erforderlich. Außerdem gibt es für die einzelnen Typen Angaben, die *erwartet* werden bzw. die *optional* sind. Bei *erwarteten* Angaben sollte man versuchen, diese zu bestimmen. Zu diesen Informationen können weitere hinzugefügt werden, wenn man dies für nützlich hält.

Im folgenden werden die Eintragstypen, die in *refdbms* vorhanden sind, beschrieben.

Article ein Artikel, der in einem Journal oder in einem Magazin veröffentlicht wurde, außer wenn die komplette Ausgabe des Journals aus Konferenzberichten besteht (in diesem Fall ist *InProceedings* zu nutzen)
zwingend: Titel, Datum
erwartet: Autor, Journal, Band, Nummer, Seiten, Stichwörter

Book eine Arbeit, die von einem Verlagshaus ausgegeben wurde
zwingend: Titel, Datum, Verlag
erwartet: Autor, Stichwörter
optional: Serie

InBook ein Kapitel oder Abschnitt innerhalb eines Buches
zwingend: Buchtitel, Titel, Datum, Verlag
erwartet: Autor, Serie, Seiten, Stichwörter

InProceedings ein Artikel aus einer Konferenz; ein einzelnes Dokument in einem Journal, das eine Konferenz zusammenfaßt, wird als *Article* betrachtet
zwingend: Titel, Datum
erwartet: Autor, Seiten, Stichwörter, Konferenzname, Konferenzort und
in Journals: Journal, Band, Nummer
sonst: Verlag
optional: Herausgeber

Manual Anleitung oder technische Dokumentation, die erklärt, wie etwas zu nutzen ist
zwingend: Titel, Datum, Verlag
erwartet: Autor, Stichwörter

Miscellaneous Dokumente, die nicht klassifizierbar sind, z.B. Dokumente in einer ungewöhnlichen Form; das Feld *öffentliche Anmerkung* muß eine Beschreibung des Dokumentes enthalten
zwingend: Titel, Datum, öffentliche Anmerkung
erwartet: Autor, Stichwörter

PhDthesis Doktorarbeit; Diplomarbeiten zählen zu *TechReport*
zwingend: Titel, Datum, Autor, Verlag (Universität oder Hochschule, an der die Doktorarbeit geschrieben wurde)
erwartet: Berichtsnummer, Stichwörter

Proceedings eine gesamte Band- oder Journalausgabe, die nur aus Konferenzberichten besteht; dieser Typ ist nur zu nutzen, wenn man sich auf die Ausgabe als Ganzes bezieht; gewöhnlich wird nur der Verlag genannt, auch wenn einige Berichte Herausgeber haben
zwingend: Titel, Datum
erwartet: Autor, Verlag, Stichwörter, Konferenzname, Konferenzort und
in Journals: Journal, Band, Nummer
sonst: Verlag

TechReport fast jedes Dokument, das von einer Universität oder einer Gesellschaft/Firma für die interne Nutzung oder Weiterverbreitung herausgegeben wurde, außer wenn es ein Buch, eine Doktorarbeit oder ein Manual ist
zwingend: Titel, Datum
erwartet: Autor, Verlag, Stichwörter

UnPublished Dokumente, die nur einem eingeschränkten Leserkreis vorbehalten und somit nicht weit verbreitet sind; das Feld *öffentliche Anmerkung* muß eine Beschreibung des Dokumentes enthalten
zwingend: Titel, Datum, öffentliche Anmerkung
erwartet: Autor, Stichwörter

Jeder Eintrag muß einen Eintragstyp, ein Schlüsselwort und einen Übermittler haben. Außerdem ist es immer nützlich, wenn der Eintrag auch die Orte, an denen er zu finden ist, Stichwörter, eine Zusammenfassung und eine private Anmerkung enthält.

2.2.5 Abkürzungen

Refdbms erlaubt Abkürzungen in einer Literaturangabe. Eine Abkürzung ist eine Folge von Buchstaben und Ziffern, die mit einem Punkt endet. Einige Abkürzungen wie `Jr`, `1st`, `2nd` und `3rd` werden ohne abschließenden Punkt geschrieben. Bei Abkürzungen ist die Groß-/Kleinschreibung signifikant. Die ausgeschriebene Form, in die eine Abkürzung bei der Ausgabe des Eintrags umgewandelt wird, ist in speziellen Dateien enthalten. Standardmäßig gibt es 2 Dateien, in denen die Abkürzungen definiert sind:

- Eine Datei enthält die vollständig ausgeschriebene Form der Abkürzung,
- die andere Datei enthält die teilweise ausgeschriebene Form der Abkürzung (dies ist sinnvoll, wenn das Literaturverzeichnis nicht so viel Platz in Anspruch nehmen soll).

Viele Abkürzungen sind für bestimmte Felder typisch, z.B. wird `CACM.` nur im Feld *journal* zu `Communications of the ACM` erweitert.

2.3 Die *NCSTRL*-Literaturdatenbank

NCSTRL [NCSa, NCSb] ist eine Datenbank für technische Berichte und nutzt ein geändertes *refer*-Format [Les78]. Eine *NCSTRL*-Datei sieht folgendermaßen aus:

NCSTRL-Datei	::=	*Eintrag Leerzeile* {*Eintrag Leerzeile*}
Eintrag	::=	*Feld* {*Feld*}
Feld	::=	'%' *Buchstabe Freiraum Wert Zeilenumbruch*
Freiraum	::=	*Leerzeichen* {*Leerzeichen*}

Jedes Feld beginnt in einer neuen Zeile mit '%', gefolgt von dem Buchstaben, der das entsprechende Feld kennzeichnet, einem oder mehreren Leerzeichen und dem Wert für dieses Feld. Jeder Literatureintrag, auch der Letzte, wird durch eine Leerzeile beendet.

Im folgenden Abschnitt werden die Felder vorgestellt, die ein technischer Report enthalten kann. Vordefinierte Abkürzungen wie in BibTeX und *refdbms* gibt es in *NCSTRL* nicht.

2.3.1 Felder

Die Felder, die für Literatureinträge in *NCSTRL* genutzt werden können, sind in der folgenden Liste beschrieben. Felder, denen nach dem Kennzeichen der

Buchstabe „n“ folgt, sind *notwendige* Angaben. Ein nachfolgendes „o“ bedeutet, daß dieses Feld *optional* ist. Der Buchstabe „e“ hinter dem Feldkennzeichen steht für eine *empfohlene* Information.

%A(n) **author** - *Name des Autors*: Jeder Autor ist in einem separaten *%A*-Feld anzugeben. Für den Namen ist die Form `Nachname, Vorname` zu nutzen, ohne die Angabe von Titeln oder Organisationen, denen der Autor zugehört.

%T(n) **title** - *Titel* des Berichts: Dieses Feld enthält den kompletten Titel mit allen Untertiteln, wenn solche existieren.

%D(n) **date** - *Datum, an dem der Bericht erschienen ist*: Das Format dieses Feldes sollte folgendes Aussehen haben: `Monat Tag, Jahr`. Der `Monat` ist auszuschreiben, der `Tag` muß zwei- und das `Jahr` vierstellig angegeben werden, z.B. `Juli 14, 1994` .

%Z (n) **modification date** - *Datum und Zeit der letzten Änderung des Literatureintrags* (nicht des Berichts): Dieses Feld muß bei jeder Änderung des Eintrags aktualisiert werden. Das Format sieht folgendermaßen aus: `Wochentag, Datum Zeit`. Der `Wochentag` ist in englisch mit den ersten drei Buchstaben anzugeben. Das `Datum` hat die Form `Tag Monat Jahr`. Der `Tag` muß 1- oder 2-, das `Jahr` 2-stellig sein. Für den `Monat` ist die englische Form, begrenzt auf 3 Buchstaben, anzugeben. Die `Zeit` ergibt sich aus der Uhrzeit und der Zeitzone, wobei immer die Greenwicher Zeit verwendet werden sollte (`GMT`). Weitere Angaben können dem RFC 822 [RFCb] entnommen werden. Der Wert dieses Feldes kann z.B. folgendermaßen aussehen: `Mon, 28 Aug 95 18:04:22 GMT`.

%R(n) **report number** - *Nummer* des Berichts.

%I (n) **report issuer** - *Herausgeber* des Berichts: Der Name des Herausgebers sollte voll ausgeschrieben werden.

%U(o) **URL** - *URL* für den Bericht oder für ein Dokument, das den Bericht beschreibt.

%X(e) **abstract** - *Abstrakt* des Berichts: Dieses Feld sollte verwendet werden, da es bei der Suche nach einem Bericht nützlich sein kann. Auch wenn ein Bericht nicht in englisch ist, sollte ein englischer Abstrakt bevorzugt werden.

%K(o) **keywords** - *Stichwörter*, die hilfreich sein können, um den Eintrag später wiederzufinden: Die einzelnen Stichwörter sollten durch ein Komma getrennt oder in separate Zeilen geschrieben werden.

%Y (o) **Computing Reviews categories** - *CR-Kategorien*, denen der Bericht zugeordnet werden kann: Der Index einer CR-Kategorie muß immer angegeben werden, gefolgt vom Namen dieser Kategorie (nicht zwingend erforderlich), wobei dieser die Namen der übergeordneten Kategorien enthalten sollte.

Die Felder für Stichwörter, CR-Kategorien und den Abstrakt können sich über mehrere Zeilen erstrecken.

Kapitel 3

Das Datenbankmodell

Da die Literaturdatenbank auf dem objektorientierten Datenbanksystem O_2 basieren soll, wird ein objektorientiertes Datenbankmodell (OODM) benötigt. Deshalb werden zunächst die Konzepte dieses Modells vorgestellt. Im zweiten Abschnitt werden die Literaturdatenbanken, die im Kapitel 2 vorgestellt wurden, miteinander verglichen. Dabei erfolgt auch eine detaillierte Beschreibung der Konflikte, die zwischen den einzelnen Literaturdatenbanken auftreten, und es werden Lösungen für diese Konflikte angegeben. Ein Datenbankschema, das eine adäquate Speicherung von BibTeX-, *refdbms*- und *NCSTRL*-Einträgen erlaubt, wird im dritten Abschnitt vorgestellt. Die Probleme, die sich bei der Modellierung ergeben, werden im vierten Abschnitt diskutiert.

3.1 Ein objektorientiertes Datenbankmodell

Zu den Konzepten des objektorientierten Datenbankmodells [Heu97] gehören:

- Basistypen und Typkonstruktoren,
- Objektidentität und Objektzustände,
- Klassen und Typen sowie
- Strukturvererbung.

Basistypen und Typkonstruktoren Jedem Standard-Datentyp wird eine Domäne zugeordnet. Diese stellt die Menge der erlaubten Attributwerte dar, die auch als Instanzen dieses Typs bezeichnet werden. Neben den Standard-Datentypen stehen im objektorientierten Datenbankmodell Typkonstruktoren zur Verfügung. Tupel-, Mengen- und Listenkonstruktor gehören zum Mindestbestandteil eines OODM. Ein Tupelkonstruktor faßt mehrere Komponenten unterschiedlicher Typen zu einem neuen Typ zusammen. Instanz dieses neuen Typs ist

dann ein Tupel bestehend aus Instanzen der zugrundeliegenden Typen. Mengen- und Listenkonstruktor erzeugen aus mehreren Elementen eines zugrundeliegenden Typs einen neuen Typ. Die Instanz des neuen Typs ist eine Menge bzw. Liste aus Instanzen des zugrundeliegenden Typs, wobei zu beachten ist, daß eine Liste im Gegensatz zur Menge Elemente mehrfach beinhalten kann und geordnet ist. Die Typkonstruktoren sind rekursiv anwendbar.

Objektidentität und Objektzustände Das OODM bietet sowohl Werte als auch Objekte. Objekte werden als unstrukturierte Elemente einer abstrakten Menge aufgefaßt. Im Gegensatz dazu sind Werte atomare Elemente von konkreten Mengen (z.B. Domänen der Standard-Datentypen) bzw. strukturierte Elemente, die durch die Verwendung der Typkonstruktoren erzeugt werden. Für jeden Objekttyp existiert eine Menge abstrakter Elemente, die man als abstrakte Domäne bezeichnet. Jedes Element einer solchen Domäne ist ein abstraktes Objekt.

Ein grundlegendes Merkmal eines OODM ist die Trennung des in der Datenbank dargestellten Objektes von seinen Werten: Jedes Objekt hat seine unveränderbare Identität unabhängig von allen Attributwerten, die es beschreiben. Dieses Merkmal wird als Objektidentität bezeichnet. Sie wird durch die abstrakten Objekte dargestellt. Sämtliche Attributwerte eines Objektes ergeben den Zustand des Objektes. Die Zuordnung des Zustandes zu einem abstrakten Objekt erfolgt über eine Zustandsfunktion.

Klassen und Typen Objekte mit ähnlichen Eigenschaften werden in einer Klasse zusammengefaßt. Jeder Klasse wird eine Domäne, eine Objektmenge, ein Zustandstyp und eine Zustandsfunktion zugeordnet. Die Domäne einer Klasse ist entweder abstrakt oder sie ergibt sich aus den Objektmengen anderer Klassen. Sie legt den Objektvorrat, d.h. die Menge der vorgesehenen Objektidentitäten, fest. Ist einer Klasse eine abstrakte Domäne zugeordnet, so wird sie als abstrakte Klasse bezeichnet. Die Domänen der abstrakten Klassen sind paarweise disjunkt. Bei freien Klassen ergibt sich die Domäne aus den Objektmengen anderer Klassen. Die Menge der aktuell vorhandenen Objekte einer Klasse wird Extension genannt. Jeder Klasse ist ein Zustandstyp funktional zugeordnet, der ein Standard-Datentyp, wiederum eine Klasse oder ein komplexer Typ sein kann. Wenn im Zustandstyp eine Klasse enthalten ist, so bezeichnet man sie als Komponentenklasse. Jedem Objekt aus der Extension wird über die Zustandsfunktion ein Element aus der Instanz des Zustandstyps zugeordnet.

Strukturvererbung Die Klassen werden in einer Klassenhierarchie angeordnet. Es existieren zwei Arten von Hierarchien: Spezialisierungen und Generalisierungen. Diese beiden Hierarchien fixieren auf unterschiedliche Weise die Domänen

der freien Klassen. Spezialisierungen ermitteln von gegebenen Domänen der Oberklasse die Domäne der freien Unterklasse, bei Generalisierungen wird die Domäne einer freien Oberklasse aus den gegebenen Domänen der Unterklasse gebildet. Bei der Spezialisierung ist die Domäne der Unterklasse der Durchschnitt der Objektmengen der Oberklassen. Die Domäne der Oberklasse ist bei der Generalisierung die Vereinigung der Extensionen der Unterklassen.

Die Instanz einer freien Klasse ist wie bei abstrakten Klassen als Teilmenge der Domäne definiert. Ein Objekt ist genau einer abstrakten Klasse zugeordnet und kann zu verschiedenen freien Klassen gehören. Es besitzt dann mehrere lokale Zustände, die zusammen den Gesamtzustand ergeben. Weiterhin ist ein Objekt in der Lage, die Klasse wechseln, d.h. es kann sich in eine Unterklasse hinein bzw. aus einer Unterklasse heraus bewegen.

Operationen Ein OODM beinhaltet generische Operationen. Generisch bedeutet, daß diese Operationen auf alle Klassen oder Extensionen anwendbar sind. Sie sind im System fest „verdrahtet“ und somit optimierbar und effizient implementierbar. Mittels generischer Operationen können Werte aus Zuständen von Objekten extrahiert und in einer Ergebnisinstanz gesammelt werden. Mit generischen Operationen lassen sich auch dynamisch neue Zustandstypen erzeugen, neue Objektmengen zu bestehenden Klassen ermitteln bzw. dynamisch neue Klassen generieren.

Weitere objektorientierte Konzepte Zu den Konzepten eines OODM gehören auch diejenigen objektorientierten Prinzipien, die sich mit dem dynamischen Verhalten von Objekten beschäftigen. Dazu zählen vor allem die objekt- und klassenspezifischen Operationen, ihre Vererbung und Einkapselung sowie ihre Darstellung. In einem OODM werden Klassen selbst wieder als Objekte aufgefaßt. Klassen in der Rolle als Objekte einer höheren Metaebene werden als Klassenobjekte bezeichnet, die wiederum einer Klasse angehören. Die Klasse auf der Metaebene wird als Metaklasse bezeichnet. Durch das Konzept der Metaklassen ist es möglich, nicht nur einzelnen Objekten, sondern einer gesamten Klasse Attributwerte zuzuordnen.

Methoden sind klassenspezifische Operationen, die auf alle Objekte einer Klasse anwendbar sind. Sie werden von Ober- zu Unterklassen vererbt. Im Normalfall wird dabei die Signatur und die Implementierung der Methode unverändert übernommen. Es besteht aber auch die Möglichkeit, daß die Implementierung einer Methode bei der Vererbung verändert werden kann. Dieser Prozeß wird als Overriding bezeichnet. Das System wählt dann selbständig zur Laufzeit die passende Implementierung aus, je nachdem, auf welches Objekt aus welcher Klasse diese Methode angewendet wird. Diesen Vorgang nennt man dynamisches Binden.

Die Einkapselung des OODM unterscheidet sich von der Programmiersprachen-Einkapselung. Bei der Programmiersprachen-Einkapselung sind die Attribute und ihre Struktur wie die Implementierungen der Methoden in der Klassendefinition eingekapselt, nur die Methoden-Schnittstellen sind von außen zugänglich. Im Gegensatz dazu sind bei der Datenbank-Einkapselung die Attribute, ihre Struktur und die Schnittstellen der Methoden nach außen hin sichtbar, nur die Methoden-Implementierungen sind versteckt. Das teilweise Aufbrechen der Einkapselung bei OODMen ist notwendig, um deskriptive Anfragen an Strukturen stellen und um Speicherstrukturen sowie Optimierungsverfahren nutzen zu können.

Graphische Darstellung Für die graphische Darstellung im OODM werden folgende Symbole verwendet: Klassen, denen eine abstrakte Domäne zugeordnet ist, werden mit einem schattierten Kreis dargestellt. Ein einfacher Kreis steht für eine freie Klasse. Für die Standard-Datentypen werden Rechtecke verwendet, in denen der Name des des jeweiligen Typs steht. Die vollständige Legende ist in der Abbildung 3.1 angegeben.

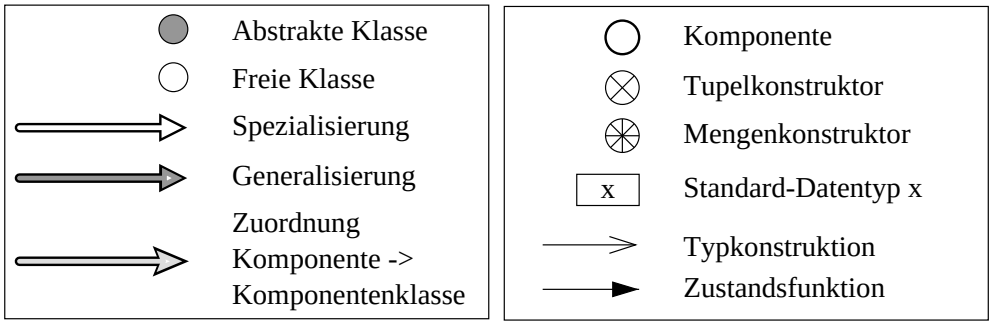


Abbildung 3.1: Legende für die graphische Darstellung des OODM

3.2 Vergleich der Literaturdatenbanken

In den folgenden Abschnitten werden die Formate von BibTeX-, *refdbms*- und *NCSTRL*-Dateien verglichen. Dieser Vergleich bildet die Grundlage für die Integration der Formate in einem gemeinsamen Datenbankschema und wird in zwei Schritten durchgeführt. Im ersten Schritt werden die Eintragstypen der Literaturdatenbanken betrachtet. Danach findet ein Vergleich der Felder, die in den einzelnen Formaten auftreten können, statt.

3.2.1 Vergleich der Eintragstypen

In BibTeX- und *refdbms*-Dateien wird jedem Eintrag ein bestimmter Typ zugeordnet. Bei *NCSTRL*-Einträgen entfällt eine solche Zuordnung, da *NCSTRL*-Dateien

nur technische Berichte enthalten. In der Tabelle 3.1 werden die möglichen Eintragstypen der einzelnen Formate aufgelistet. Vergleichbare Eintragstypen sind in einer Zeile angeordnet.

BibTeX	***refdbms***	***NCSTRL***
Article	Article	
Book	Book	
Booklet		
Conference		
Inbook	InBook	
Incollection		
Inproceedings	InProceedings	
Manual	Manual	
Mastersthesis		
Misc	Miscellaneous	
Phdthesis	PhDthesis	
Proceedings	Proceedings	
Techreport	TechReport	alle Einträge
Unpublished	UnPublished	
weitere Eintragstypen, die nicht standardmäßig sind		

Tabelle 3.1: Vergleich der Eintragstypen

In *refdbms* stehen weniger Eintragstypen zur Verfügung als im BibTeX-Format. Die Einteilung ist somit im BibTeX-Format genauer, denn in *refdbms* fehlen die Eintragstypen *Booklet*, *Conference*, *Incollection* und *Mastersthesis*. Aufgrund der Beschreibungen der einzelnen Typen kann man folgende Aussagen treffen:

- Einträge des Typs *Booklet* werden in *refdbms* dem Eintragstyp *Miscellaneous* zugeordnet.
- Der Eintragstyp *Conference* wird im BibTeX-Format aus Kompatibilitätsgründen unterstützt und stimmt mit dem Typ *InProceedings* überein. *Conference*-Einträge sind somit im *refdbms*-Format in *InProceedings* enthalten.
- Einträge, die im BibTeX-Format den Typ *Incollection* haben, erhalten in *refdbms* den Eintragstyp *InBook*.
- Diplomarbeiten (*Mastersthesis*) werden in *refdbms* als *TechReport* betrachtet.

NCSTRL enthält nur technische Berichte. Diplom- und Doktorarbeiten (*Phdthesis*) kann man ebenfalls als technische Berichte ansehen, da sie von einer Universität veröffentlicht werden. Referenzen, denen ein anderer Eintragstyp zugeordnet wird, dürfen nicht in einer *NCSTRL*-Datei vorkommen.

3.2.2 Vergleich der Felder

In diesem Abschnitt werden die Felder der einzelnen Literaturdatenbanken verglichen. Zwischen diesen Felder kann es zu Konflikten kommen. Im folgenden werden drei Konfliktarten unterschieden:

- *Beschreibungskonflikte* (**B**) treten auf, wenn semantisch gleiche Attribute in verschiedenen Literaturdatenbanken unterschiedlichen Feldern zugeordnet werden. Es ist aber auch der umgekehrte Fall denkbar, d.h. daß semantisch unterschiedlichen Werten gleiche Felder zugewiesen werden.

- Ein *semantischer Konflikt* (**Se**) liegt vor, wenn ein Feld einer Literaturdatenbank Teil eines Feldes einer anderen ist.

  Im folgenden Beispiel tritt ein semantischer Konflikt auf: In der Literaturdatenbank *A* sind die Felder *day*, *month* und *year* vorhanden. Die Literaturdatenbank *B* besitzt ein Feld *date*, das ein Datum in der Form `Tag Monat Jahr` enthält. Die Felder *day*, *month* und *year* stehen in einem semantischen Konflikt mit dem Feld *date*, da sie jeweils einen Teil von *date* darstellen.

- Ein *Strukturkonflikt* (**St**) liegt vor, wenn sich ein Feld aus mehreren Feldern einer anderen Literaturdatenbank zusammensetzt.

  Im Beispiel für semantische Konflikte tritt ein Strukturkonflikt zwischen dem Feld *date* und den Feldern *day*, *month*, *year* auf, da das Datum aus den Angaben dieser Felder zusammengesetzt wird. Strukturkonflikte und semantische Konflikte stellen somit verschiedene Sichtweisen auf einen Konflikt dar.

In der Tabelle 3.2 sind alle Felder aufgelistet, die sich bei der Vereinigung der Standardfelder des BibTeX-Formats mit den Feldern von *refdbms* und *NCSTRL* ergeben. Die Tabelle enthält Informationen darüber, aus welcher Literaturdatenbank ein Feld stammt und welche Konflikte mit anderen Feldern auftreten können. Ein Konflikt wird in der Form `Konfliktart(Feld-Nr., ...)` angegeben. Für die Konfliktart wird das jeweilige Kürzel verwendet, gefolgt von den Nummern der Felder, mit denen das Feld in Konflikt steht.

Nr.	**Feld**	BibTeX	***refdbms***	***NCSTRL***	**Konflikt**
1	address	√			**Se**(34,38)
2	abstract	√[1]		√	**B**(17)
3	annote	√			**B**(35)
4	author	√	√	√	

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite

Nr.	Feld	BibTeX	*refdbms*	*NCSTRL*	Konflikt
5	author note		√		
6	booktitle	√	√		
7	bibtex key		√		
8	chapter	√			**Se**(29)
9	conference name		√		**Se**(30)
10	conference location		√		
11	computing reviews categories			√	
12	crossref	√			
13	date		√	√	**St**(26,47)
14	edition	√			**Se**(29)
15	editor	√	√		
16	editor note		√		
17	extract		√		**B**(2)
18	ftp	√[1]			**Se**(25,45)
19	howpublished	√			**Se**(34)
20	institution	√			**Se**(34,38)
21	ISDN/ISSN		√		
22	journal	√	√		
23	key	√			
24	keywords	√[1]	√	√	
25	location		√		**Se**(18,45)
26	month	√			**Se**(13)
27	modification date			√	
28	note	√			**B**(35)
29	number	√	√		**Se**(8,14,37)
30	organization	√			**Se**(9,34)
31	organizational affiliation		√		
32	pages	√	√		
33	public note		√		**B**(28)
34	publisher	√	√		**St**(1,34) **B**(38) **Se**(19,20,30,39)
35	private note		√		**B**(3)
36	reftype		√		
37	report number		√	√	**Se**(29)

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite					
Nr.	**Feld**	BIBTEX	***refdbms***	***NCSTRL***	**Konflikt**
38	report issuer			√	**St**(1,34) **B**(34) **Se**(20,39)
39	school	√			**Se**(34,38)
40	series	√	√		
41	submitter		√		
42	tag		√		
43	title	√	√	√	
44	type	√			
45	URL	√[1]		√	**Se**(18,25)
46	volume	√	√		
47	year	√			**Se**(13)

Tabelle 3.2: Felder der Datenbank

Die in der Tabelle 3.2 angegebenen Konflikte werden im nächsten Abschnitt genauer beschrieben.

3.2.3 Konflikte zwischen den Feldern

In diesem Abschnitt erfolgt eine detaillierte Beschreibung der Konflikte, die bei der Integration von BIBTEX-, *refdbms*- und *NCSTRL*-Literaturdatenbanken zwischen den Feldern auftreten. Für die einzelnen Konflikte werden Lösungen vorgestellt, da die Bereinigung dieser Konflikte eine Grundvoraussetzung für den Entwurf des Datenbankschemas ist.

Bei den festgestellten Beschreibungskonflikten tritt nur der Fall auf, daß die Felder y_1 und y_2 verschiedener Literaturdatenbanken semantisch gleiche Werte enthalten. Gelöst werden kann dieses Problem, indem nur das Feld y_1 in die resultierende Datenbank übernommen wird. Die Angaben im Feld y_2 werden dem Feld y_1 zugeordnet. Zwischen den Feldern der einzelnen Formate treten folgende Beschreibungskonflikte auf:

- **abstract ⇔ extract**
  Konflikt: Die Felder *abstract* und *extract* haben die gleiche Bedeutung.
  Lösung: Das Feld *abstract* wird in die resultierende Datenbank übernommen. Die Angaben in *extract* werden dem Feld *abstract* zugeordnet.

[1] normalerweise kein BIBTEX-Standardfeld

- **annote ⇔ private note**
  Konflikt: Das Feld *annote* entspricht *private note*.
  Lösung: Die Angabe in *private note* wird dem Feld *annote* zugeordnet.

- **note ⇔ public note**
  Konflikt: Die Felder *note* und *public note* haben die gleiche Bedeutung.
  Lösung: Die Angabe in *public note* wird dem Feld *note* zugeordnet.

Ein Strukturkonflikt liegt vor, wenn sich ein Feld x aus der Zusammenfassung mehrerer Felder $y_1, \ldots, y_n$ einer anderen Literaturdatenbank ergibt. Für diesen Konflikt sind zwei Lösungen denkbar: Entweder wird dem Feld x die Zusammenfassung der Felder $y_1, \ldots, y_n$ zugeordnet, oder das Feld x wird in seine Bestandteile aufgespalten, die dann den Feldern $y_1, \ldots, y_n$ zugewiesen werden. Die zweite Variante kann nur angewendet werden, wenn die Auftrennung des Feldes x in die Bestandteile $y_1, \ldots, y_n$ eindeutig ist. Folgende Strukturkonflikte liegen vor:

- **date ⇔ month**, **year**
  Konflikt: Die Angaben *month* und *year* sind im *refdbms*- und *NCSTRL*-Format im Feld *date* enthalten.
  In *refdbms* wird das Datum in der Form `Tag Monat Jahr` angegeben (der `Monat` wird durch die ersten drei Buchstaben spezifiziert). In *NCSTRL* erfolgt die Datumsangabe in der Form `Monat Tag, Jahr` (der `Monat` wird ausgeschrieben).
  Lösung: Das Feld *date* wird in seine einzelnen Bestandteile aufgespalten, und es wird ein zusätzliches Feld *day* eingeführt: `Tag`, `Monat` und `Jahr` werden dann den entsprechenden Feldern *day*, *month* und *year* zugeordnet.

- **publisher** (*refdbms*) ⇔ **publisher** (BibTeX), **address**
  Konflikt: Im BibTeX-Format enthält das Feld *publisher* nur den Verlagsnamen. Die Angaben in *publisher* und *address* ergeben im *refdbms*-Format zusammen das Feld *publisher*.
  Lösung: Im *refdbms*-Format wird das Feld *publisher* in die Bestandteile `Verlagsname` und `Adresse` aufgespalten. Diese Aufspaltung basiert auf folgender Annahme: Die `Adresse` wird vom `Verlagsnamen` durch ein Komma getrennt und der `Verlagsname` selber enthält kein Komma.
  Somit können alle Zeichen nach dem ersten Komma dem Feld *address* zugeordnet werden. Das Feld *publisher* enthält dann auch im *refdbms*-Format nur noch den `Verlagsnamen`.

- **report issuer ⇔ publisher** (BibTeX), **address**
  Konflikt: Die Angaben in *publisher* und *address* ergeben zusammen das Feld *report issuer*.

Lösung: Die Angabe in *report issuer* liegt ebenfalls in der Form `Verlagsname, Adresse` vor. Alle Zeichen nach dem ersten Komma werden dem Feld *address* zugeordnet, der `Verlagsname` wird dem Feld *publisher* zugewiesen.

In allen Fällen wurde also der zweite Lösungsansatz gewählt, da beim Export der Einträge in das BibTeX-Format die spezielleren Felder benötigt werden. Beim Datum kommt noch ein weiterer Grund hinzu: Durch das unterschiedliche Format der Datumsangaben in *refdbms* und *NCSTRL* muß ohnehin eine Transformation in ein gemeinsames Format vorgenommen werden.

Ein semantischer Konflikt tritt auf, wenn das Feld x im Feld y einer anderen Literaturdatenbank enthalten ist. Für die Lösung dieses Konflikts gibt es mehrere Lösungen:

- Unter bestimmten Bedingungen wird das Feld y dem Feld x zugeordnet.
- Das Feld x wird aus dem Feld y herausgefiltert.
- Unter bestimmten Bedingungen wird das Feld x aus dem Feld y herausgefiltert.

Die Lösung eines semantischen Konflikts gestaltet sich in vielen Fällen schwierig. In einigen Fällen ist sie sogar unmöglich. Die folgenden semantischen Konflikte treten zwischen den Feldern auf:

- **address ⇔ publisher**
  Konflikt: Die Adresse eines Verlages ist im *refdbms*-Format im Feld *publisher* enthalten.
  Lösung: Die Adresse wird aus dem Feld *publisher* herausgefiltert und *address* zugeordnet.
- **address ⇔ report issuer**
  Konflikt: Die Adresse eines Verlages ist im *NCSTRL*-Format im Feld *report issuer* enthalten.
  Lösung: Die Adresse wird aus dem Feld *report issuer* herausgefiltert und *address* zugewiesen.
- **chapter ⇔ number**
  Konflikt: Die Kapitelangabe ist im *refdbms*-Format in *number* enthalten.
  Lösung: Die einzelnen Angaben im Feld *number* sind durch Kommata voneinander getrennt. Wenn eine dieser Angaben nur aus Ziffern besteht und der Eintragtyp *InBook* ist (bei anderen Eintragstypen treten Kapitelangaben üblicherweise nicht auf), dann wird sie dem Feld *chapter* zugewiesen.

- **conference name ⇔ organization**
  Konflikt: Die Angaben in *conference name* entsprechen teilweise denen im Feld *organization*.
  Lösung: Da keine Regel aufstellbar ist, wann die Angaben übereinstimmen könnten, wird so getan, als ob kein Zusammenhang zwischen den Feldern besteht.

- **edition ⇔ number**
  Konflikt: Die Auflage ist im *refdbms*-Format im Feld *number* in der Form `xxx edition` enthalten, wobei `xxx` für die Auflage steht.
  Lösung: Enthält *number* die Angabe `xxx edition`, dann wird sie in *number* gelöscht und `xxx` wird dem Feld *edition* zugeordnet.

- **ftp ⇔ location**
  Konflikt: Die Angaben in *ftp* sind im Feld *location* enthalten.
  Lösung: Alle Angaben in *location*, die mit `ftp` beginnen, werden dem Feld *ftp* zugeordnet.

- **ftp ⇔ URL**
  Konflikt: Die Angaben in *ftp* sind im *NCSTRL*-Format im Feld *URL* enthalten.
  Lösung: Alle Angaben in *URL*, die mit `ftp` beginnen, werden dem Feld *ftp* zugeordnet.

- **howpublished ⇔ publisher**
  Konflikt: Im BibTeX-Format wird bei den Eintragstypen *Misc* und *Booklet* das Feld *howpublished* statt *publisher* genutzt, die Angabe in *howpublished* ist in *refdbms* im Feld *publisher* enthalten.
  Lösung: Die Angabe in *publisher* hat die Form `Name, Adresse`. Bei *refdbms*-Einträgen vom Typ *Miscellaneous* wird der `Name` dem Feld *howpublished* zugewiesen, die `Adresse` wird dem Feld *address* zugeordnet.

- **institution ⇔ publisher**
  Konflikt: Im BibTeX-Format wird beim Eintragstyp *TechReport* das Feld *institution* statt *publisher* (*refdbms*) genutzt, die Angabe in *institution* ist in *refdbms* im Feld *publisher* enthalten.
  Lösung: Die Angabe in *publisher* hat das Format `Name, Adresse`. Ist der Eintragstyp *TechReport*, dann wird der `Name` dem Feld *institution* zugewiesen, die `Adresse` wird dem Feld *address* zugeordnet.

- **institution ⇔ report issuer**
  Konflikt: Die Angabe in *institution* ist im Feld *report issuer* enthalten.

Lösung: Das Format der Angabe in *report issuer* ist `Name, Adresse`. Der `Name` wird dem Feld *institution* und die `Adresse` dem Feld *address* zugewiesen.

- **location ⇔ URL**
  Konflikt: Die Angaben in *URL* sind im Feld *location* enthalten.
  Lösung: Alle Angaben in *location*, die mit `http` beginnen, werden dem Feld *URL* zugeordnet (`ftp`-Angaben werden dem Feld *ftp* zugewiesen).

- **month ⇔ date**
  Konflikt: Die *month*-Angabe ist im *refdbms*- und *NCSTRL*-Format im Feld *date* enthalten.
  Lösung: Der Monat wird aus dem Feld *date* herausgefiltert und *month* zugeordnet.

- **number ⇔ report number**
  Konflikt: Die technische Berichtsnummer wird im BibTeX-Format im Feld *number* angegeben (beim Eintragstyp *TechReport*).
  Lösung: Bei BibTeX-Einträgen vom Typ *TechReport* wird die Angabe in *number* dem Feld *report number* zugeordnet.

- **organization ⇔ publisher**
  Konflikt: Im BibTeX-Format wird beim Eintragstyp *Manual* das Feld *organization* statt *publisher* genutzt, die Angabe in *organization* ist im *refdbms*-Format im Feld *publisher* enthalten.
  Lösung: Die Angabe in *publisher* liegt in der Form `Name, Adresse` vor. Bei *refdbms*-Einträgen vom Typ *Manual* wird der `Name` dem Feld *organization* zugewiesen, die `Adresse` wird dem Feld *address* zugeordnet.

- **school ⇔ publisher**
  Konflikt: Im BibTeX-Format wird bei den Eintragstypen *Phdthesis* bzw. *Mastersthesis* das Feld *school* statt *publisher* genutzt, die Angabe in *school* ist im *refdbms*-Format im Feld *publisher* enthalten.
  Lösung: Die Angabe in *publisher* liegt in der Form `Name, Adresse` vor. Bei *refdbms*-Einträgen vom Typ *PhDthesis* wird der `Name` dem Feld *school* zugewiesen, die `Adresse` wird dem Feld *address* zugeordnet.
  Problem: In *refdbms* sind Diplomarbeiten im Typ *Techreport* enthalten. Da nicht erkennbar ist, welcher technische Report eine Diplomarbeit ist, wird einer Diplomarbeit in *refdbms* das Feld *institution* zugeordnet (wie bei jedem *TechReport*).

- **school ⇔ report issuer**
  Konflikt: Beim Eintragstyp *Mastersthesis* wird im BibTeX-Format *school* statt *publisher* genutzt, die Angabe in *school* ist im Feld *report issuer* enthalten.
  Problem: Diplomarbeiten sind in *NCSTRL* nicht von anderen technischen Berichten zu unterscheiden. Sie müssen deshalb wie alle Einträge in *NCSTRL* behandelt werden, d.h. die Angabe in *report issuer* wird den Feldern *institution* und *address* zugeordnet.

- **year ⇔ date**
  Konflikt: Die *year*-Angabe ist im *refdbms*- und *NCSTRL*-Format im Feld *date* enthalten.
  Lösung: Das Jahr wird aus dem Feld *date* herausgefiltert und *year* zugeordnet.

3.3 Entwurf des Datenbankschemas

In diesem Abschnitt wird ein objektorientiertes Datenbankschema vorgestellt, das eine adäquate Speicherung von Einträgen der im Abschnitt 2 beschriebenen Literaturdatenbanken erlaubt. Die Beschreibung des Schemas erfolgt schrittweise. In den einzelnen Schritten wird dargelegt, warum bestimmte Eigenschaften so und nicht anders modelliert wurden bzw. welche Alternativen möglich sind.

3.3.1 Darstellung von Personen

Da das Format für Autoren- und Herausgeberangaben in den einzelnen Literaturdatenbanken variiert, ist eine Vereinheitlichung dieser Angaben notwendig. Bei allen zugrundeliegenden Literaturdatenbanken kann eine Namensangabe in vier Bestandteile getrennt werden: Vorname, *von*-Teil, Nachname und *Jr*-Teil (siehe Unterabschnitt 2.1.7.1 und 2.2.3). Bei der Beschreibung der *NCSTRL*-Namensangaben in Abschnitt 2.3.1 wurden *von*- und *Jr*-Teil zwar nicht spezifiziert, sie sind aber wie folgt bestimmbar: Alle kleingeschriebenen Wörter am Anfang des Nachnamens werden dem *von*-Teil zugeordnet, die Angabe `Jr` bzw. `Sr` am Ende des Nachnamens wird als *Jr*-Teil betrachtet. Die 4 Bestandteile eines Personennamens sollten getrennt gespeichert werden, damit

- der Export in die verschiedenen Formate einfach und
- eine flexible Suche nach Personen möglich ist.

Autoren, Herausgeber und Erfasser müssen eindeutig identifizierbar sein. Deshalb wird eine Klasse *Person* eingeführt. Attribute dieser Klasse sind die Bestandteile

einer Namensangabe. Um die Eindeutigkeit zu gewährleisten, wird ein Schlüssel für die Klasse definiert, der sich aus den 4 Attribute zusammensetzt.

Unterschiedlichen Autoren werden nur verschiedene Objekte zugeordnet, wenn die Angaben des Nutzers eindeutig sind. Gibt man z.B. die unterschiedlichen Autoren `Marc H. Scholl` und `Michel Scholl` mit `M. Scholl` an, dann kann das System nicht feststellen, daß es sich um verschiedene Autoren handelt. In diesem Fall werden die beiden Autoren durch dasselbe Objekt dargestellt.

In der Abbildung 3.2 ist die Klasse *Person* graphisch dargestellt.

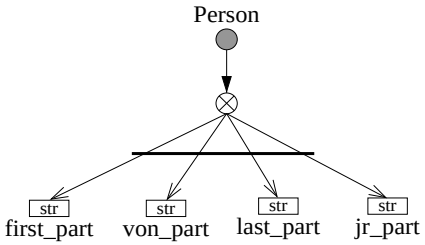


Abbildung 3.2: Klasse Person

3.3.2 Darstellung von Feldwerten

Im BibTeX-Format bestehen Felder aus einem Namen und einem Wert. In *refdbms* und *NCSTRL* wird statt des Namens ein Buchstabe angegeben, der das Feld kennzeichnet. Der Wert eines Feldes kann:

- eine Namensangabe (Autor oder Herausgeber),
- ein Querverweis (BibTeX) oder
- eine Abkürzung sein (BibTeX und *refdbms*),
- durch eine Ziffernfolge oder
- als Zeichenkette (die keine Abkürzung darstellt) angegeben sein bzw.
- sich aus Abkürzungen, Ziffernfolgen und Zeichenketten zusammensetzen.

Die Darstellung von Namensangaben und Querverweisen wird später diskutiert. Ziffernfolgen werden nicht als numerischer Wert, sondern als Zeichenkette festgehalten, da dadurch die Ausgabe solcher Werte einfacher ist. Das Format, in dem ein Feldwert vorliegt, kann sehr stark variieren und hängt vom Erfasser ab. Da gleiche Dokumente unterschiedlicher Erfasser zu einem Eintrag zusammengefaßt

werden sollen, entsteht ein Problem: Will man das Format des Feldwertes erhalten, dann muß der Wert für jeden Erfasser separat gespeichert werden. Dies soll an folgendem Beispiel verdeutlicht werden: Zwei BibTeX-Dateien verschiedener Erfasser enthalten den gleichen Eintrag. In der Datei von Erfasser 1 ist zusätzlich die Abkürzung

```
STOC = " Symposium on the Theory of Computing"
```

definiert. Der Eintrag enthält einen Buchtitel, der von

- Erfasser 1 mit `"Proc. Fifteenth Annual ACM" # STOC` und von
- Erfasser 2 mit `"Proc. Fifteenth Annual ACM Symposium on the Theory of Computing"`

angegeben wurde. Da die Buchtitel identisch sind, würde die separate Speicherung zu einer Datenredundanz führen. Das Problem kann gelöst werden, indem der zusammengesetzte Wert als Zeichenkette gespeichert wird. Die Zeichenkette ergibt sich aus der Verkettung der ausgeschriebenen Form der Abkürzung mit den weiteren Angaben, die der Feldwert enthält. Bei sprachabhängigen Abkürzungen wie z.B. Monatsnamen zeigt sich der Nachteil dieser Lösung: Eine Änderung der Sprache des Eintrags hat keinen Einfluß mehr auf die Abkürzung. Dieser Nachteil kann bei Feldwerten, die eine Abkürzung ohne weitere Angaben enthalten, umgangen werden. In einer BibTeX-Stildatei ist z.B. die Abkürzung `jan` für `January` definiert. Für denselben Eintrag gibt

- Erfasser 1 `month = jan` und
- Erfasser 2 `month = January`

an. In diesem Fall sollte der Feldwert als Abkürzung gespeichert werden. Um Konflikte zu vermeiden, wird folgende Festlegung getroffen: Kommt derselbe Feldwert als Abkürzung und als Zeichenkette vor, dann wird immer die Abkürzung bevorzugt.

Die Angaben in einigen Feldern wie z.B. *annote* werden nicht dem Dokument entnommen, sondern sie hängen vom jeweiligen Erfasser ab und können sich somit auch bei Einträgen, die das gleiche Dokument beschreiben, unterscheiden. Die Werte solcher Felder müssen deshalb für jeden Erfasser separat gespeichert werden.

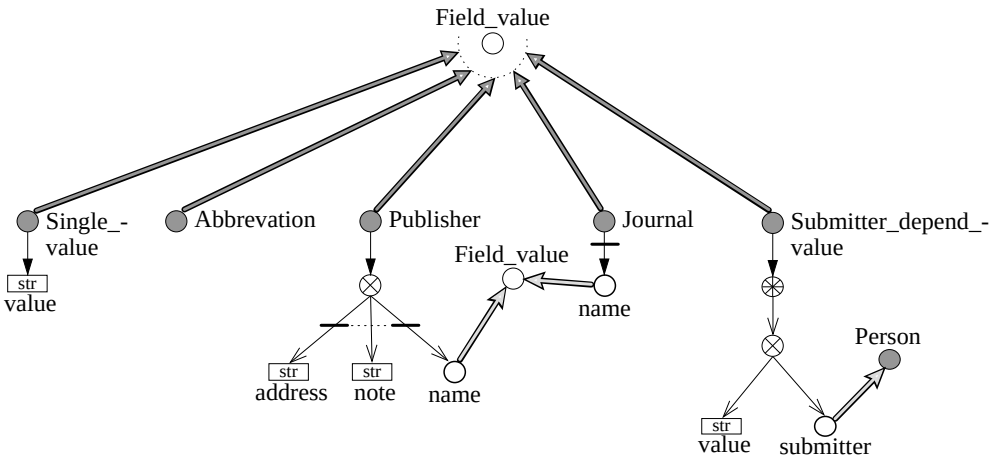


Abbildung 3.3: Klassen für die Darstellung von Feldwerten

Modellierung Verlage und Journals werden als eigenständige Anwendungsobjekte aufgefaßt und deshalb durch eigene Klassen repräsentiert.

Feldwerte, die aus einer Zeichenkette bestehen, werden im folgenden als einfache Werte bezeichnet. Um die Feldwerte beschreiben zu können, werden also Klassen für Abkürzungen, Verlage, Journale, erfasserabhängige und einfache Werte benötigt. In der Abbildung 3.3 sind diese Klassen dargestellt.

Die Klasse *Single_value* enthält Objekte, die einfache Werte darstellen. Die Objekte der Klasse besitzen nur das Attribut *value*. Dieses Attribut ist aber kein Schlüssel für die Klasse *Single_value*, da die einzelnen Werte unabhängig voneinander sind. Verlage werden durch Objekte der Klasse *Publisher* beschrieben. Die Klasse besitzt die Attribute *Name*, *Adresse* und *Anmerkung*. Anmerkungen zu einem Verlag wurden zusätzlich in den Typ der Klasse aufgenommen, obwohl so ein Feld in keiner der zugrundeliegenden Datenbanken existiert. Der Name eines Verlages kann eine Abkürzung bzw. ein einfacher Wert sein. Deshalb wird dem Verlagsnamen ein Objekt aus der Klasse *Field_value* zugeordnet. Verlagsname und -adresse bilden zusammen den Schlüssel für die Klasse *Publisher*. Objekte der Klasse *Journal* besitzen das Attribut *Name*. Der Journalname ist ein Objekt aus der Klasse *Field_value*, da er entweder durch eine Abkürzung oder einen einfachen Wert angegeben werden kann. Für erfasserabhängige Felder wird die Klasse *Submitter_depend_value* eingeführt. Der Typ dieser Klasse ist eine Menge von Tupeln, die aus einem Erfasser und dem zugehörigen Wert bestehen. Der Erfasser ist ein Personen-Objekt und stellt den Schlüssel für ein erfasserabhängiges Attribut dar. Der zugehörige Wert kann als Zeichenkette festgehalten werden, da erfasserabhängige Felder gewöhnlich nicht nur aus einer Abkürzung bestehen. Die Klasse *Abbrevation* wird im nächsten Abschnitt gesondert betrachtet.

Die Klassen *Single_value*, *Abbrevation*, *Publisher*, *Journal* und *Submitter_depend_value*, denen jeweils eine abstrakte Domäne zugeordnet wird, haben die gemeinsame Oberklasse *Field_value*. Die Domäne der Klasse *Field_value* ergibt sich aus der Vereinigung der Objektmengen dieser Klassen.

3.3.2.1 Die Klasse für Abkürzungen

In BibTeX und *refdbms* können Abkürzungen verwendet werden. Sie werden durch die Klasse *Abbrevation* repräsentiert. Die ausgeschriebene Form einer Abkürzung hängt vom Stil ab, der bei der Erstellung eines Literaturverzeichnisses verwendet wird. Sie kann in der Sprache und der Länge variieren. Unterschiedliche Längen bei der Expansion einer Abkürzung kommen dadurch zustande, daß in einigen Stilen Teile der ausgeschriebenen Form abgekürzt werden, um das Literaturverzeichnis kompakter zu gestalten. Zum Beispiel wird die Abkürzung `jan` in der Kurzform zu `Jan.` und in der Langform zu `January` erweitert. Der Wert einer Abkürzung muß somit in einem mengenwertigen Attribut gespeichert werden. In den einzelnen Tupeln dieser Menge wird die Kurzform und die Langform in einer bestimmten Sprache festgehalten. Dadurch kann eine Abkürzung abhängig von der Sprache des Eintrags und vom gewählten Stil expandiert werden.

In *refdbms* werden viele Abkürzungen nur in bestimmten Feldern erweitert. Deshalb muß für jede Abkürzung eine Liste von Feldern festgehalten werden, in denen sie expandiert wird. Um in diesem Punkt eine Vereinheitlichung der drei Formate zu erreichen, wird folgendes gemacht: Die Klasse *Abbrevation* erhält die booleschen Attribute *bibtex*, *refdbms* und *ncstrl*, die „wahr“ sind, wenn beim Import von Einträgen im entsprechenden Format die Feldliste ausgewertet werden soll. Ansonsten wird die Abkürzung in allen Felder ausgeschrieben.

In der Abbildung 3.4 ist die Klasse *Abbrevation* dargestellt. Schlüssel für die Klasse ist der Name der Abkürzung.

3.3.3 Modellierung der Einträge

Bei der Modellierung von Einträgen tritt das Problem auf, das jeder Anwender für die einzelnen Literaturtypen seine eigenen Felder wählen kann. Damit sind die Daten „semistrukturiert“, d.h. zwei Einträge desselben Typs können unterschiedliche Felder haben, die nicht vorhersehbar sind. Aus diesem Grund kann ein Eintrag nicht einer vordefinierten Klasse, die als Attribute die möglichen Felder des entsprechenden Eintragstyps besitzt, zugeordnet werden. Da zur Zeit noch kein Datenbankmodell für semistrukturierte Daten existiert, muß es mit dem objektorientierten Modell simuliert werden.

Die Literatureinträge kann man wie folgt modellieren: Alle Einträge werden einer Klasse zugeordnet. Diese Klasse besitzt ein Attribut für den Eintragstyp und ein

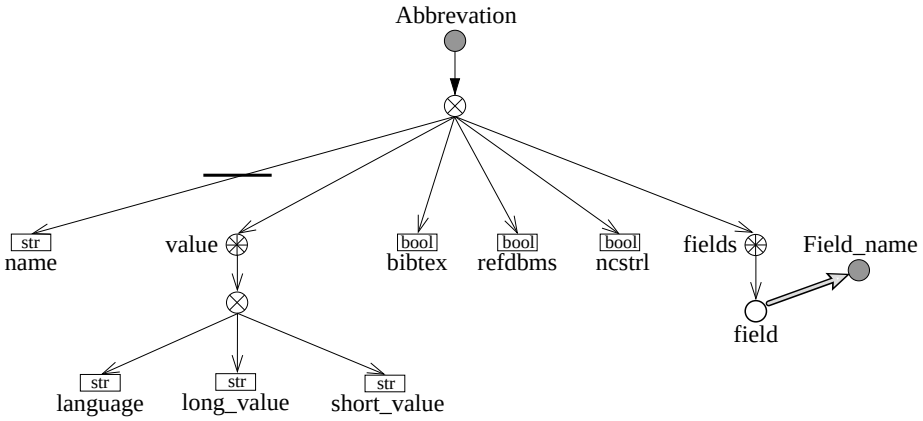


Abbildung 3.4: Klasse Abbrevation

mengenwertiges Attribut, in dem alle Felder des Eintrags festgehalten werden.

Der Nachteil dabei ist, daß die Methoden nicht abhängig vom Eintragstyp implementiert werden können. Eine eintragsabhängige Implementierung ist aber vor allem bei Ausgabe- und Export-Methoden von Vorteil. Deshalb sieht die Lösung folgendermaßen aus: Es wird die Klasse *Entry* eingeführt, die ein mengenwertiges Attribut für die Felder besitzt. Die einzelnen Eintragstypen werden durch Unterklassen von *Entry* dargestellt. Eine weitere Unterklasse repräsentiert BibTeX-Referenzen mit selbstdefinierten Eintragstypen. Diese Klasse muß ein Attribut für den Eintragstyp der Referenz besitzen. Die Abbildung 3.5 zeigt die Klassen, die für die einzelnen Eintragstypen definiert werden.

Die Domänen der Klassen für die verschiedenen Eintragstypen werden durch die Extension der Klasse *Entry* bestimmt. Eine Vorstellung der weiteren notwendigen Attribute der Klasse *Entry* erfolgt im nächsten Abschnitt.

3.3.3.1 Die Oberklasse für Einträge

Darstellung von Autoren und Herausgebern Für einen Eintrag können mehrere Autoren und Herausgeber spezifiziert werden. Bei *NCSTRL*-Einträgen entfällt die Angabe von Herausgebern, da dafür kein Feld vorgesehen ist. *Refdbms* bietet die Möglichkeit, Anmerkungen zu einem Autoren bzw. Herausgeber und Organisationen, denen ein Autor angehört, anzugeben. Diese Angaben müssen somit zusammen mit dem Autoren bzw. Herausgeber abgespeichert werden. Sie sind dem Dokument zu entnehmen und somit erfasserunabhängig. Die Klasse *Entry* enthält deshalb je ein mengenwertiges Attribut für die Autoren und die Herausgeber eines Dokumentes. Die einzelnen Tupel dieser Mengen enthalten neben dem Autoren bzw. Herausgeber die Informationen zu diesen Personen.

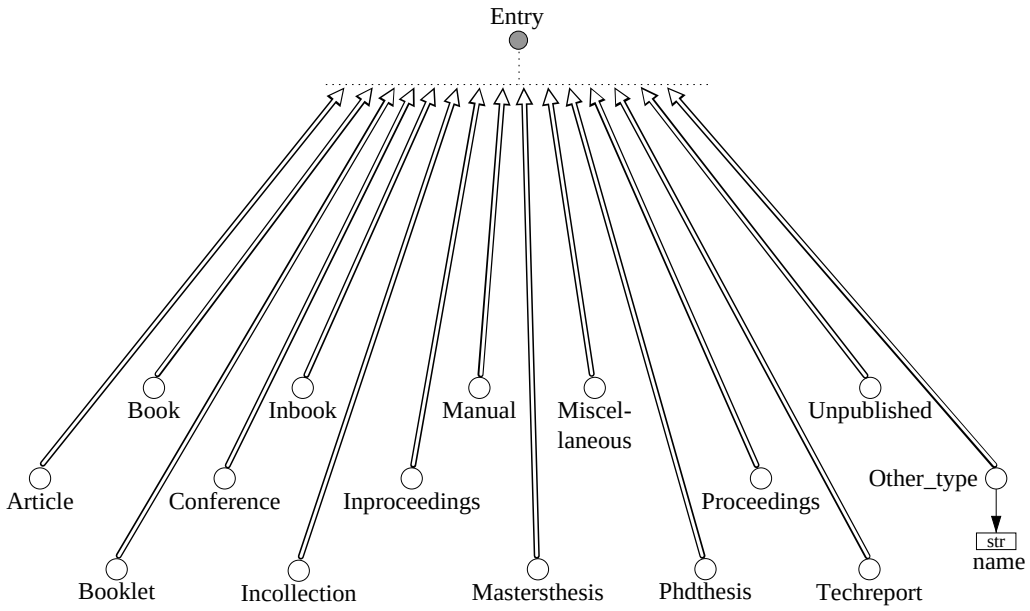


Abbildung 3.5: Klassen für die verschiedenen Eintragstypen

Im BibTeX-Format ist es möglich, die Autoren- und die Herausgeberliste mit der Angabe `and others` (siehe Abschnitt 2.1.7.2) abzukürzen. Dies kann aber nicht im mengenwertigen Autoren- bzw. Herausgeberattribut festgehalten werden. Aus diesem Grund muß die Klasse *Entry* zwei boolesche Attribute haben, die „wahr“ sind, wenn die Autoren- bzw. die Herausgeberliste mit `and others` beendet wurde.

Darstellung von Querverweisen In BibTeX-Literaturdatenbanken können Querverweise enthalten sein. Diese werden in Einträgen verwendet, die Teil eines Buches bzw. Artikel in einem Konferenzband sind, um auf den entsprechenden Buch- bzw. Konferenzband-Eintrag zu verweisen. Der Querverweis eines Eintrags sollte daher auch bei unterschiedlichen Erfassern übereinstimmen und kann somit durch ein Komponentenobjekt dargestellt werden.

Darstellung von Schlüsselwörtern In BibTeX- und *refdbms*-Literaturdatenbanken enthält jeder Eintrag ein Schlüsselwort. Es ist innerhalb der Literaturdatenbank eindeutig, da es für die Identifizierung der Einträge genutzt wird. Gleiche Einträge verschiedener Erfasser können aber unterschiedliche Schlüsselwörter haben. Ein Schlüsselwort muß somit zusammen mit dem jeweiligen Erfasser in einem mengenwertigen Attribut gespeichert werden. Die einzelnen Tupel dieses Attributs stellen einen Schlüssel für den Eintrag dar. Die Nutzung dieses

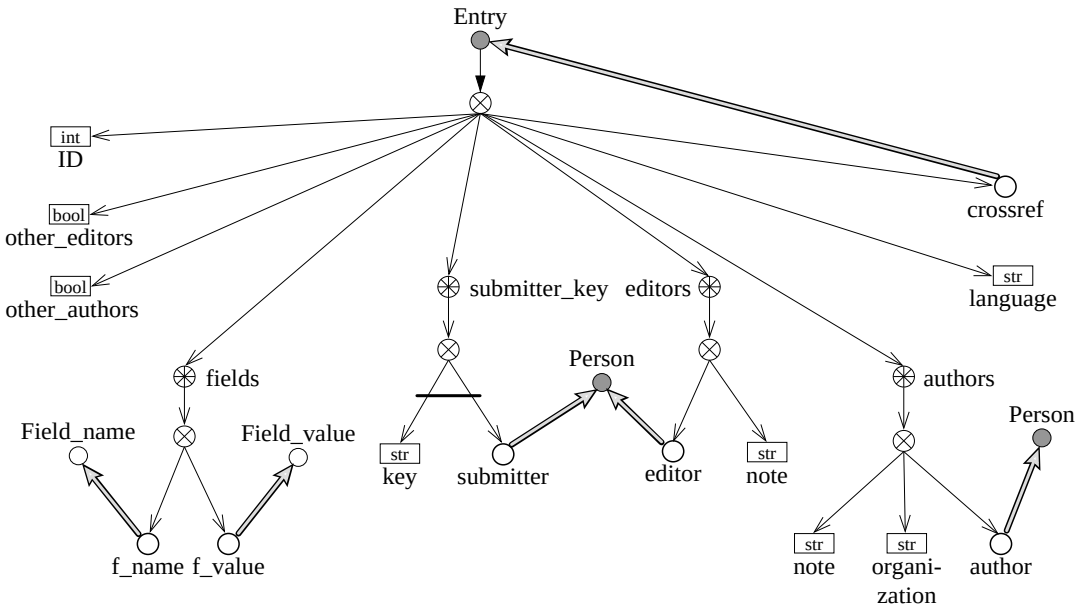


Abbildung 3.6: Klasse Entry

Schlüssels ist in einigen Fällen sehr umständlich:

> Es ist oft notwendig, daß die Einträge, die in einer HTML-Seite enthalten sind, bei weiteren Aktionen identifiziert werden müssen. Der Erfasser wird durch ein Personen-Objekt dargestellt, dessen Objekt-Identität aber nicht sichtbar ist. Sie kann somit nicht für die Identifizierung eines Erfassers nicht verwendet werden. Dies bedeutet, daß pro Eintrag das Schlüsselwort und die vier Bestandteile des Erfassers festgehalten werden müssen.

Deshalb sollte die Klasse *Entry* ein zusätzliches Attribut *ID* besitzen, das für jeden Eintrag eine eindeutige Nummer enthält. *NCSTRL*-Einträge besitzen kein Schlüsselwort, aber die Nummer des technischen Berichts kann als solches verwendet werden. Diese Nummer ist auch als Feld abzuspeichern, da das Schlüsselwort bei Konflikten verändert werden kann.

Modellierung Die resultierende Klasse *Entry* wird in der Abbildung 3.6 dargestellt. Ein einzelnes Feld besteht aus den Komponentenklassen *Name* und *Wert*, d.h. der Feldname ist ein Objekt aus der Klasse *Field_name* (siehe Abschnitt 3.3.4), und dem Wert wird ein Objekt aus der Klasse *Field_value* zugeordnet. Das Schlüsselwort eines Eintrags muß für den Erfasser eindeutig sein. Deshalb bilden Schlüsselwort und Erfasser zusammen den Schlüssel für die Klasse *Entry*.

3.3.4 Darstellung von Feldnamen

Bei der Modellierung von Literatureinträgen wurde beschrieben, weshalb die Felder nicht durch Klassenattribute dargestellt, sondern in einem mengenwertigen Attribut gespeichert werden. Die einzelnen Tupel dieses Attributs bestehen aus dem Feld und dem zugehörigen Wert. Bei der Angabe des Feldes werden neben dem Namen weitere Informationen benötigt. Zu diesen Information zählt z.B. die Erfasserabhängigkeit eines Feldes. Aus diesem Grund werden Felder durch Objekte repräsentiert. Da BibTeX-Literaturdatenbanken neben den Standard-Feldern frei definierbare Felder enthalten können, werden zwei Klassen eingeführt: eine für Standardfelder und eine für frei definierbare Felder. Gleichnamige freie Felder verschiedener Erfasser müssen nicht die gleiche Bedeutung haben. Deshalb wird für ein frei definierbares Feld neben dem Namen auch der Erfasser festgehalten. Beide Angaben bilden zusammen einen Schlüssel, da folgendes angenommen wird: Für einen Erfasser haben gleichnamige freie Felder auch in verschiedenen BibTeX-Dateien dieselbe Bedeutung.

Einige Felder wie *keywords* bestehen aus einer Liste von Angaben, die unabhängig voneinander sind. Bei der Suche in solchen Feldern darf nicht auf Gleichheit getestet werden, da dies zu falschen Ergebnissen führt, sondern es muß immer die Ungewißheitsselektion [HS95] angewendet werden. Als Ungewißheitsselektion wird die Suche nach Teilzeichenketten bezeichnet. Folgendes Beispiel soll das Problem verdeutlichen: Es werden alle Einträge gesucht, bei denen im Feld *keywords* das Wort „Datenbank“ enthalten ist. Das *keywords*-Feld eines Eintrag ist mit „Datenbank, Objektorientierte Datenbank“ angegeben. Wird bei der Suche der Gleichheitsoperator angewendet, dann kommt dieser Eintrag nicht im Anfrageergebnis vor. Sucht man dagegen im Feld *keywords* nach der Teilzeichenkette „Datenbank“, dann ist der Eintrag im Anfrageergebnis enthalten. Aus diesem Grund wird das boolesche Attribut *multiple_value* definiert, das „wahr“ ist, wenn ein Feld aus einer Liste von Angaben besteht.

Modellierung Die Klassen, die für die Repräsentation von Feldnamen benötigt werden, sind in Abbildung 3.7 dargestellt. Den Klassen *Standard_field* und *Free_field* werden abstrakte Domänen zugeordnet, die Domäne von *Field_name* ergibt sich aus der Vereinigung der Extensionen dieser Unterklassen. Die Klasse *Field_name* besitzt ein Attribut *meaning*. Interessant ist dieses Attribut vor allem bei einem freien Feld, da darin die Bedeutung dieses Feldes für den Erfasser festgehalten werden kann. Der Name des Feldes ist der Schlüssel für die Klasse *Field_name*.

In *refdbms* und *NCSTRL* wird ein Feld nicht durch den Namen, sondern durch einen Buchstaben gekennzeichnet. Objekte der Klasse *Standard_field* haben deshalb die beiden Attribute *refdbms_tag* und *ncstrl_tag*, in denen dieser Buchstabe für das entsprechende Format festgehalten wird. Die booleschen Attribute, die

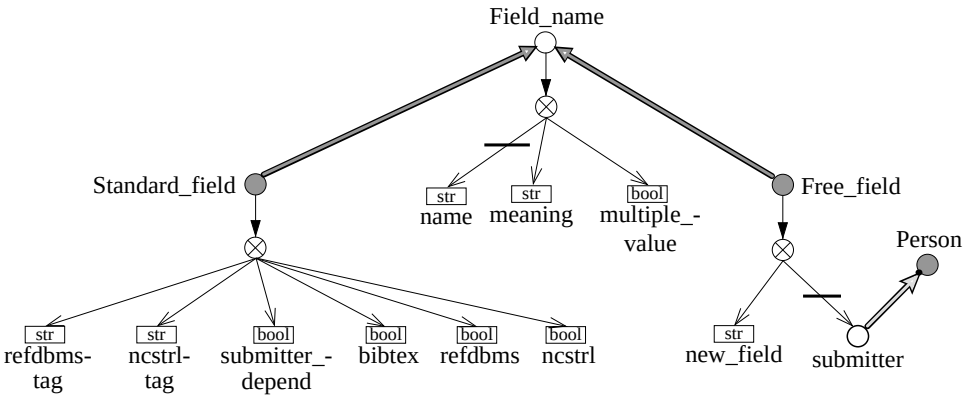


Abbildung 3.7: Klassen für die Darstellung von Feldnamen

für die Klasse *Standard_field* definiert wurden, haben folgende Bedeutung:

- Der Wert des Attributs *submitter_depend* ist „wahr“, wenn die Angaben in einem Feld erfasserabhängig sind.
- Die Attribute *bibtex*, *refdbms* und *ncstrl* sind dann „wahr”, wenn das Feld in der jeweiligen Literaturdatenbank definiert ist.

Ein Objekt der Klasse *Free_field* besitzt ein boolesches Attribut *new_field*, das aus programmiertechnischen Gründen eingeführt wurde. Es ist nach dem Erzeugen eines freien Feldes solange „wahr“, bis die benötigten Angaben (z.B. Bedeutung) für dieses Feld vom Anwender gemacht worden sind. Der Name und der Erfasser bilden zusammen den Schlüssel für ein frei definierbares Feld.

3.4 Ungelöste Probleme

Ein Problem stellt die Präambel in BibTeX-Dateien dar, die in diesem Modell nicht berücksichtigt wurde. Dieses Problem ist aber nicht schwerwiegend, da Präambel-Befehle selten verwendet werden. Aus den folgenden Gründen wird die Präambel nicht berücksichtigt:

- Die in der Präambel definierten Befehle können beim Export in das *refdbms*- bzw. *NCSTRL*-Format nicht verwendet werden, da eigene Befehlsdefinitionen in diesen Formaten nicht möglich sind.
- Die Feldwerte müßten separat für jeden Erfasser mit einem Verweis auf die Präambel abgespeichert werden. Die Probleme, die dabei auftreten, wurden bereits im Abschnitt 3.3.2 diskutiert.

Die sinnvollste Lösung für dieses Problem ist das Überlesen von Präambel-Befehlen beim Import einer BibTeX-Datei.

3.5 Literaturvergleich

Einige Arbeiten, die sich mit dem Thema Datenintegration bzw. Transformation von Attributen in Literaturdatenbanken beschäftigen, sollen im folgenden kurz beschrieben werden.

In [Con97] werden verschiedene Lösungen für die Schemaintegration diskutiert. Man kann die Integration der Literaturdatenbanken mit dem föderativen Datenbankentwurf vergleichen, da unabhängig voneinander entworfene Literaturdatenbanken zusammengebracht werden sollen, deren Datenorganisation selbst nicht geändert werden kann. Das Ausgangsproblem beim föderativen Datenbankentwurf ist folgendes: Zwischen den zu integrierenden Schemata können strukturelle und semantische Inkompatibilitäten auftreten. Diese Inkompatibilitäten müssen bei der Integration erkannt und geeignete Lösungen gefunden werden. Bei der Integration der Literaturdatenbanken treten Konflikte zwischen den Feldern auf, die für einen Eintrag angegeben werden können, sowie zwischen den möglichen Eintragstypen. Im Datenbankschema (siehe Abschnitt 3.3) existieren keine strukturellen Konflikte, da

- die Mengen der Eintragstypen von BibTeX und *refdbms* in einer Teilmengenbeziehung stehen,
- für technische Berichte, die in *NCSTRL* enthalten sind, in BibTeX und *refdbms* der Eintragstyp `TechReport` existiert, und
- die Felder nicht als Attribute in den Klassen für die einzelnen Eintragstypen verwendet wurden.

Die Felder eines Eintrags werden in einem mengenwertigen Attribut festgehalten. Zwischen den Werten dieses Attributs können semantische Konflikte auftreten. Die Lösungsansätze, die in [Con97] für solche Konflikte vorgestellt werden, sind aber zu allgemein und somit nicht für diese Arbeit verwendbar.

Die Artikel [GCGMP96, BCG97, BCGP96] wurden im Rahmen des Stanford Digital Library Projekts veröffentlicht. Das Ziel dieses Projektes ist die Bereitstellung einer Infrastruktur, die eine Interoperabilität zwischen heterogenen, autonomen digitalen Bibliotheksdiensten gewährleistet. Dazu wurde eine Metadaten-Architektur entwickelt, die verschiedene Attributmodell-Transformatoren enthält. Diese Transformatoren bilden Attribute und Attributwerte von einem Attributmodell in ein anderes ab (wenn dies möglich ist). Bei der Entwicklung der Transformatoren wurden folgende Erkenntnisse gewonnen:

- Selbst wenn eine Transformation möglich ist, dann gestaltet sie sich oft sehr schwierig und ist häufig auch mit dem Verlust von Informationen verbunden.
- Die Transformation kann über Regeln erfolgen, oder sie kann heuristisch ausgeführt werden und weitere Attribute, die für das Dokument angegeben wurden, berücksichtigen.
- Eine Transformation beinhaltet nicht nur die Abbildung der Quellattribute auf Zielattribute. Sie muß auch bei jeden Attributwert eine Konvertierung des Datentyps, der für der für das Quellattribut spezifiziert wurde, in den Datentyp für das Zielattribut vornehmen. Dabei kann es auch wieder zu Informationsverlusten kommen.

Es wurde bereits ein Attributmodell-Transformator implementiert, der die Transformation zwischen *refer* und BibTeX gewährleistet. In den Artikeln wird leider nicht beschreiben, nach welchen Regeln diese Transformation erfolgt. Deshalb ist kein Vergleich mit dem Ansatz, der in dieser Arbeit vorgestellt wurde, möglich. Außerdem unterscheidet sich der in den Artikeln vorgestellte Ansatz und von dem dieser Arbeit, denn die Transformatoren arbeiten paarweise, d.h. es wird eine Transformation von einem Modell in das andere und umgekehrt vorgenommen, während in dieser Arbeit die einzelnen Formate in einem zentralen Modell zusammengefaßt werden.

Kapitel 4

Übertragung von Dateien an den WWW-Server

Damit Dateien mit Literaturreferenzen über das WWW in die Literaturdatenbank importiert werden können, müssen diese für den WWW-Server zugänglich sein. Dies ist kein Problem, wenn der WWW-Server auf das Verzeichnis, in dem sich die zu importierende Datei befindet, zugreifen kann. Ist dies aber nicht der Fall, dann muß die Datei an den WWW-Server übertragen werden.

Für die Übertragung von Dateien an den WWW-Server gibt es grundsätzlich folgende Möglichkeiten:

- Die Übertragung kann über das FTP-Protokoll erfolgen. Dazu muß das Programm FTP gestartet und eine Verbindung zum WWW-Server aufgebaut werden. Danach muß man sich auf dem Server einloggen. Dies ist aber nur möglich, wenn man dort als Benutzer registriert ist. Da diese Möglichkeit nicht über das WWW zu realisieren ist, wird sie nicht in Betracht gezogen.
- Die Datei kann durch die Nutzung der PUT-Methode des HTTP-Protokolls übertragen werden.
- Per HTML-Formular kann die Datei an den WWW-Server geschickt werden. Diese Möglichkeit ist aber nicht in allen WWW-Clients implementiert.

Auf die letzten beiden Möglichkeiten wird in den nächsten Abschnitten näher eingegangen.

4.1 Senden einer Datei per PUT-Methode

Das HTTP-Protokoll dient der Verbindung und dem Datenaustausch zwischen einem WWW-Server und WWW-Clients. Die Kommunikation über HTTP erfolgt nach einem einfachen Request/Response-Schema:

1. *Verbindungsaufbau*: Im ersten Schritt baut der Client eine TCP-Verbindung zum Server auf.

2. *Request*: Nach erfolgreichen Verbindungsaufbau sendet der Client dem Server einen Request. Der Request enthält eine Request-Methode, einen Uniform Resource Identifier (URI), weitere Angaben und gegebenfalls ein Datenobjekt, das als Entity bezeichnet wird.

3. *Response*: Der Server antwortet auf den Request mit einer Response. Die Antwort enthält neben weiteren Angaben einen Statuscode und gegebenfalls wieder ein Entity. Der Statuscode enthält Informationen darüber, ob der Request erfolgreich war oder nicht.

4. *Verbindungsabbau*: Nach Request und Response können sowohl der Server als auch der Client die Verbindung abbauen.

URIs werden in Uniform Resource Names (URNs) und Uniform Resource Locators (URLs) unterteilt. Da URNs noch nicht spezifiziert sind, hat man es in der Regel stets mit URLs zu tun.

Mit der Request-Methode PUT kann ein Entity vom Client zum Server übertragen werden. Der Client spezifiziert im Request einen URI, unter dem der Server das Datenobjekt speichern und anschließend verfügbar machen soll. Das Senden einer Datei muß somit über ein Skript erfolgen, das den entsprechenden Request zusammenstellt und als Entity die Datei enthält. Vor der Abspeicherung des Entities wird geprüft, ob der Absender berechtigt ist, an die Stelle unter dem angegebenen URI zu schreiben. Dies bedeutet, daß wie beim FTP-Protokoll der Absender als Benutzer registriert sein muß. Für die Autorisierung muß der Request also den Nutzer und dessen Paßwort enthalten.

Das folgende Beispiel zeigt, wie ein PUT-Request aussehen muß, wenn man eine BibTeX-Datei, die 317 Byte groß ist und einen Book-Eintrag enthält, an den WWW-Server übertragen will:

```
PUT /users/s00/weber/public_html/test.bib HTTP/1.1
Host: wwwdb.informatik.uni-rostock.de
Authorization: Basic cGhldGhtb246c2FtYm8=
Context-Type: text/html
Context-Length: 317

@Book{BTW9710,
  editor       = "Dittrich, K. R. and Geppert, A.",
  title        = "Proc. GI-Fachtagung ``Datenbanksysteme in B{\"u}ro,
                  Technik und Wissenschaft'' (BTW'97)",
  publisher    = "Springer-Verlag",
  address      = "Bonn",
```

```
  year         = 1997,
  series       = "Informatik aktuell"
}
```

Für die Erstellung des `Authorization`-Headers müssen Nutzer-ID und Paßwort, getrennt durch einen Doppelpunkt, miteinander verknüpft werden. Danach wird diese Verknüpfung mit dem Base64-Algorithmus [RFCa] verschlüsselt.

4.2 Schicken einer Datei per Formular

Um Dateien im WWW übertragen zu können, gibt es eine Erweiterung von HTML, die im RFC 1867 standardisiert und ab Netscape 3.x implementiert ist: Der File-Upload in Formularen.

Dazu wird ein weiterer Typ des `<INPUT>`-Tags definiert, dessen Attribut `TYPE` den Wert `FILE` hat. Dieser Typ stellt ein Eingabefeld für einen Dateinamen aus dem Dateisystem des Benutzers bereit. Zusätzlich ist dieses Feld mit einem Button verbunden. Bei Betätigung dieses Buttons wird eine Dialogbox für die Auswahl einer Datei im lokalen Dateisystem geöffnet.

Die angegebene Datei wird vom WWW-Client eingelesen, kodiert und als Eingabe für ein CGI-Skript an den Server übermittelt. Für die spezielle Kodierung muß das `ENCTYPE`-Attribut des `<FORM>`-Tags den MIME[1]-Typ `multipart/form-data` tragen. Dieser Typ bewirkt, daß alle Eingabefelder als mehrteilige MIME-Mitteilung geschickt werden. Zusätzlich muß das Attribut `METHOD` den Wert `POST` haben.

Folgendes Formular enthält ein normales Eingabefeld und ein Feld, in dem der Benutzer den Namen einer zu übertragenden Datei angeben kann:

```
<FORM ENCTYPE="multipart/form-data" ACTION="upload.cgi" METHOD=POST>
<INPUT TYPE="text" NAME="text"><P>
<INPUT TYPE="file" NAME="upload"><P>
<INPUT TYPE="submit" VALUE="Abschicken">
</FORM>
```

Das in `ACTION` bezeichnete Skript erhält die übermittelten Daten über die Standardeingabe. Die Umgebungsvariable `Content-Type` liefert die Information, daß es sich um eine kombinierte MIME-Mitteilung handelt, die Formulareingaben enthält:

```
multipart/form-data;
boundary=---------------------------181946630912934342792110275512
```

[1] Multipurpose Internet Mail Extension

Der boundary-Teil enthält eine Zeichenkette, die die einzelnen Teile der Mitteilung trennt. Der WWW-Client muß sicherstellen, daß sie in den Feldinhalten selber nicht vorkommt.

Die MIME-Mitteilung, die beim Abschicken des oben vorgestellten Formulars vom WWW-Client generiert wird, sieht folgendermaßen aus:

```
-----------------------------181946630912934342792110275512\r
Content-Disposition: form-data; name="text"\r
\r
Upload-Test\r
-----------------------------181946630912934342792110275512\r
Content-Disposition: form-data; name="upload"; filename="dbb2.bib"\r
\r
@Book{BTW9710,
  editor    = "Dittrich, K. R. and Geppert, A.",
  title     = {Proc. GI-Fachtagung ``Datenbanksysteme in B{\"u}ro,
               Technik und Wissenschaft'' (BTW'97)},
  publisher = "Springer-Verlag",
  address   = "Bonn",
  year      = 1997,
  series    = "Informatik aktuell"
}\r
-----------------------------181946630912934342792110275512--\r
```

Das Zeichen \r stellt eine Escape-Sequenz dar und steht für carriage return. Der boundary-Teil wird bei jedem Abschicken neu generiert und sieht somit jedesmal anders aus. Um die Orginaldatei zu erhalten, müssen die überflüssigen Zeichen, die die Mitteilung enthält, überlesen werden.

4.3 Vergleich der Methoden

Das Senden einer Datei über die PUT-Methode hat den Nachteil, daß der Absender einer Datei auch als Benutzer auf dem WWW-Server registriert sein muß. Da für die Autorisierung das Paßwort des Nutzers benötigt wird, müßte dieses im Formular abgefragt werden. Aus Sicherheitsgründen ist dies aber nicht wünschenswert, da es unverschlüsselt vom Client zum Server geschickt wird. Der Nachteil beim File-Upload durch ein Formular liegt darin, daß die Datei von einem Skript aus der empfangenen Nachricht herausgefiltert werden muß. Außerdem unterstützen nicht alle WWW-Clients diese Methode. Sie wird trotzdem für den Import verwendet, da eine Autorisierung entfällt und somit auch Nutzer, die nicht auf dem WWW-Server registriert sind, Literatureinträge importieren können.

Kapitel 5

Das Datenbanksystem

Das entwickelte Datenbankschema soll nun als Grundlage für ein Datenbanksystem dienen, das folgende Funktionen bietet:

- Definition und Änderung von Abkürzungen, die beim Import verwendet werden können,
- Import beliebiger BibTeX-, *refdbms*- und *NCSTRL*-Einträge mit
  - Konfliktauflösung,
  - einer Unterscheidung zwischen deutschen und englischen Einträgen und
  - einer Zuordnung der Einträge zum jeweiligen Erfasser,
- Änderung vorhandener Einträge inklusive Reklassifizierung,
- Export frei wählbarer Referenzen sowie
- mehrere Sichten auf die Datenbank, die die Suche nach
  - Themen,
  - Autoren,
  - den Einträgen eines Erfassers und
  - einzelnen Attributen

  ermöglichen.

In den folgenden Abschnitten wird dargestellt, wie diese Funktionen im Datenbanksystem umgesetzt werden können.

5.1 Benötigte Standardfelder

Beim Import werden einige Felder speziellen Attributen eines *entry*-Objektes zugewiesen und im Zuge der Konfliktauflösung werden einige Felder anderen zugeordnet. In der Tabelle 5.1 sind alle Felder aufgelistet, für die Objekte der Klasse *Standard_field* erzeugt werden müssen. Die Objekte besitzen die Attribute *bibtex* (**b**), *refdbms* (**r**), *ncstrl* (**n**), *refdbms_tag* (**r_t**), *ncstrl_tag* (**n_t**), *submitter_depend* (**s_d**) und *multiple_value* (**m_v**). Die Werte, die diesen Attributen in den jeweiligen Feld-Objekten zugewiesen werden müssen, sind ebenfalls in der Tabelle angegeben.

Feld	b	r	n	r_t	n_t	s_d	m_v
address	√						
abstract	√	√[1]	√	x	X	√	
annote	√	√[1]		o		√	
booktitle	√	√		B			
bibtex key		√		b		√	
chapter	√						
conference name		√		C			
conference location		√		c			
computing reviews categories			√		Y	√	√
day[2]							
edition	√						
ftp	√					√	
howpublished	√						
institution	√						
ISBN/ISSN		√		I			
journal	√	√		J			
key	√					√	
keywords	√	√	√	k	K	√	√
location		√		L			
month	√						
modification date			√		Z		
note	√	√[1]		O		√	
number	√	√		N			
organization	√						
pages	√	√		P			
publisher	√	√		p			
report number		√	√	R	R		

Fortsetzung auf nächster Seite

Fortsetzung von vorheriger Seite							
Feld	**b**	**r**	**n**	**r_t**	**n_t**	**sd**	**mv**
school	√						
series	√	√		S			
submitter		√		s		√	
title	√	√	√	T	T		
type	√						
URL	√		√		U	√	
volume	√	√		V			
year	√						

Tabelle 5.1: Felder der Datenbank

5.2 Abkürzungen

Refdbms und BibTeX bieten die Möglichkeit, Abkürzungen in Feldtexten zu verwenden. BibTeX stellt einige Standardabkürzungen in den BibTeX-Stildateien bereit. In *refdbms* sind die Abkürzungen ebenfalls in Dateien enthalten, die entsprechend dem Stil des Literaturverzeichnisses verwendet werden. Dies bedeutet, daß die ausgeschriebene Form einer Abkürzung in Abhängigkeit vom gewählten Stil variieren kann. Wenn man die definierten Abkürzungen in den BibTeX-Stildateien bzw. *refdbms*-Abkürzungsdateien nutzen will, muß man diese Dateien beim Import ebenfalls an den WWW-Server übertragen und analysieren. Es kann dabei zu Konflikten kommen, wenn eine eingelesene Abkürzung bereits in der Datenbank existiert und sich die Werte unterscheiden. Deshalb wurde die Entscheidung getroffen, beim Import keine BibTeX-Stildateien bzw. *refdbms*-Abkürzungsdateien zu berücksichtigen. Das Datenbanksystem muß aus diesem Grund Möglichkeiten für die Definition und die Änderung von Abkürzungen bereitstellen.

5.2.1 Definition

Im Datenbankschema können einer Abkürzung Werte in verschiedenen Sprachen zugeordnet werden. Für den Wert wird eine Kurz- und eine Langform festgehalten. Folgende Angaben müssen bei der Definition einer Abkürzung eingegeben werden:

[1] Zuordnung nach Konfliktauflösung
[2] zusätzliches Feld für Konfliktauflösung

- Name der Abkürzung,
- Sprache, in der die Werte angegeben werden,
- Kurz- und Langform,
- Wert des *bibtex*-, *refdbms* und *ncstrl*-Flags („wahr“ oder „unwahr“),
- Felder, in denen die Abkürzung beim Import durch die Kurz- oder Langform bzw. durch eine Referenz auf das Abkürzungsobjekt ersetzt wird, wenn das entsprechende Flag für das Eintragsformat auf „wahr“ gesetzt ist (ansonsten erfolgt die Ersetzung in allen Feldern).

Außerdem sollte bei der Definition die Möglichkeit bestehen, Werte in weiteren Sprachen anzugeben.

Vor dem Einfügen in die Datenbank muß geprüft werden, ob eine Abkürzung mit dem gleichen Namen bereits existiert. In diesem Fall darf die Abkürzung nicht eingefügt werden, da der Name der Schlüssel für die Klasse ist. Ansonsten wird ein Objekt für die Abkürzung erzeugt. Die eingegebenen Werte müssen dann den entsprechenden Attributen des Objektes zugeordnet werden.

5.2.2 Änderung

Alle Angaben, die einer Abkürzung zugeordnet sind, können geändert werden. Dazu zählt auch der Name der Abkürzung. Weiterhin muß der Benutzer in der Lage sein, Werte in weiteren Sprachen anzugeben. Dabei muß darauf geachtet werden, daß eine Sprache nur einmal in der Wertemenge enthalten ist.

Bei der Änderung des Namens muß sichergestellt werden, daß eine Abkürzung mit dem neuen Namen noch nicht existiert. Ist bereits eine Abkürzung mit dem neuen Namen vorhanden, wird der Zustand des Abkürzungsobjektes nicht verändert. Ansonsten werden die Attribute des Abkürzungsobjektes entsprechend den eingebenen Werten geändert.

5.3 Import-Schnittstelle

Um Dateien in den jeweiligen Formaten importieren zu können, wird ein HTML-Formular benötigt, das folgende Eingaben gestattet:

- Name des Erfassers,
- die Datei, die importiert werden soll (Eingabefeld vom Typ `file`),

- das Format der Datei (BibTeX, *refdbms* oder *NCSTRL*),
- die Sprache, die den Einträgen der Datei zugeordnet wird.

Das `ENCTYPE`-Attribut des Formulars muß die Angabe `multipart/form-data` enthalten (siehe Abschnitt 4.2). Der Erfasser wird beim Import der Einträge durch ein Personen-Objekt dargestellt. Dieses Personen-Objekt hat die Attribute *Vorname*, *von-Teil*, *Nachname* und *Jr-Teil*. Aus diesem Grund müssen für den Erfassernamen diese vier Bestandteile angegeben werden.

Durch das Abschicken des Formulars wird ein Skript angestoßen. Dieses Skript erhält die übermittelten Daten über die Standardeingabe. Aus diesen Daten müssen die vier Bestandteile des Erfassernamens, die Sprache sowie die Orginaldatei herausgefiltert werden. Die Orginaldatei muß dann auf dem WWW-Server temporär gespeichert werden, wobei der aktuelle Zeitticker als Dateiname genutzt wird. Existiert bereits eine Datei, deren Name dem aktuellen Zeitticker entspricht, dann muß eine Zahl, die fortlaufend erhöht wird, angehängt werden. Der Parser für das entsprechende Format kann somit die Datei auswerten. Sie wird nach nach dem Einlesen aller Einträge wieder gelöscht.

Der Import von Einträgen kann in folgende Schritte gegliedert werden:

1. Es wird geprüft, ob in der Klasse *Person* bereits ein Objekt existiert, dessen Attribute mit den Bestandteilen des Erfassernamens übereinstimmen. Ist noch kein entsprechendes Objekt in dieser Klasse vorhanden, dann muß es erzeugt werden.
2. Die Einträge werden von einem Parser eingelesen. Für jedes Format wird ein eigener Parser benötigt.
3. Die eingelesenen Einträge werden in die Literaturdatenbank eingefügt. Dabei können Konflikte mit bereits vorhandenen Einträgen auftreten, die aufgelöst werden müssen.

Das Einfügen von Einträgen läuft in den verschiedenen Formaten ähnlich ab. Aus diesem Grund wird für den dritten Schritt eine Schnittstelle vorgestellt, die für alle Formate verwendet werden kann. Die verschiedenen Parser legen die Informationen, die für einen Eintrag bestimmt wurden, in folgenden Variablen ab:

- `entry_type`: Der Typ des Eintrags wird in dieser Variable festgehalten.
- `key`: Das Schlüsselwort, das einem Eintrag zugeordnet wurde, ist in dieser Variable enthalten.

- `crossref`: Diese Variable enthält das Schlüsselwort des Eintrags, auf den verwiesen wird.

- `author_list`: In dieser Liste werden alle Autoren eines Eintrags gesammelt. Ein Tupel dieser Liste besteht aus den Komponenten `Vorname`, `von-Teil`, `Nachname`, `Jr-Teil`, `Anmerkung` und `Organization`.

- `other_authors`: Diese Variable enthält einen Wahrheitswert. Er ist „wahr“, wenn die Autorenangabe im BibTeX-Format mit `and others` endet.

- `editor_list`: Diese Liste enthält alle Herausgeber eines Eintrags. Die Tupel dieser Liste bestehen aus den Komponenten `Vorname`, `von-Teil`, `Nachname`, `Jr-Teil` und `Anmerkung`.

- `other_editors`: Diese Variable ist „wahr“, wenn die Herausgeberangabe im BibTeX-Format mit `and others` beendet wurde.

- `field_list`: In dieser Liste werden die Felder eines Eintrags gesammelt, denen ein Objekt der Klasse *Field_name* zugeordnet ist. Die einzelnen Tupel besitzen die Komponenten `Feldname` und `Feldwert`. Die Komponente `Feldname` enthält das *Field_name*-Objekt.

- Das Erfasser-Objekt, das im ersten Schritt ermittelt wurde, wird in der Variable `submitter` abgelegt.

In den folgenden Abschnitten wird für die einzelnen Formate beschrieben, wie die Informationen, die für einen Eintrag angegeben wurden, in den Variablen festgehalten werden. Danach erfolgt dann die Vorstellung der Schnittstelle, die zum Einfügen eines Eintrags in die Datenbank genutzt wird.

5.3.1 Einlesen eines BibTeX-Eintrags

Die folgenden Felder müssen beim Einlesen eines BibTeX-Eintrags gesondert behandelt werden müssen:

- *author*: Dieses Feld kann mehrere Autoren beinhalten. Der Parser muß die einzelnen Autoren erkennen und die Namen der Autoren in die Bestandteile *Vorname*, *von-Teil*, *Nachname*, *Jr-Teil* zerlegen. Diese Bestandteile und die Komponenten `Anmerkung` und `Organization`, die jeweils eine leere Zeichenkette enthalten, bilden zusammen ein Tupel, das in die Liste `author_list` eingefügt wird. Endet das Feld mit `and others`, dann wird die Variable `other_authors` auf „wahr“ gesetzt.

- *editor*: In diesem Feld können mehrere Herausgeber angegeben sein. Die Namen der einzelnen Herausgeber werden in die Bestandteile *Vorname*, *von-Teil*, *Nachname*, *Jr-Teil* zerlegt. Diese Bestandteile und die Komponente `Anmerkung`, der eine leere Zeichenkette zugeordnet ist, bilden zusammen ein Tupel, das in die Liste `editor_list` eingefügt wird. Wenn das Feld mit `and others` endet, dann wird `other_editors` auf „wahr“ gesetzt.
- *crossref*: Der Wert des Feldes wird der Variable `crossref` zugeordnet.
- *title* und *booktitle*: Bei diesen Feldern müssen die geschweiften Klammern, die zur Unterdrückung der Kleinschreibung angegeben sind, herausgefiltert werden.
- *number*: Ist der Typ des Eintrags *Techreport*, dann wird der Wert dieses Feldes dem Feld *report number* zugeordnet.

Bei einem Standardfeld wird in der Klasse *Standard_field* nach einem Objekt gesucht, dessen Attribut *name* mit dem Feldnamen übereinstimmt. Bei einem frei definierbaren Feld muß geprüft werden, ob bereits ein Objekt mit dem Feldnamen und dem Erfasser in der Klasse *Free_field* existiert. Ist dies nicht der Fall, dann wird ein solches Objekt erzeugt. Für alle vorhandenen Felder außer *author*, *editor* und *crossref* muß ein Tupel erzeugt werden, das aus dem Objekt, das für den Feldnamen bestimmt wurde, und dem Feldwert gebildet wird. Die einzelnen Tupel werden dann in die Liste `field_list` eingefügt. Der Eintragstyp wird der Variable `entry_type` zugeordnet (der Eintragstyp *Misc* wird zu *Miscellaneous* erweitert) und das Schlüsselwort des Eintrags wird `key` zugewiesen.

5.3.2 Einlesen eines *refdbms*-Eintrags

Zwischen den Feldern der einzelnen Literaturdatenbankformate treten Konflikte auf. Um diese Konflikte auflösen zu können, müssen einige *refdbms*-Felder anderen Felder der Datenbank zugeordnet werden. Andere *refdbms*-Felder enthalten Informationen, die für die Erzeugung eines Eintragsobjektes benötigt werden. Aus diesen Gründen sind die folgenden Felder beim Einlesen eines *refdbms*-Eintrags gesondert zu behandeln:

%z Der Wert dieses Feldes wird der Variable `entry_type` zugewiesen.

%K Der Variable `key` wird der Wert dieses Feldes zugewiesen.

%A Der Name des Autors muß in die Bestandteile *Vorname*, *von-Teil*, *Nachname*, *Jr-Teil* getrennt werden. Diese Bestandteile und die Komponenten `Anmerkung` und `Organization`, die jeweils eine leere Zeichenkette enthalten, bilden zusammen ein Tupel, das in die Liste `author_list` eingefügt wird.

%a Dieses Feld enthält Anmerkungen zu den davor angegebenen Autoren. Somit müssen alle Tupel der Liste `author_list` durchlaufen werden: Enthält die Komponente `Anmerkung` eines Tupels eine leere Zeichenkette, dann wird ihr der Wert dieses Feldes zugewiesen.

%D Dieses Feld hat das Format *Tag Monat Jahr*. Der Parser muß die einzelnen Angaben erkennen. Dem Feld *month* wird der *Monat*, *day* der *Tag* und *year* das *Jahr* zugeordnet.

%E Der Herausgebername muß in die Bestandteile *Vorname*, *von-Teil*, *Nachname*, *Jr-Teil* getrennt werden. Diese Bestandteile und die Komponente `Anmerkung`, die eine leere Zeichenkette enthält, bilden zusammen ein Tupel, das in die Liste `editor_list` eingefügt wird.

%e Dieses Feld enthält Anmerkungen zu den davor angegebenen Herausgebern. Bei allen Tupeln der Liste `editor_list` wird geprüft, ob die Komponente `Anmerkung` eine leere Zeichenkette enthält. Ist dies der Fall, dann wird `Anmerkung` der Wert dieses Feldes zugewiesen.

%L Enthält das Feld Angaben, die mit `ftp` beginnen, dann werden diese aus dem Feld entfernt und dem Feld *ftp* zugeordnet. Angaben, die mit `http` beginnen, werden ebenfalls aus dem Feld herausgefiltert und dem Feld *URL* zugewiesen.

%N Enthält das Feld eine Angabe in der Form *xxx* `edition`, dann wird diese aus dem Feld herausgefiltert und *xxx* wird dem Feld *edition* zugeordnet. Kommt bei einem *InBook*-Eintrag in diesem Feld eine Angabe vor, die nur aus Ziffern besteht, dann wird diese dem Feld *chapter* zugeordnet.

%p Der Verlag ist in der Form *Verlagsname, Adresse* angegeben. Die *Adresse* wird dem Feld *address* zugewiesen. Welchem Feld der *Verlagsname* zugeordnet wird, hängt vom Eintragstyp ab:

Miscellaneous:	*howpublished*
TechReport:	*institution*
Manual:	*organization*
PhDthesis:	*school*
sonst:	*publisher*

%y Dieses Feld enthält Organisationen, denen die zuvor angegebenen Autoren angehören. Bei allen Tupeln der Liste `author_list` wird geprüft, ob die `Organization`-Komponente eine leere Zeichenkette enthält. Ist dies der Fall, dann wird `Organization` der Wert dieses Feldes zugewiesen.

Enthält ein *%L*- bzw. *%N*-Feld nach dem Herausfiltern von Informationen noch weitere Angaben, dann wird es in Liste `field_list` aufgenommen. Die restlichen *refdbms*-Felder des Eintrags und die Felder, denen Werte zugeordnet wurden, werden ebenfalls in `field_list` eingefügt. Ein Tupel dieser Liste besitzt die Komponente `Feldname`, der ein Objekt aus der Klasse *Field_name* zugewiesen werden muß. Dieses Objekt wird folgendermaßen bestimmt:

- Bei den Feldern, denen Werte zugeordnet wurden, wird in der Klasse *Standard_field* nach einem Objekt gesucht, dessen Attribut *name* mit dem Feldnamen übereinstimmt.
- Bei den übrigen Feldern wird in der Klasse *Standard_field* nach einem Objekt gesucht wird, bei dem das *refdbms_tag*-Attribut mit dem Buchstaben, der das Feld kennzeichnet, übereinstimmt.

5.3.3 Einlesen eines *NCSTRL*-Eintrags

Die folgenden Felder eines *NCSTRL*-Eintrags müssen vom Parser gesondert ausgewertet werden:

%A Ein Eintrag kann mehrere *%A*-Felder besitzen, die jeweils einen Autorennamen enthalten. Dieser Name muß in die Bestandteile *Vorname*, *von-Teil*, *Nachname*, *Jr-Teil* getrennt werden. Diese Bestandteile und die Komponenten `Anmerkung` und `Organization`, die jeweils eine leere Zeichenkette enthalten, bilden zusammen ein Tupel, das in die Liste `author_list` eingefügt wird.

%D Dieses Feld hat das Format *Monat Tag, Jahr*. Der Parser muß die einzelnen Angaben erkennen. Dem Feld *month* wird der *Monat*, *day* der *Tag* und *year* das *Jahr* zugeordnet.

%I Das Feld enthält den Herausgeber des technischen Berichts in der Form *Institution, Adresse*. Die *Institution* wird dem Feld *institution* und die *Adresse* *address* zugewiesen.

%U Wenn der Feldwert mit `ftp` beginnt, dann wird er dem Feld *ftp* zugeordnet.

Die Felder, denen Werte zugeordnet wurden, und die restlichen *NCSTRL*-Felder werden in die Liste `field_list` eingefügt. Wenn das *%U*-Feld keine `ftp`-Angabe enthält, dann wird es ebenfalls in diese Liste aufgenommen. Für die Komponente `Feldname` eines `field_list`-Tupels wird ein *Field_name*-Objekt folgendermaßen ermittelt:

- Bei den Feldern, denen Werte zugeordnet wurden, wird in der Klasse *Standard_field* nach einem Objekt gesucht, dessen Attribut *name* mit dem Feldnamen übereinstimmt.
- Bei den übrigen Feldern wird in der Klasse *Standard_field* nach einem Objekt gesucht wird, bei dem das *ncstrl_tag*-Attribut mit dem Buchstaben, der das Feld kennzeichnet, übereinstimmt.

Eine *NCSTRL*-Datei besteht technischen Berichten. Der Variable `entry_type` wird somit die Angabe *Techreport* zugewiesen. Ein *NCSTRL*-Eintrag besitzt kein Schlüsselwort. Aus diesem Grund wird die technische Berichtsnummer, die im *%R*-Feld angegeben ist, der Variable `key` zugewiesen. Da das Schlüsselwort bei Konflikten verändert werden kann, ist die technische Berichtsnummer auch in der Liste `field_list` festzuhalten.

5.3.4 Anmerkungen zu den Parsern

Die Parser für das BibTeX- und das *refdbms*-Format müssen prüfen, ob die Schlüsselwörter aller Einträge in der Datei eindeutig sind. Besitzt ein Eintrag ein Schlüsselwort, das bereits einem anderen Eintrag zugeordnet wurde, dann wird er ignoriert und der Nutzer erhält eine entsprechende Mitteilung.

Kommt in einem Feldwert eines BibTeX- bzw. *refdbms*-Eintrags eine Abkürzung vor, müssen die folgenden Aktionen durchgeführt werden:

- Es muß ermittelt werden, ob in der Klasse *Abbrevation* ein Objekt existiert, dessen *name*-Attribut mit der Abkürzung übereinstimmt.
- Konnte ein Objekt bestimmt werden, dann ist zu prüfen, ob es auch für Abkürzungen in dem Feld genutzt werden kann. Dazu wird je nach Format das *bibtex*- bzw. *refdbms*-Attribut des *Abbrevation*-Objektes ausgewertet. Ist es „wahr“, dann muß getestet werden, ob das Feld im mengenwertigen Attribut *fields* des Objektes vorkommt. Wenn dies der Fall ist oder wenn das *bibtex*- bzw. *refdbms*-Attribut „unwahr“ ist, dann wird das Objekt für die Abkürzung verwendet.
- Enthält das Feld neben der Abkürzung weitere Angaben, dann muß die Abkürzung aufgelöst werden. Dazu wird die Langform in der Sprache, die dem Eintrag zugeordnet wurde, gewählt. Der Feldwert ergibt sich aus der Verkettung der ausgeschriebenen Form der Abkürzung und den anderen Angaben des Feldes.

Die Parser für die verschiedenen Formate sollten auch überprüfen, ob die zwingenden Felder für einen Eintrag angegeben wurden und den Nutzer gegebenfalls auf fehlende Felder hinweisen.

Der BibTeX-Parser hat noch eine weitere Aufgabe: Enthält der Eintrag x einen Querverweis auf den Eintrag y, dann muß geprüft werden, ob y in der Datei vorhanden ist. Der Querverweis wird ignoriert, wenn der Eintrag y in der BibTeX-Datei vor x steht. Besitzt der Eintrag y wiederum einen Querverweis, dann wird dieser nicht berücksichtigt, da geschachtelte Querverweise nicht erlaubt sind.

5.3.5 Einfügen eines Eintrags in die Datenbank

Bevor der Eintrag in die Datenbank eingefügt wird, muß geprüft werden, ob der Eintrag bereits vorhanden ist. Einträge sind potentiell gleich, wenn sie in den folgenden Angaben übereinstimmen:

- Autoren,
- Herausgeber sowie die Felder
- *title*, *publisher*, *howpublished*, *institution* und *school*.

Bei mehreren Autoren und Herausgeber spielt die Reihenfolge, in der sie angegeben sind, keine Rolle. Für die Felder wird folgende Festlegung getroffen: Der Wert eines nichtdefinierten Feldes ist eine leere Zeichenkette.

Wurden potentiell gleiche Einträge erkannt, dann muß der Nutzer entscheiden, ob einer der Einträge wirklich gleich ist. Wie die Angaben des eingelesenen Eintrags verarbeitet werden, wenn ein bereits vorhandener Eintrag vom Nutzer als gleich eingestuft wurde, wird im Abschnitt 5.3.6 dargestellt.

Existiert kein gleicher Eintrag, dann wird ein Objekt in der Klasse, die `entry_type` entspricht, erzeugt. Konnte `entry_type` keine Klasse zugeordnet werden, dann muß ein Objekt in der Klasse *Other_type* erzeugt und dem Attribut *name* dieses Objektes die Variable `entry_type` zugewiesen werden. Das erzeugte Objekt wird im folgenden mit `entry` bezeichnet. Dem Attribut *language* wird die Sprache zugeordnet, die zu Beginn des Imports spezifiziert wurde. Für das Attribut *ID* muß eine Nummer, die innerhalb der Datenbank eindeutig ist, ermittelt werden. Den anderen Attributen des `entry`-Objektes werden die Angaben, die nach dem Einlesen des Eintrags in den verschiedenen Variablen enthalten sind, folgendermaßen zugewiesen:

- `author_list`: Für jedes Tupel dieser Liste wird ein Tupel mit den Komponenten `author`, `note` und `organization` zusammengestellt, das mit `author` bezeichnet wird. Die Komponenten des `author`-Tupels werden wie folgt bestimmt:

  - Es wird geprüft, ob ein Objekt in der Klasse *Person* vorhanden ist, dessen Attribute *first_part*, *von_part*, *last_part* und *jr_part* mit den Komponenten `Vorname`, `von-Teil`, `Nachname` und `Jr-Teil` des `author_list`-Tupels übereinstimmen.
  - Existiert noch kein entsprechendes Objekt, dann muß es erzeugt werden.
  - Das ermittelte bzw. erzeugte Objekt wird der Komponente `author` zugewiesen.
  - Die Komponenten `note` und `organization` ergeben sich aus den Komponenten `Anmerkung` und `Organization` des `author_list`-Tupels.

  Die einzelnen `author`-Tupel werden dann in das mengenwertige Attribut *authors* eingefügt.

- `other_authors`: Diese Variable wird dem gleichnamigen Attribut zugewiesen.

- `editor_list`: Für jedes Tupel dieser Liste wird ein Tupel mit den Komponenten `editor` und `note` zusammengestellt, das mit `editor` bezeichnet wird. Die Komponenten des `editor`-Tupels werden wie folgt bestimmt:

  - Es wird geprüft, ob ein Objekt in der Klasse *Person* vorhanden ist, dessen Attribute *first_part*, *von_part*, *last_part* und *jr_part* mit den Komponenten `Vorname`, `von-Teil`, `Nachname` und `Jr-Teil` des `editor_list`-Tupels übereinstimmen.
  - Existiert noch kein entsprechendes Objekt, dann wird es erzeugt.
  - Das ermittelte bzw. erzeugte Objekt wird der Komponente `editor` zugewiesen.
  - Die Komponente `note` ergibt sich aus der Komponente `Anmerkung` des `editor_list`-Tupels.

  Die einzelnen `editor`-Tupel werden dann in das mengenwertige Attribut *editors* eingefügt.

- `other_editors`: Diese Variable wird dem gleichnamigen Attribut zugewiesen.

- `key`: Es muß geprüft werden, ob das Schlüsselwort, das in dieser Variablen enthalten ist, vom Erfasser bereits für ein anderen Eintrag angegeben wurde. Wenn dies der Fall ist, dann muß der Nutzer ein anderes Schlüsselwort für den Eintrag definieren. Für das neue Schlüsselwort wird dann auch wieder eine Überprüfung vorgenommen.

  Das Schlüsselwort muß zusammen mit dem `submitter`-Objekt in das Attribut *submitter_key* eingefügt werden.

- `crossref`: Der Eintrag, auf den `crossref` verweist, muß vor diesem Eintrag in die Datenbank eingefügt werden. Das entsprechende Objekt wird dann dem Attribut *crossref* des `entry`-Objektes zugeordnet.

- `field_list`: Für die einzelnen Tupel dieser Liste wird ein Tupel mit den Komponenten `f_name` und `f_value` zusammengestellt, das mit `field` bezeichnet wird. Die Komponente `f_name` ergibt sich aus der Komponente `Feldname` des `field_list`-Tupels. Der Komponente `f_value` wird ein Objekt zugewiesen, das aus einer Unterklasse von *Field_value* stammt. Dieses Objekt ergibt aus der Komponente `Feldwert` des `field_list`-Tupels. Die Aktionen, die für die Ermittlung bzw. Erzeugung eines `f_value`-Objektes nötig sind, werden im folgenden für die einzelnen Felder beschrieben:

  - *publisher*: Diesem Feld wird ein Verlagsobjekt zugeordnet, das aus Verlagsname und -adresse besteht. Wenn eine Verlagsadresse angegeben wurde, dann steht sie im Feld *address*. Existiert bereits ein Verlagsobjekt mit diesem Namen und dieser Adresse, dann wird es `f_value` zugewiesen.

    Im anderen Fall wird ein Verlagsobjekt erzeugt. Wenn der Eintrag das Feld *address* enthält, dann wird der Wert dieses Feldes dem Attribut *address* des Objektes zugewiesen. Stellt der Verlagsname eine Abkürzung dar bzw. stimmt er mit der Kurz- oder Langform eines Abkürzungsobjektes überein, dann wird das entsprechende Objekt dem Attribut *name* des Verlagsobjektes zugeordnet. Ansonsten muß für das Attribut *name* ein *Single_value*-Objekt erzeugt und der Verlagsname dem Attribut *value* dieses Objektes zugewiesen werden.

  - *journal*: Bei diesem Feld wird `f_value` durch ein Objekt der Klasse *Journal* dargestellt. Existiert noch kein Journalobjekt, dessen Name mit dem `Feldwert` übereinstimmt, dann muß es erzeugt werden. Stellt der Journalname eine Abkürzung dar bzw. stimmt der Name mit der Kurz- oder Langform eines Abkürzungobjektes überein, dann wird das entsprechende Objekt dem Attribut *name* des Journalobjektes zugeordnet. Ansonsten muß für das Attribut *name* ein *Single_value*-Objekt erzeugt und der Journalname dem Attribut *value* des Objektes zugewiesen werden.

  - *month, institution, organization, school, series*: Bei diesen Feldern wird versucht, `f_value` durch ein Abkürzungsobjekt darzustellen, d.h. stellt der `Feldwert` eine Abkürzung dar bzw. stimmt er mit der Kurz- oder Langform eines Abkürzungsobjektes überein, dann wird das entsprechende Objekt `f_value` zugeordnet.

    Ansonsten muß für `f_value` ein *Single_value*-Objekt erzeugt und der `Feldwert` dem Attribut *value* dieses Objektes zugewiesen werden.

- frei definierbares Feld: Für `f_value` wird ein *Single_value*-Objekt erzeugt. Anschließend muß der `Feldwert` dem Attribut *value* dieses Objektes zugewiesen werden.
- erfasserabhängige Felder: Der Komponente `f_value` wird ein *Submitter_depend_value*-Objekt zugeordnet. Der Typ dieses Objektes besteht aus einer Menge von Tupeln. Das Tupel, das in diese Menge eingefügt wird, ergibt sich aus dem `submitter`-Objekt und dem `Feldwert`. Stellt der `Feldwert` eine Abkürzung dar, dann wird diese durch die Langform des zugehörigen *Abbrevation*-Objektes in der Sprache des Eintrags ersetzt.
- sonstige Felder: Für `f_value` muß ein *Single_value*-Objekt erzeugt und der `Feldwert` dem Attribut *value* dieses Objektes zugewiesen werden. Stellt der `Feldwert` eine Abkürzung dar, dann wird diese durch die Langform des zugehörigen *Abbrevation*-Objektes in der Sprache des Eintrags ersetzt.

  Das Feld *address* wird nur eingefügt, wenn der Eintrag kein *publisher*-Feld enthält.

Die einzelnen `field`-Tupel werden in das mengenwertige Attribut *fields* des `entry`-Objektes eingefügt.

Bei einem frei definierbaren Feld müssen vom Nutzer die folgenden Informationen erfragt werden:

- die Bedeutung des Feldes und
- die Angabe, ob es sich um ein *multiple_value*-Feld handelt oder nicht (siehe Abschnitt 3.3.4).

Bei der Ersetzung einer Abkürzung durch ein *Abbrevation*-Objekt muß folgendes beachtet werden: Ein *Abbrevation*-Objekt enthält die booleschen Attribute *bibtex*, *refdbms* und *ncstrl*. Entsprechend dem Format des Eintrags wird eines dieser Attribute ausgewertet. Ist es „wahr“, dann muß getestet werden, ob das Feld im mengenwertigen Attribut *fields* des Objektes vorkommt. Wenn dies der Fall ist oder wenn das entsprechende boolesche Attribut „unwahr“ ist, dann kann das Objekt für die Abkürzung verwendet werden.

5.3.6 Gleiche Einträge

Wenn zwei Einträge als gleich eingestuft wurden, dann müssen die einzelnen Angaben der beiden Einträge miteinander verglichen werden. Der Eintrag, der eingefügt wird, soll im folgenden mit `new_entry` bezeichnet werden. Der in der Datenbank vorhandene Eintrag erhält die Bezeichnung `old_entry`.

Für den Vergleich von Einträgen werden zwei Mengen eingeführt:

- `diff_fields`: In dieser Menge sind die Eigenschaften enthalten, in denen sich die Einträge unterscheiden. Für die Eigenschaft wird der Wert von `new_entry` und der Wert von `old_entry` festgehalten. Die Komponenten des Tupels werden mit `feature`, `new_value` und `old_value` bezeichnet.

- `change_fields`: In dieser Menge werden die Eigenschaften, die geändert werden sollen, mit ihren neuen Werten festgehalten. Ein Tupel der Menge hat die Komponenten `feature` und `value`.

Im folgenden werden die Eigenschaften der Einträge miteinander verglichen. Treten Unterscheide auf, dann wird ein Tupel in der Form (Eigenschaft, `new_entry`-Wert, `old_entry`-Wert) in die Menge `diff_fields` eingefügt:

- Eintragstyp: Die Variable `entry_type` wird mit dem Namen der Klasse, der das `old_entry`-Objekt angehört, verglichen. Wenn das `old_entry`-Objekt aus der Klasse *Other_type* stammt, dann wird `entry_type` mit dem Attribut *name* des Objektes verglichen.

- Schlüsselwort: Ist der Erfasser im mengenwertigen Attribut *submitter_key* des `old_entry`-Objektes vorhanden, dann wird das zugehörige Schlüsselwort mit der Variable `key` verglichen.

  Ansonsten muß geprüft werden, ob `key` vom Erfasser bereits für einen anderen Eintrag angegeben wurde. Ist dies der Fall, dann muß der Nutzer ein anderes Schlüsselwort definieren. Für das neue Schlüsselwort ist dann auch wieder eine Überprüfung vorzunehmen. Anschließend wird das Schlüsselwort zusammen mit dem `submitter`-Objekt in das Attribut *submitter_key* des `old_entry`-Objektes eingefügt.

- *language*: Die Sprache, die `new_entry` zu Beginn des Imports zuordnet wurde, wird mit dem Attribut *language* des `old_entry`-Objektes verglichen.

- *other_authors*: Es muß geprüft werden, ob die Variable `other_authors` muß mit dem Attribut *other_authors* des `old_entry`-Objektes übereinstimmt.

- *other_editors*: Die Variable `other_editors` wird mit dem Attribut *other_editors* des `old_entry`-Objektes verglichen.

- *crossref*: Der Eintrag, auf den die Variable `crossref` verweist, muß in die Datenbank eingefügt werden. Anschließend kann man dann prüfen, ob das für den Eintrag erzeugte bzw. ermittelte Objekt mit dem Attribut *crossref* des `old_entry`-Objektes übereinstimmt.

- Felder: Für jedes Feld, das in der Variable `field_list` enthalten ist, wird geprüft, ob es im mengenwertigen Attribut *fields* des `old_entry`-Objektes vorkommt. Ist es vorhanden, dann werden die Werte der Felder verglichen. Bei erfasserabhängigen Feldern wird für den Vergleich der Wert gewählt, der dem Erfasser zugeordnet ist. Enthält das erfasserabhängige Feld keinen Wert für den Erfasser, dann wird der Feldwert zusammen mit dem Erfasser eingefügt.

  Wenn das Feld nicht vorhanden ist, dann muß es in *fields* eingefügt werden. Der entsprechende Feldwert wird so erzeugt, wie es im Abschnitt 5.3.5 beschrieben wurde.

Im nächsten Schritt muß der Nutzer für jedes Tupel in der Menge `diff_fields` angeben, ob der neue Wert oder der alte Wert für die Eigenschaft genutzt werden soll. Wenn der neue Wert ausgewählt wurde, dann wird die Eigenschaft zusammen mit dem Wert in die Menge `change_fields` eingefügt.

Anschließend müssen die Eigenschaften, die in der Menge `change_fields` vorhanden sind, geändert werden. Die einzelnen Tupel dieser Menge bestehen aus den Komponenten `feature` und `value`. Im folgenden wird für die einzelnen Eigenschaften beschrieben, wie die Änderungen vorzunehmen sind:

- Eintragstyp: Der Eintrag muß reklassifiziert werden (siehe Abschnitt 5.6).
- Schlüsselwort: Das Schlüsselwort wird `value` entnommen. Es muß geprüft werden, ob das Schlüsselwort vom Erfasser bereits für ein anderen Eintrag angegeben wurde. Ist dies der Fall, dann muß der Nutzer ein anderes Schlüsselwort definieren.

  Das Schlüsselwort wird zusammen mit dem `submitter`-Objekt in das mengenwertige Attribut *submitter_key* des `old_entry`-Objektes eingefügt.
- *language*: Die Angabe in `value` wird dem Attribut *language* des `old_entry`-Objektes zugewiesen.
- *other_authors*: Dem Attribut *other_authors* des `old_entry`-Objektes wird `value` zugewiesen.
- *other_editors*: Die Angabe in `value` wird dem Attribut *other_editors* des `old_entry`-Objektes zugeordnet.
- *crossref*: Dem Attribut *crossref* des `old_entry`-Objektes wird `value` zugewiesen.
- Felder: Für jedes Feld, das im mengenwertigen Attribut *fields* des `old_entry`-Objektes enthalten ist, wird geprüft, ob es in der Menge `change_fields` vorkommt. Wenn das Feld vorhanden ist, dann wird

  - bei einem erfasserabhängigen Feld der Wert, der dem Erfasser zugeordnet ist, durch `value` ersetzt, und
  - bei den übrigen Feldern ein entsprechendes Objekt für den Feldwert erzeugt bzw. ermittelt (wie in Abschnitt 5.3.5 beschrieben).

5.4 Änderung von Einträgen

Die Angaben, die einem Eintrag zugeordnet sind, müssen bei Bedarf geändert werden können. Die folgenden Änderungen sollten möglich sein:

- Löschen von einzelnen Autoren und Herausgebern,
- Hinzufügen von Autoren und Herausgebern,
- Änderung der Angabe, ob ein Dokument weitere Autoren bzw. Herausgeber enthält, die aber nicht im einzelnen spezifiziert wurden (Änderung der booleschen Attribute *other_authors* und *other_editors* eines *Entry*-Objektes),
- Änderung der Sprache eines Eintrags,
- Änderung des Querverweises (Auswahl eines anderen Eintrags, der in der Datenbank vorhanden ist),
- Änderung des eigenen Schlüssels bzw. Angabe eines eigenen Schlüssels, wenn der Eintrag nicht vom Nutzer importiert wurde,
- Hinzufügen und Löschen von Feldern sowie
- Änderung der Feldwerte der vorhandenen Felder.

Für einen Autoren bzw. Herausgeber müssen die einzelnen Namensbestandteile angegeben werden. Bei erfasserabhängigen Feldern ist zu beachten, daß nur Werte, die dem Nutzer zugeordnet sind, geändert werden dürfen. Ein freies Feld kann ebenfalls nur geändert werden, wenn es dem Nutzer zugeordnet ist. Die Feldwerte werden durch Objekte dargestellt. Im Abschnitt 5.5 wird beschrieben, wie die Bestimmung des eigentlichen Wertes eines Feldes erfolgt.

Nach der Änderung der Angaben eines Eintrags wird in der Datenbank nach potentiell gleichen Einträgen gesucht. Konnten solche Einträge ermittelt werden, dann muß der Nutzer entscheiden, ob sie wirklich gleich sind. Das Vorgehen bei gleichen Einträgen wurde bereits im Unterabschnitt 5.3.6 beschrieben, wobei in diesem Fall der geänderte Eintrag mit `new_entry` bezeichnet wird. Das `new_entry`-Objekt muß anschließend gelöscht werden. Existiert kein gleicher Eintrag, dann werden die geänderten Angaben den Attributen des *Entry*-Objektes so zugewiesen, wie es im Abschnitt 5.3.5 dargestellt wurde.

5.5 Bestimmung der Feldwerte

In der Menge *fields* eines *Entry*-Objektes ist dem Feldwert ein Objekt aus einer Unterklasse von *Field_name* zugeordnet. Wie aus diesem Objekt der eigentliche Feldwert ermittelt wird, soll im folgenden für die einzelnen Unterklassen dargestellt werden:

- *Abbrevation*: Die Werte des *Abbrevation*-Objektes sind im mengenwertigen Attribut *value* enthalten. In diesem Attribut wird ein Tupel gesucht, in dem die Komponente *language* mit der Sprache, die dem Eintrag zugeordnet ist, übereinstimmt. Konnte ein Tupel ermittelt werden, dann wird die zugehörige Langform der Abkürzung als Feldwert genutzt. Im anderen Fall wird dem Feldwert die englische Langform zugeordnet.

- *Publisher*: Das Attribut *name* enthält ein *Abbrevation*- oder *Single_value*-Objekt. Der Feldwert des Objektes wird entsprechend bestimmt. Wenn dem *address*-Attribut keine leere Zeichenkette zugeordnet ist, dann wird es an den Verlagsnamen angehängt (durch ein Komma getrennt).

- *Journal*: Dem Attribut *name* ist *Abbrevation*- oder *Single_value*-Objekt zugeordnet. Der Feldwert des Objektes muß entsprechend bestimmt werden.

- *Submitter_depend_value*: Diesem Objekt ist eine Menge zugeordnet. In den einzelnen Tupeln dieser Menge wird für einen Erfasser der zugehörige Feldwert festhalten. Wenn der Nutzer in dieser Menge als Erfasser vorkommt, dann wird der zugeordnete Feldwert genutzt. Im anderen Fall ist der Feldwert eine leere Zeichenkette.

- *Single_value*: Der Feldwert ist im Attribut *value* des *Single_value*-Objektes enthalten.

5.6 Reklassifizierung von Einträgen

Reklassifizierung bedeutet, daß der Typ eines Eintrags geändert wird. Jeder Eintragstyp wird durch eine Unterklasse von *Entry* beschrieben. Aus diesem Grund muß ein Eintrag bei einer Änderung des Eintragstyps einer anderen Klasse zugeordnet werden.

In dem objektorientierten Datenbankmodell, das in Abschnitt 3.1 beschrieben wurde, ist ein Objekt in der Lage, die Klasse wechseln. Dies bedeutet, daß sich ein Objekt in eine Unterklasse hinein bzw. aus einer Unterklasse heraus bewegen kann.

Somit kann ein Eintrags-Objekt bei einer Änderung des Eintragstyps der entsprechenden Klasse zugeordnet werden. Kann für den neuen Eintragstyp keine Klasse bestimmt werden, dann wird das Eintrags-Objekt in die Klasse *Other_type* bewegt. Ein Eintrags-Objekt besitzt das Attribut *name*, wenn es der Klasse *Other_type* zugeordnet ist. In diesem Attribut wird der Eintragstyp festgehalten.

5.7 Anfragen

Das Datenbanksystem stellt verschiedene Möglichkeiten für die Suche nach Einträgen bereit. In den folgenden Abschnitten werden diese Möglichkeiten näher erläutert. Im Abschnitt 5.7.4 wird eine Suchfunktion konzipiert, die für die Suche nach bestimmten Feldern genutzt werden kann.

Bei jeder Suche kann der Suchraum auf eine Unterklasse von *Entry* eingeschränkt werden, d.h. wenn nur Artikel mit bestimmten Eigenschaften ermittelt werden sollen, dann wird die Suche auf Objekte aus der Klasse *Article* beschränkt.

Ausgabe des Ergebnisses Die ermittelten Einträge kann man nach bestimmten Kriterien gruppieren:

- *Eintragstyp*: Das Anfrageergebnis wird nach dem Klassennamen der Objekte gruppiert.
- *Jahr*: Die Gruppierung des Ergebnisses erfolgt nach dem Erscheinungsjahr.
- *Eintragstyp und Jahr*: Es wird zuerst eine Gruppierung nach dem Klassennamen der ermittelten Objekte vorgenommen und anschließend erfolgt dann eine Gruppierung nach dem Erscheinungsjahr.

Außerdem kann das Suchergebnis nach den folgenden Eigenschaften sortiert werden:

- *Eintragstyp*: Das Ergebnis wird nach dem Namen der Klasse, der die einzelnen Objekte angehören, sortiert.
- *Titel*: Die Sortierung erfolgt nach dem Feld *title*. Einträgen, die dieses Feld nicht besitzen, wird eine leere Zeichenkette als Titel zugeordnet.
- *Autoren*: Der Name des Autoren, der als erstes in der Autorenliste vorkommt, wird für die Sortierung genutzt. Der Autorenname wird in der Reihenfolge *von-Teil*, *Nachname*, *Vorname*, *Jr-Teil* zusammengesetzt. Einträgen ohne Autorenangabe wird eine leere Zeichenkette zugeordnet.

- *Herausgeber*: Das Suchergebnis wird nach dem Herausgeber, der als erstes für den Eintrag angegeben wurde, sortiert. Der Name des Herausgebers wird genauso ermittelt wie der des Autoren.
- *Jahr*: Die Sortierung erfolgt nach dem Jahr, in dem ein Eintrag herausgegeben wurde.

Ob die Sortierung aufsteigend oder absteigend erfolgt, kann ebenfalls vom Nutzer angegeben werden.

5.7.1 Suchen nach Autoren

Diese Funktion dient der Suche nach Einträgen, deren Autorenlisten bestimmte Autoren enthalten bzw. nicht enthalten.

Man kann mehrere Autoren angegeben, die mit UND oder ODER verknüpft werden. Für den Autorennamen sind die Bestandteile *Vorname*, *von-Teil*, *Nachname* und *Jr-Teil* zu spezifizieren. Die einzelnen Namensbestandteile können Wildcards (*) enthalten. Die Suche erfolgt dann über eine Ungewißheitsselektion. Außerdem können Autorenangaben negiert werden, d.h. diese Autoren dürfen nicht in der Autorenliste eines Eintrags vorkommen.

5.7.2 Suchen nach Eintragsschlüsseln

Über diese Funktion können Einträge ermittelt werden, die von einem Nutzer mit einem bestimmten Schlüsselwort erfaßt wurden.

Der Erfassername ist durch die einzelnen Bestandteile zu spezifizieren. Es können mehrere Schlüsselwörter angegeben werden, die Verknüpfung erfolgt dann über die ODER-Funktion.

5.7.3 Suchen aller Einträge für einen Erfasser

Mit dieser Funktion können alle Einträge ermittelt werden, die von einer bestimmten Person erfaßt wurden.

Der Erfasser ist durch die einzelnen Namensbestandteile anzugeben.

5.7.4 Allgemeine Suchfunktion für Felder

Jedes *Entry*-Objekt muß eine allgemeine Suchfunktion besitzen. Sie wird verwendet, um in der Datenbank nach Einträgen zu suchen, die Felder mit einem bestimmten Wert besitzen. Die Suchfunktion hat die folgenden Argumente:

- *field_name*: Dieses Argument enthält den Namen des Feldes.
- *field_value*: In diesem Argument ist der gesuchte Wert des Feldes enthalten.
- *like*: Dieses Argument ist „wahr“, wenn bei der Suche die Ungewißheitsselektion angewendet werden soll. In diesem Fall enthält das Argument *field_name* sogenannte Wildcards (*), die für beliebige Zeichenkette stehen.
- *negation*: Dieses Argument ist „wahr“, wenn das Ergebnis der Suche negiert werden soll.

Der Rückgabewert der Funktion ist ein Wahrheitswert. Er ist „wahr“, wenn der Eintrag die Suchkriterien, die über die Argumente spezifiziert wurden, erfüllt.

Umsetzung Für die Bestimmung des Suchergebnisses müssen die folgenden Aktionen durchgeführt werden:

1. In der Menge *fields* des Objektes wird nach einem *Field_name*-Objekt gesucht, dessen Attribut *name* mit dem Argument *field_name* übereinstimmt.
2. Wurde ein entsprechendes Objekt gefunden, dann wird der Wert des Feldes mit dem Argument *field_value* verglichen. Der Feldwert ist ein *Field_value*-Objekt. Wenn das Argument *multiple_value* des Objektes „wahr“ ist, muß unabhängig vom *like*-Argument eine Ungewißheitsselektion angewendet und dem Argument *field_value* ein * voran- und nachgestellt werden.
   (a) Bei einem Standardfeld hängt der Ablauf des Vergleichs davon ab, welcher Klasse das *Field_value*-Objekt angehört:
      - *Abbrevation*: Das Argument *field_value* wird mit den Kurz- und Langformen aller Sprachen, die im *Abbrevation*-Objekt enthalten sind, verglichen. Wenn eine Übereinstimmung gefunden wurde, wird die Suche abgebrochen und das Suchergebnis auf „wahr“ gesetzt.
      - *Publisher*: Das Attribut *name* enthält ein *Abbrevation*- oder *Single_value*-Objekt. Für dieses Objekt wird ein Suchergebnis ermittelt. Anschließend erfolgt ein Vergleich von *field_value* und dem Attribut *address* des *Publisher*-Objektes. Die beiden Suchergebnisse werden dann über die ODER-Funktion miteinander verknüpft.
      - *Journal*: Dem *name*-Attribut des *Journal*-Objektes ist ein *Abbrevation*- oder *Single_value*-Objekt zugeordnet. Für dieses Objekt wird das Suchergebnis bestimmt.

- *Submitter_depend_value*: Das Objekt besteht aus einer Menge von Tupeln, die die Komponenten *submitter* und *value* enthalten. Für jedes Tupel wird *field_value* mit der *value*-Komponente verglichen. Wenn eine Übereinstimmung gefunden wurde, wird die Suche abgebrochen und das Suchergebnis auf „wahr“ gesetzt.
- *Single_value*: *Field_value* wird mit dem Attribut *value* des Objektes verglichen.

(b) Bei einem freien Feld wird das Argument *field_value* mit dem Attribut *value* des zugeordneten *Single_value*-Objektes verglichen. Außerdem wird die Suche in *fields* nach *Field_name*-Objekten, deren Attribut *name* mit dem Attribut *field_name* übereinstimmt, fortgesetzt.

(c) Wenn das Argument *field_name* den Wert `address` enthält und kein entsprechendes Feld gefunden wurde, dann muß *field_value* im Feld *publisher* gesucht werden.

3. Wenn kein Feld gefunden wurde und das Argument *field_value* eine leere Zeichenkette ist, dann wird das Suchergebnis auf „wahr“ gesetzt.

4. Enthält der Eintrag mehrere freie Felder, die mit *field_name* übereinstimmen, muß für jedes dieser Felder ein Ergebnis bestimmt werden. Die Verknüpfung der einzelnen Ergebnisse erfolgt dann über die ODER-Funktion.

5. Ist das Argument *negation* „wahr“, wird das Ergebnis negiert.

5.7.5 Suchen nach Themen

Diese Funktion dient der Suche nach Einträgen, die bestimmten Themen zugeordnet werden können.

Man kann mehrere Themen angeben, die mit UND oder ODER verknüpft werden. Ein Eintrag wird einem Thema zugeordnet, wenn es in den Feldern *title*, *booktitle*, *abstract*, *keywords*, *note*, *annote* vorkommt. Bei der Überprüfung, ob ein Thema in einem Feld enthalten ist oder nicht, muß die Ungewißheitsselektion angewendet werden.

5.7.6 Nutzerdefinierte Anfragen

Diese Funktion kann zur Ermittlung von Einträgen, deren Felder bestimmte Bedingungen erfüllen, genutzt werden.

Es können mehrere Felder angegeben werden, die bestimmte Werte haben sollen. Die Verknüpfung der einzelnen Felder erfolgt durch UND oder ODER. Eine Feldangabe kann negiert werden, d.h. es wird dann nach Einträgen gesucht, bei denen

sich der Wert des Feldes vom spezifizierten Wert unterscheidet. Bei Feldwerten, die Wildcards enthalten, wird bei der Suche die Ungewißheitsselektion verwendet.

5.8 Export-Schnittstelle

Die durch eine Anfrage ermittelten Einträge können in die verschiedenen Formate exportiert werden. Dazu muß der Nutzer festlegen, in welches Format der Export erfolgen soll, und er muß angegeben, ob die Abkürzungen durch die Kurz- oder Langform ersetzt werden sollen.

Die exportierten Einträge werden in ein HTML-Dokument geschrieben. Der Header eines HTML-Dokuments enthält mehrere Felder. Damit das HTML-Dokument im lokalen Dateisystem gespeichert wird, muß das Feld `content-type` einen Typ enthalten, der dem WWW-Client nicht bekannt ist (z.B. `export`). Der WWW-Client öffnet dann ein Dialogfenster, in dem dann der Name der Datei angegeben werden kann, in der die exportierten Einträge zu speichern sind.

In den folgenden Abschnitten wird dargestellt, wie der Export von Einträgen in die verschiedenen Formate ablaufen muß.

5.8.1 Export in das BibTeX-Format

Beim Export von Einträgen in das BibTeX-Format ist folgendes zu beachten: Besitzt ein Eintrag einen Querverweis, dann muß auch der Eintrag, auf den verwiesen wird, exportiert werden (siehe Abschnitt 2.1.8).

Zur Bestimmung der zu exportierenden Einträge werden die Mengen `entry` und `crossref` eingeführt. In der Menge `crossref` werden die Einträge festgehalten, auf die verwiesen wird. Der Export wird in folgenden Schritten durchgeführt:

- Die Einträge, die zu exportieren sind, werden in die Menge `entry` eingefügt.
- Besitzt ein Eintrag einen Querverweis, dann wird der Eintrag, auf den verwiesen wird, in die Menge `crossref` eingefügt. Dabei ist zu beachten, daß ein Eintrag nicht doppelt in der Menge `crossref` vorkommt.
- Alle Einträge in der Menge `entry`, die nicht in `crossref` enthalten sind, werden exportiert.
- Zum Schluß mußsen dann alle Einträge in der Menge `crossref` exportiert werden.

Der Export eines einzelnen Eintrags läuft wie folgt ab:

- Im ersten Schritt erfolgt die Ausgabe der Eintragtyps in der Form

  `@`*Eintragstyp*`{`

  Der Eintragstyp stimmt mit dem Namen der Klasse überein bis auf eine Ausnahme: Ist der Eintrag der Klasse *Other_type* zugeordnet, dann wird der Typ dem Attribut *name* des Eintragsobjektes entnommen.

- Als nächstes wird das Schlüsselwort des Eintrags in Abhängigkeit vom Nutzer bestimmt. Ist der Nutzer auch Erfasser des Eintrags, so wird das zugeordnete Schlüsselwort verwendet. Ansonsten wird das Schlüsselwort aus den Nachnamen und den zugehörigen Schlüsselwörtern aller Erfasser, getrennt durch einen Doppelpunkt, zusammengesetzt. Dadurch kann sichergestellt werden, daß es eindeutig ist. Folgendes Beispiel soll die Bestimmung des Schlüsselwortes verdeutlichen: Ein Eintrag enthält für den Nutzer `Meier` das Schlüsselwort `Heu95` und für `Schulz` `HS95`. Exportiert `Meier` den Eintrag, dann wird `Heu95` als Schlüsselwort genutzt. Wird der Eintrag dagegen vom Nutzer `Müller` exportiert, dann lautet das Schlüsselwort `Meier:Heu95Schulz:HS95`.

- Der Name eines Autoren wird in der Reihenfolge

  [*von-Teil*] *Nachname*[, *Jr-Teil*][, *Vorname*]

  zusammengesetzt. Die Angaben in den eckigen Klammern werden nur berücksichtigt, wenn die entsprechenden Namensteile nicht leer sind. Die einzelnen Autoren eines Eintrags werden über das Wort `and` miteinander verbunden. Wenn das Attribut *other_authors* des Eintrags „wahr“ ist, dann wird an die Autorenliste die Angabe `and other` angehängt. Die gesamte Autorenangabe ist in der Form

  `author = {`*Autorenangabe*`},`

  auszugeben.

- Die Herausgeberangabe eines Eintrags wird in der gleichen Art und Weise zusammengesetzt wie die Autorenangabe und in der Form

  `editor = {`*Herausgeberangabe*`},`

  ausgegeben.

- Besitzt der Eintrag einen Querverweis, dann muß der Schlüssel des Eintrags, auf den verweisen wird, ermittelt werden. Dabei wird genauso verfahren wie bei der Bestimmung des Schlüsselwortes für den aktuellen Eintrag. Der Querverweis ist in der Form

```
crossref = {Schlüssel},
```

anzugeben.

- Anschließend erfolgt dann der Export der einzelnen Felder des Eintrags, wobei die folgenden Felder gesondert betrachtet werden müssen (wenn sie im Eintrag enthalten sind):
  - *title* und *booktitle*: Alle Buchstaben, die in diesen Felder groß geschrieben sind, werden in geschweifte Klammern eingefaßt, da sie sonst von einigen BibTeX-Stilen beim Erzeugen eines Literaturverzeichnisses in Kleinbuchstaben umgewandelt werden (siehe Abschnitt 2.1.7.3).
  - *publisher*: Diesem Feld ist ein Verlagsobjekt zugeordnet. Das Attribut *Name* dieses Objektes wird als Wert des Feldes *publisher* verwendet. Wenn das Attribut *address* des Verlagsobjektes nicht leer ist, dann wird es als Wert des Feldes *address* genutzt.
  - freie Felder: Ein freies Feld wird nur exportiert, wenn der Nutzer mit dem Erfasser des Feldes übereinstimmt.
  - erfasserabhängige Felder: Kommt in diesem Feld ein Wert vor, der dem Nutzer zugeordnet ist, dann wird dieser Wert exportiert. Schwierigkeiten treten bei der Wahl des Feldwertes auf, wenn vom Nutzer kein Wert angegeben wurde, da die Reihenfolge der Werte zufällig ist und somit keinen Aufschluß darüber gibt, welcher Wert sich am besten für den Export eignet. Aus diesem Grund wird folgende Festlegung getroffen: Der erste Wert, der in einem erfasserabhängigen Feld enthalten ist, wird beim Export verwendet.
  - *report_number*, wenn der Typ des Eintrags *Techreport* ist: Der Wert dieses Feldes wird mit dem Feldnamen *number* exportiert, da die technische Berichtsnummer im BibTeX-Format im Feld *number* enthalten ist (siehe Abschnitt 3.2.2).

  Ein Feld wird in der Form

  Feldname `= {`*Feldwert*`},`

  ausgegeben. Wenn der *Feldwert* durch ein Abkürzungsobjekt dargestellt wird, dann muß es entsprechend der Sprache des Eintrags und der gewählten Form (Kurz- oder Langform) aufgelöst werden.

- Der Export des Eintrags wird mit einer schließenden geschweiften Klammer beendet.

5.8.2 Export in das *refdbms*-Format

In *refdbms* fehlen die Eintragstypen *Booklet*, *Conference*, *Incollection* und *Mastersthesis*. Der Export von Einträgen aus den entsprechenden Klassen ist aber möglich, da sie den vorhandenen *refdbms*-Eintragstypen zugeordnet werden können (siehe Abschnitt 3.2.1). Einträge, die in der Klasse *Other_type* vorkommen, können dagegen nicht exportiert werden. In der Tabelle 5.2 wird ein Überblick darüber gegeben, welchem *refdbms*-Typ die Einträge der vorhandenen Klassen zugewiesen werden.

Klasse	Eintragstyp
Article	Article
Book	Book
Booklet	Miscellaneous
Conference	InProceedings
Inbook	InBook
Incollection	InBook
Inproceedings	InProceedings
Manual	Manual
Mastersthesis	TechReport
Miscellaneous	Miscellaneous
Phdthesis	PhDthesis
Proceedings	Proceedings
Techreport	TechReport
Unpublished	UnPublished

Tabelle 5.2: Zuordnung der *refdbms*-Eintragtypen zu den einzelnen Klassen

Im folgenden wird das Vorgehen beim Export eines Eintrags in das *refdbms*-Format beschrieben:

- In der ersten Zeile eines *refdbms*-Eintrags wird der Eintragstyp in der Form

  `%z` *Eintragstyp*

  angegeben. Die Groß-/Kleinschreibung der Eintragstypen ist signifikant. Es ist sinnvoll, für die einzelnen Klassen eine Methode zu definieren, die den entsprechenden *refdbms*-Typ für die Einträge der Klasse liefert.

- Die zweite Zeile eines *refdbms*-Eintrags enthält das Schlüsselwort des Eintrags. Sie wird in der Form

  `%K` *Eintragstyp*

angegeben. Ist der Nutzer auch Erfasser des Eintrags, so wird das zugeordnete Schlüsselwort verwendet. Ansonsten wird das Schlüsselwort aus den Nachnamen und den zugehörigen Schlüsselwörtern aller Erfasser, getrennt durch einen Doppelpunkt, zusammengesetzt.

- Als nächstes werden die Autoren des Eintrags mit den zugehörigen Informationen angegeben. Jeder Autor wird in der Form

  `%A` [*Vorname*] [*von-Teil* \ ] *Nachname* [, *Jr-Teil*]

  dargestellt. Die Namensteile, die in eckige Klammern eingeschlossen sind, werden nur berücksichtigt, wenn sie nicht leer sind. Bestehen *von-Teil* und *Nachname* aus mehreren Angaben, dann wird den trennenden Leerzeichen ein \ vorangestellt. Ist ein *von-Teil* vorhanden, dann steht ebenfalls ein \ vor dem folgenden Leerzeichen. Wenn der *Jr-Teil* nicht leer ist, dann muß er durch ein Komma von den restlichen Angaben getrennt werden.

  Für ein nichtleeres *note*-Attribut eines Autorentupels gilt: Ist der Autor der Letzte in der Autorenliste bzw. unterscheidet sich das *note*-Attribut von dem des nächsten Autoren, dann ist dieses Attribut unmittelbar nach dem Autoren in der Form

  `%a` *note*

  anzugeben.

  Das *organization*-Attribut eines Autorentupels kann genauso behandelt werden. Es wird in der Form

  `%y` *organization*

  ausgegeben.

- Die Herausgeber eines Eintrags werden in der gleichen Form ausgegeben wie die Autoren. Der einzige Unterschied liegt darin, daß die Herausgeberangabe mit `%E` (statt `%A`) eingeleitet wird.

  Die Anmerkung zu einem Herausgeber wird wie das *note*-Attribut eines Autorentupels behandelt. Sie ist in der Form

  `%e` *note*

  unmittelbar nach dem Herausgeber anzugeben.

- Alle *refdbms*-Felder eines Eintrags außer dem *publisher*-Feld werden in einer Menge gesammelt, die im folgenden `refdbms_fields` genannt wird. Ein *refdbms*-Feld ist jedes Standardfeld-Objekt, dessen *refdbms*-Attribut „wahr“ ist. Die anderen Standardfelder des Eintrags werden der Menge `other_fields` zugeordnet. Frei definierbare Felder spielen beim Export in das *refdbms*-Format keine Rolle. Die Feldwerte werden nicht als Objekt, sondern als Zeichenkette in die Menge `refdbms_fields` eingefügt. Dabei ist folgendes zu beachten:

  - Kommt in einem erfasserabhängigen Feld ein Wert vor, der dem Nutzer zugeordnet ist, dann wird dieser Wert verwendet. Ansonsten wird der erste Wert, der für ein erfasserabhängiges Feld angegeben wurde, in die Menge `refdbms_fields` eingefügt.
  - Ist der Feldwert eine Abkürzung, dann wird diese entsprechend der Sprache des Eintrags und der gewählten Form (Kurz- oder Langform) aufgelöst.

  Enthält der Eintrag einen Querverweis, dann werden die beiden Mengen durch die Felder des Eintrags, auf den verwiesen wird, erweitert: `refdbms_fields` werden alle *refdbms*-Attribute des *crossref*-Eintrags zugeordnet, die noch nicht in dieser Menge vorkommen und `other_fields` erhält alle anderen Standardfelder, die noch nicht in der Menge vorhanden sind.

- Ein *chapter*-Feld in der Menge `other_fields` wird wie folgt behandelt:

  - Ist ein *number*-Feld in `refdbms_fields` vorhanden, dann wird die *chapter*-Angabe an den Wert des Feldes angehängt (abgetrennt durch ein Komma).
  - Wenn kein *number*-Feld vorhanden ist, dann wird es in die Menge `refdbms_fields` mit der *chapter*-Angabe als Wert eingefügt.

- Für ein *edition*-Feld in der Menge `other_fields` gilt dasselbe wie für das *chapter*-Feld. Es ist zu beachten, daß an den Wert des *edition*-Feldes das Wort `edition` angehängt werden muß.

- Sind die Felder *ftp* und *URL* in `other_fields` vorhanden, dann müssen sie folgendermaßen behandelt werden:

  - Die beiden Felder sind erfasserabhängig. Wie die Auswahl eines Wertes bei erfasserabhängigen Feldern erfolgt, wurde bereits beschrieben.
  - Kommt das Feld *location* in `refdbms_fields` vor, dann werden die Angaben in *ftp* und *URL* an den Wert dieses Feldes angehängt. Vor den einzelnen Angaben sollte ein Zeilenumbruch eingefügt werden (*location* ist ein mehrzeiliges Feld).

  - Ist kein *location*-Feld vorhanden, dann wird es in die Menge `refdbms_fields` mit den *ftp*- und *URL*- Angaben eingefügt.

- Das *date*-Feld wird aus den Felder *day*, *month* und *year* in der Form

  `%D` [*day*] [*month*] *year*

  zusammengesetzt. Sind die Felder *day* und *month* nicht in der Menge `other_fields` vorhanden, dann wird nur das Jahr angegeben. Monatsnamen werden durch die ersten 3 Buchstaben und einen nachfolgenden Punkt spezifiziert (Ausnahmen: `May`, `June`, `July`).

- Das *publisher*-Feld besteht im *refdbms*-Format aus den Angaben Verlagsname und Adresse. Der Verlagsname ist in einem der folgenden Felder enthalten: *publisher*, *institution*, *school*, *organization* oder *howpublished*. Im folgenden wird aufgelistet, welches Feld aus `other_fields` in Abhängigkeit vom Eintragstyp für die Angabe des Verlagsnamens genutzt wird:

Miscellaneous:	*howpublished*
TechReport:	*institution*
Manual:	*organization*
PhDthesis, *Mastersthesis*:	*school*
bei allen anderen Eintragstypen:	*publisher*

  Kommt das entsprechende Feld nicht in `other_fields` vor, dann wird der Verlagsname aus einem der anderen Felder übernommen.

  Dem Feld *publisher* ist ein Verlagsobjekt zugeordnet. Der Verlagsname ist im Attribut *name* dieses Objektes enthalten, die Adresse wird im Attribut *address* festgehalten. Bei allen anderen Felder ist die Adresse im Feld *address* gespeichert.

  Wenn ein Verlagsname bestimmt werden konnte, dann wird das *publisher*-Feld in der Form

  `%p` Verlagsname[, Adresse]

  angegeben. Konnte keine Adresse ermittelt werden (wenn kein *address*-Feld in der Menge `other_fields` vorhanden bzw. das *address*-Attribut des Verlagsobjektes leer ist), dann wird nur der Verlagsname ausgegeben.

- Enthält die Menge `refdbms_fields` eines *InProceedings*-Eintrags kein *conference name*-Feld und ist in `other_fields` das Feld *organization* enthalten, dann wird ein *conference name*-Feld in `refdbms_fields` mit der *organization*-Angabe als Wert eingefügt.

- Alle Felder, die in der Menge `refdbms_fields` vorhanden sind, werden in der Form

  %*Feldbuchstabe Feldwert*

  exportiert. Der *Feldbuchstabe* ist im Attribut *refdbms_tag* des jeweiligen Feldobjektes enthalten. Dabei ist zu beachteten, das bei mehrzeiligen Feldern (*%I*, *%k*, *%L*, *%O*, *%o*, *%s*, *%x*) jede Zeile mit %*Feldbuchstabe* eingeleitet werden muß.

- Der Eintrag wird mit einer Leerzeile beendet.

5.8.3 Export in das *NCSTRL*-Format

In das *NCSTRL*-Format können alle Einträge exportiert werden, die den Klassen *Techreport*, *Mastersthesis* und *Phdthesis* zugeordnet sind und die eine technische Berichtsnummer (im Feld *report number*) besitzen. Der Export dieser Einträge läuft in den folgenden Schritten ab:

- Die Autoren eines Eintrags werden in separaten Zeilen in der Form

  %A [*von-Teil*] *Nachname* [*Jr-Teil*][, *Vorname*]

  ausgegeben. Die Angaben in den eckigen Klammern werden nur berücksichtigt, wenn die entsprechenden Namensteile nicht leer sind.

- Alle *NCSTRL*-Felder eines Eintrags außer dem *publisher*-Feld werden in einer Menge gesammelt, die im folgenden `ncstrl_fields` genannt wird. Ein *NCSTRL*-Feld ist ein Standardfeld-Objekt, dessen *ncstrl*-Attribut „wahr“ ist. Die anderen Standardfelder eines Eintrags sind der Menge `other_fields` zuzuordnen. Die Feldwerte werden nicht als Objekt, sondern als Zeichenkette in die Menge `ncstrl_fields` eingefügt. Dabei ist folgendes zu beachten:

  - Kommt in einem erfasserabhängigen Feld ein Wert vor, der dem Nutzer zugeordnet ist, dann wird dieser Wert verwendet. Ansonsten wird der erste Wert, der für ein erfasserabhängiges Feld angegeben wurde, in die Menge `ncstrl_fields` eingefügt.
  - Ist der Feldwert eine Abkürzung, dann wird diese entsprechend der Sprache des Eintrags und der gewählten Form (Kurz- oder Langform) aufgelöst.

- Das *report issuer*-Feld besteht im *NCSTRL*-Format aus den Angaben Herausgeber und Adresse. Bei technischen Berichten ist der Herausgeber im allgemeinen im Feld *institution* enthalten. Bei Diplom- und Doktorarbeiten wird er im Feld *school* festgehalten. Kommt das entsprechende Feld nicht in `other_fields` vor, dann wird der Herausgeber aus einem der Felder *publisher*, *howpublished* bzw. *organization* übernommen.

  Dem Feld *publisher* ist ein Verlagsobjekt zugeordnet. Der Herausgeber ist im Attribut *name* dieses Objektes enthalten, die Adresse wird im Attribut *address* festgehalten. Bei allen anderen Felder ist die Adresse im Feld *address* gespeichert.

  Wenn ein Herausgeber bestimmt werden konnte, dann wird das *report issuer*-Feld in der Form

  `%I` Herausgeber[, Adresse]

  angegeben. Konnte keine Adresse ermittelt werden (wenn kein *address*-Feld in der Menge `other_fields` vorhanden bzw. das *address*-Attribut des Verlagsobjektes leer ist), dann wird nur der Herausgeber ausgegeben.

- Das *date*-Feld wird aus den Felder *day*, *month* und *year* in der Form

  `%D` ([*month*`,`] | [*month day*`,`]) *year*

  zusammengesetzt. Die Felder *day* und *month* werden nur berücksichtigt, wenn sie in der Menge `other_fields` vorhanden sind. Kommt eines der beiden Felder vor, dann muß dem Jahr ein Komma vorangestellt werden. Der Monatsname ist in der vollständig ausgeschriebenen Form anzugeben.

- Ist kein *modification date*-Feld in der Menge `ncstrl_fields` vorhanden, dann wird dieses Feld aus der aktuellen Systemzeit bestimmt. Die einzelnen Angaben der Systemzeit werden entsprechend dem Format des *modification date*-Feldes (siehe Abschnitt 2.3.1) zusammengesetzt. Das Ergebnis wird dann in der Form

  `%Z` *modification date*

  ausgegeben.

- Wenn das Feld *ftp* in `other_fields` vorhanden ist, dann wird es wie folgt behandelt:

  – Das Feld *ftp* ist erfasserabhängig. Wie die Auswahl eines Wertes bei einem erfasserabhängigen Feld erfolgt, wurde bereits beschrieben.

  - Kommt das Feld *URL* in `ncstrl_fields` vor, dann wird die *ftp*-Angabe an den Wert dieses Feldes angehängt.
  - Ist kein *URL*-Feld vorhanden, dann wird es in die Menge `refdbms_fields` mit der *ftp*-Angabe eingefügt.

- Alle Felder, die in der Menge `ncstrl_fields` enthalten sind, werden in der Form

  %*Feldbuchstabe Feldwert*

  exportiert. Der *Feldbuchstabe* ist im Attribut *ncstrl_tag* des jeweiligen Feldobjektes enthalten.

- Der *NCSTRL*-Eintrag wird durch eine Leerzeile beendet.

Kapitel 6

Umsetzung des Datenbanksystems

In diesem Kapitel wird beschrieben, wie das entwickelte Datenbanksystem in O_2 umgesetzt wird. Der erste Abschnitt beschäftigt sich mit der Architektur des konkreten Datenbanksystems. Anschließend erfolgt dann eine Vorstellung von O_2 und den weiteren Komponenten von O_2, die für die Umsetzung des Datenbanksystems genutzt werden. In den weiteren Abschnitten wird dargelegt, wie einzelne Funktionen des Systems verwirklicht wurden. Es werden aber nur die Funktionen betrachtet, bei deren Umsetzung Abweichungen vom Konzept, das in Kapitel 5 vorgestellt wurde, auftreten. Die Implementierung der Import- und Export-Schnittstelle erfolgte nur für das BibTeX-Format. Bei der Darstellung der Import-Schnittstelle werden auch die Grundlagen, auf denen der BibTeX-Parser basiert, beschrieben.

6.1 Architektur des Datenbanksystems

Die Literaturdatenbank basiert auf O_2. Für die Repräsentation der Datenbank im WWW wird das O_2Web-Modul genutzt. Ein WWW-Client kann über eine spezielle URL das auf dem WWW-Server vorhandene O_2Web-Interface nutzen, um auf Objekte bzw. Methoden eines Objektes der Datenbank zuzugreifen. Über das O_2Web-Interface sind z.B. Updates, Anfragen und Reklassifizierungen möglich. Die Parser für die verschiedenen Formate wurden mit Hilfe von `lex` und `yacc` entwickelt und stellen eigene Programme dar. Diese Programme können über das ODMG C++-Binding auf die O_2-Datenbank zugreifen. Die einzelnen Parser müssen auf dem WWW-Server vorhanden sein, damit sie über das O_2Web-Interface aufgerufen werden können.

Die Architektur des Datenbanksystems ist in der Abbildung 6.1 dargestellt.

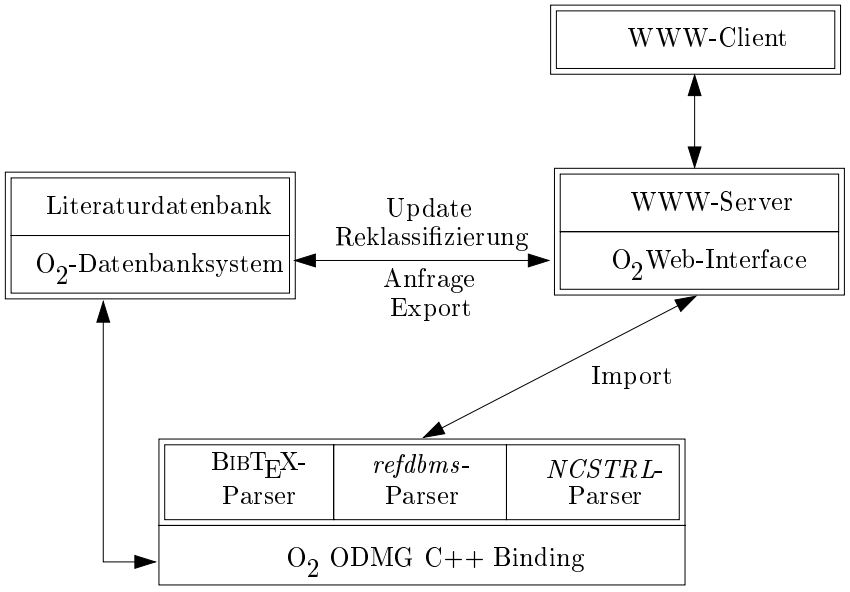


Abbildung 6.1: Architektur

6.2 O_2

O_2 ist ein objektorientiertes Client/Server-Datenbanksystem. Es bietet verschiedene Sprach- und Interaktionsschnittstellen. Neben den traditionellen Sprachen C und C++ gibt es die O_2-eigene Datenbank-Programmiersprache O_2C sowie die Anfragesprache OQL. Aus den Standarddatentypen können mit den Typkonstruktoren `tuple`, `list`, `set` (Multimenge) und `unique set` (Menge) komplexere Typen gebildet werden. Die Typkonstruktoren sind orthogonal anwendbar.

Eine Klasse in O_2 hat einen Zustandstyp und eine Menge von Methoden, sie besitzt aber keine Extension. O_2 unterstützt eine reine Typhierarchie[1] mit Mehrfachvererbung. Typhierarchie bedeutet, daß ein Objekt die Eigenschaften von anderen Klassen erben und selbst nur Mitglied einer Klasse sein kann. Die Signaturen der Methoden werden in der Klassenbeschreibung angegeben. Die Implementierungen erfolgen dann getrennt davon in den Sprachen O_2C, C oder C++. Die Methodensignaturen und -implementierungen werden auf die Unterklassen vererbt. Die ererbten Methoden kann man in den Unterklassen redefinieren.

Damit die erzeugten Objekte in der Datenbank gespeichert werden, müssen sie persistent gemacht werden. Persistenz wird in O_2 durch die Benennung von Objekten oder Werten erreicht, d.h. jedes Objekt, dem zur Laufzeit ein Name zugewiesen wird, ist persistent. Jedes Objekt, daß von einem anderen persistenten

[1] Gegensatz zur Klassenhierarchie, wo ein Objekt in allen Oberklassen vertreten ist. Gehört ein Objekt zu mehreren Klassen, dann besitzt dieses Objekt mehrere lokale Zustände, die zu einem globalen Zustand verbunden werden müssen.

Objekt aus erreicht werden kann, ist ebenfalls persistent. Nicht mehr erreichbare Objekte werden durch eine automatische Freispeicherverwaltung entfernt. O_2 verwirklicht also Persistenz durch Erreichbarkeit. Objekte, die einem Namen zugeordnet wurden, sind Einstiegspunkte in die Datenbank, von denen aus man mittels OQL entlang den Objektbeziehungen durch die Datenbank navigiert.

6.3 O_2Web

Mit dem Modul O_2Web [$O_2$98d] können Web-Clients auf Informationen, die in einer O_2-Datenbank gespeichert sind, zugreifen. Die Informationen, die vom Web-Client präsentiert werden, stellen eine Sicht auf die Objekte der Datenbank dar.

Der Zugriff auf die Objekte, die in der O_2-Datenbank enthalten sind, erfolgt über eine OQL-Anfrage. Diese ist Bestandteil einer speziellen URL, die vom Web-Client an den HTTP-Server geschickt wird:

`http://`*host*`/cgi-bin/o2web_gateway/`*system*`/`*base*`?`*OQL-Anfrage*

Host ist der Name des Internet-Servers, auf dem der HTTP-Server läuft. *System* ist die logische Datenbank, in der die Datenbasis *base* enthalten ist.

O_2Web umfaßt die Elemente O_2Web-Server, O_2Web-Dispatcher und O_2Web-Gateway:

- O_2Web-Server: Der O_2Web-Server nimmt Anfragen vom O_2Web-Gateway entgegen und beantwortet sie.

  Wenn der O_2Web-Server gestartet wird, dann baut er Verbindungen zu einem laufenden O_2Web-Dispatcher und einem laufenden O_2-Datenbanksystem auf.

- O_2Web-Dispatcher: Der Dispatcher verwaltet alle O_2Web-Server, die in einem LAN laufen, und liefert dem O_2Web-Gateway die Adresse eines Servers, der in der Lage ist, eine OQL-Anfrage zu beantworten.

  Zur Auswahl des O_2Web-Servers nutzt der Dispatcher Heuristiken. Für jeden Server im Netzwerk wird eine Bewertung vorgenommen und der Server mit der besten Bewertung wird an das Gateway zurückgeliefert. Die folgenden Faktoren werden zur Bewertung herangezogen:

  - Läuft der Server auf dem gleichen Host wie das Gateway?
  - Ist der Server bereits mit der Datenbank verbunden, an die die Anfrage gestellt wird?

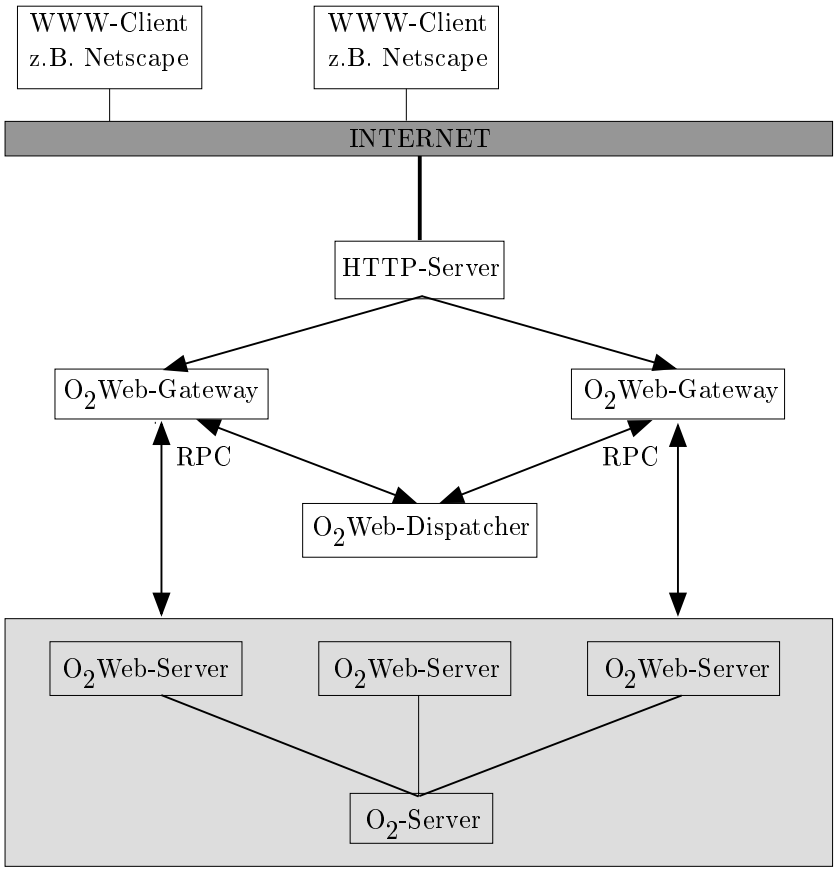


Abbildung 6.2: O_2Web-Architektur

  - Wie hoch ist die aktuelle Auslastung der einzelnen Server (Anzahl der Anfragen, die bearbeitet werden)?

- O_2Web-Gateway: Das Gateway ist ein CGI-Programm, das bei Anfragen an eine Datenbank vom HTTP-Server gestartet wird.

  Das O_2Web-Gateway baut eine Verbindung zum O_2Web-Dispatcher auf und erhält von diesem die Adresse eines O_2Web-Servers, der die Anfrage beantworten kann.

Die Architektur des O_2Web-Systems ist in der Abbildung 6.2 dargestellt. Das O_2Web-System arbeitet in der folgenden Weise:

1. Der Web-Client sendet eine URL im HTTP-Format. Diese URL enthält eine OQL-Anfrage.

2. Der HTTP-Server leitet diese Anfrage an das O_2Web-Gateway weiter.

3. Das O_2Web-Gateway stellt eine Verbindung zum O_2Web-Dispatcher her, der im LAN läuft.

4. Der Dispatcher teilt dem Gateway mit, welcher O_2Web-Server die Anfrage auswerten soll.

5. Das O_2Web-Gateway baut eine Verbindung zu dem entsprechenden O_2-Web-Server auf.

6. Der O_2Web-Server wertet die Anfrage aus, die in der URL spezifiziert wurde, und transformiert das Ergebnis der Anfrage in das HTML-Format.

7. Die ermittelten Daten werden an den Web-Client zurückgesendet.

HTML-Erzeugung Das Modul O_2Web kann in verschiedenen Stufen genutzt werden.

In der ersten und einfachsten Stufe wird es dem Modul O_2Web überlassen, HTML-Dokumente für das Anfrageergebnis zu generieren. In diesem allgemeinen Modus werden die O_2-Objekte durch eine Standard-Darstellung repräsentiert.

Die zweite Stufe erlaubt dem Programmierer, globale Teile der HTML-Generation zu ändern. Für die Daten wird die Standard-Darstellung beibehalten, aber es ist möglich, HTML-Text am Anfang oder am Ende einer Seite einzufügen. In dieser Stufe ist der Programmierer ebenfalls in der Lage, auf Ereignisse wie den Aufbau oder Abbau einer Verbindung zum Server bzw. auf das Auftreten bestimmter Fehler zu reagieren. Die globalen Änderungen werden durch die Definition der folgenden Methoden in der Klasse `O2WebInteractor` erreicht:

- `connect`: Diese Methode wird aufgerufen, wenn eine Verbindung mit einem O_2Web-Server aufgebaut wird.

- `disconnect`: Diese Methode wird nach der Generierung des HTML-Textes für das Anfrageergebnis aufgerufen.

- `error`: Wenn ein Fehler auftritt, dann wird eine Fehlermeldung an den Web-Client zurückgeliefert. Die Fehlermeldungen können mit dieser Methode geändert werden.

- `header`: Der HTML-Text, der in dieser Methode enthalten ist, wird im Kopf jeder HTML-Seite ausgegeben.

- `footer`: In dieser Methode wird der HTML-Text definiert, der am Fuß jeder HTML-Seite ausgegeben wird.

Die dritte Stufe erlaubt die vollständige Steuerung der HTML-Generation für jede Klasse eines Schemas. Somit ist der Programmierer in der Lage, HTML-Text für die Objekte jeder Klasse anzugeben. Der Text kann abhängig vom Zustand eines Objektes bestimmt werden, da er durch eine Methode generiert wird. Der Programmierer kann außerdem für jede Klasse HTML-Text definieren, der am Anfang und am Ende jeder Seite eingefügt wird. Für die lokale Anpassung der HTML-Generation können die folgenden Methoden in einer Klasse redefiniert werden:

- `html_prolog`: Mit dieser Methode wird der MIME-Type des HTML-Dokumentes festgelegt. Außerdem kann diese Methode weitere protokollspezifische Informationen enthalten. Zu diesen Informationen gehört z.B. die Angabe, ob ein HTML-Dokument in den Cache geschrieben werden soll oder nicht.

- `html_header`: Diese Methode ersetzt den globalen HTML-Kopf, der in der Klasse `O2WebInteractor` definiert wurde.

- `html_report`: Durch diese Methode wird die Standard-Darstellung des Objekt-Zustandes ersetzt.

- `html_footer`: Diese Methode ersetzt den globalen HTML-Fuß, der in der Klasse `O2WebInteractor` definiert wurde.

- `html_title`: Mit dieser Methode kann der Text, der bei einem Link auf das Objekt erscheint, geändert werden.

- `get_query`: Diese Methode liefert eine OQL-Anfrage für das aktuelle Objekt. Die OQL-Anfrage kann z.B. genutzt werden, wenn ein HTML-Dokument einen Link auf dieses Objekt enthalten soll.

6.4 O_2 ODMG C++ Binding

Das O_2 ODMG C++-Binding [$O_2$98c, Sch97] bietet eine Schnittstelle zu einer O_2-Datenbank. Durch diese Schnittstelle ist es möglich, in einer C++-Anwendung auf Objekte der einzelnen O_2-Klassen und die zugehörigen Methoden zuzugreifen.

Die O_2-Klassen, die in einer C++-Anwendung genutzt werden sollen, müssen exportiert werden. Dazu stellt O_2 das Werkzeug `o2export` zur Verfügung. Dieses Werkzeug generiert aus den existierenden O_2-Klassendefinitionen Dateien, die dann in einer C++-Anwendung genutzt werden können, um O_2-Objekte so zu manipulieren, als ob sie lokale Objekte wären. Die O_2C-Methoden, die für diese Objekte definiert sind, können ebenfalls genutzt werden.

Typsystem Die C++-Schnittstelle basiert auf einer Menge von „Zwillings"-Klassen, eine Klasse in O_2 und eine in C++. Der Zugriff auf ein persistentes Objekt erfolgt über eine spezielle Klasse in C++, die als persistenter Zeiger bezeichnet wird. Diese Klasse wird beim Export einer O_2-Klasse generiert. Ein persistenter Zeiger ist als parametrisierte Klasse definiert, die mit `d_Ref<T>` bezeichnet wird. Die Objekte, die mit diesem Zeiger referenziert werden, haben alle den gleichen Typ `T`. Beim Dereferenzieren eines `d_Ref`-Objektes wird festgestellt, ob sich das referenzierte Objekt bereits im Adreßraum des Anwendungsprogramms befindet. Ist dies nicht der Fall, dann muß es aus der Datenbank geladen werden. Als Funktionswert wird die Adresse des Objektes zurückgeliefert. Der Programmierer kann somit auf persistente C++-Objekte so zugreifen, als ob sie im Speicher wären. Alle exportierten O_2-Klassen haben die Oberklasse `d_Object`. Die von `d_Object` abgeleiteten Klassen sind persistenzfähig, d.h. ihre Objekte können sowohl persistent als auch transient sein. Die Entscheidung wird bei der Objekterzeugung getroffen. Jedes Objekt, das mit einem persistenten Objekt verbunden wird, ist ebenfalls persistent (siehe Abschnitt 6.2).

Die C++-Objekte können in generischen Collection-Klassen gruppiert werden. Zu diesen Collection-Klassen gehören `d_List`, `d_Set`, `d_Bag` und `d_Varray` (eindimensionales Feld variabler Länge). Diese sind von der abstrakten parametrisierten Klasse `d_Collection` abgeleitet. Alle Objekte, die in einem Collection-Objekt enthalten sind, haben den gleichen Typ. Dieser Typ muß bei der Erzeugung eines Collection-Objektes angegeben werden und kann ein `d_Ref`-Typ oder ein atomarer Typ sein. Da die `d_Collection`-Klasse eine Unterklasse von `d_Objekt` ist, können für die abgeleiteten Collection-Klassen persistente Objekte erzeugt werden.

Kommunikation mit der Datenbank Die Anwendung baut mit der Funktion `d_Session::begin` eine Verbindung zur O_2-Datenbank auf und öffnet die entsprechende Datenbasis mit der Element-Funktion `d_Database::open`. Nachdem Durchführung aller Datenbankoperationen wird die Datenbasis mit der Funktion `d_Database::close` geschlossen und die Verbindung zur Datenbank mit `d_Session::close` beendet.

Transaktionen Das Anwendungsprogramm wird in Transaktionen gegliedert. Jeder Zugriff auf persistente Objekte, also das Erzeugen, Lesen, Modifizieren und Löschen, muß innerhalb einer Transaktion vorgenommen werden. Eine Standard-Transaktion wird mit der Element-Funktion `d_Transaction::begin` gestartet. Die Transaktion endet mit der Funktion `d_Transaction::validate`, `d_Transaction::commit` oder `d_Transaction::abort`. Durch die `commit`- und die `validate`-Funktion werden alle in der Transaktion modifizierten oder erzeugten Objekte mit ihrem neuen Zustand permanent in die Datenbank eingetragen und gelöschte Objekte dauerhaft aus der Datenbank entfernt. Die Transaktion wird

beendet und sämtliche während der Transaktion erworbenen Sperren werden freigegeben. Am Transaktionsende sind alle Änderungen für die anderen Transaktionen sichtbar. Der Unterschied zwischen beiden Funktionen liegt darin, das bei `commit` die mit `d_Ref`-Objekten gespeicherten Verweise auf persistente Objekte nach dem Transaktionsende undefiniert sind. Mittels der `abort`-Funktion werden geänderte Objekte in ihren Ausgangszustand zurückgesetzt, und alle Objekterzeugungen und -löschungen rückgängig gemacht. Diese Funktion beendet ebenfalls die Transaktion und gibt sämtliche Sperren frei. Die Verweise in den `d_Ref`-Objekten sind nach dem Transaktionsende undefiniert.

6.5 Umsetzung des Datenbankschemas

Das Datenbankschema, das in Abschnitt 3.3 vorgestellt wurde, ist in O_2 umsetzbar. Die einzelnen Klassendefinitionen sind im Anhang D vorgestellt. In den Klassendefinitionen sind die Signaturen der Methoden, die für die Klasse benötigt werden, enthalten. Die Methoden-Implementierungen erfolgen in O_2C außerhalb der Klassendefinition. Die Datenbankprogrammiersprache O_2C ist eine Obermenge von C. Sie stellt weitere Möglichkeiten zur Objekt- und Wertdeklaration sowie zur Objekt- und Wertmanipulation durch eine OQL-Schnittstelle und O_2-eigene Primitive bereit.

Im Abschnitt 6.2 wurde bereits erwähnt, das Objekte, die persistent sein sollen, einem persistenten Objekt bzw. einem Namen zugewiesen werden müssen. Ein persistenter *Name* wird durch den Befehl

`name` *Name*: <*Objekt oder Wert*>

definiert. Es können *Objekte* und *Werte* benannt werden. Somit ist es möglich, für eine Klasse einen Namen zu definieren, dem eine Menge von Objekten der Klasse zugeordnet wird. Die Objekte, die diesem Namen zugewiesen werden, sind persistent.

Für alle Klassen des Datenbankschemas, die einen Eintragstyp beschreiben, werden Namen eingeführt. Die Namen dienen ebenfalls als Einstiegspunkte für die Navigation in der Datenbank. Um Anfragen zu ermöglichen, die alle Einträge der Literaturdatenbank berücksichtigen, wird ein Name `entries` eingeführt, in dem alle erzeugten Einträge festgehalten werden. Ein Objekt, das für einen Eintrag erzeugt wird, muß in den entsprechenden Namen der Klasse und in den Namen `entries` eingefügt werden. Außerdem müssen Namen für die Klassen *Person*, *Abbrevation*, *Standard_field*, *Free_field*, *Publisher* und *Journal* definiert werden, da beim Import von Einträgen nach Objekten dieser Klassen gesucht wird. Die Namen werden im folgenden zur besseren Kennzeichnung als Extensionen bezeichnet.

6.6 Übergabe des Nutzers zwischen den HTML-Seiten

Um mit der Literaturdatenbank arbeiten zu können, muß am Anfang der Nutzer erfragt werden, da in den Einträgen nutzerabhängige Elemente existieren. Der Name des Nutzers muß in den Bestandteilen *Vorname*, *von-Teil*, *Nachname* und *Jr-Teil* angegeben werden. Er wird für den Import, die Ausgabe von Einträgen und den Export benötigt. Somit ist eine Übergabe des Nutzernamens zwischen den einzelnen HTML-Seiten, die generiert werden, notwendig. Bei HTML-Seiten, die ein Formular enthalten, ist das kein Problem, da die einzelnen Namensbestandteile in speziellen Formularfeldern übergeben werden können. Diese Felder werden über das `<INPUT TYPE=hidden>`-Tag definiert. Dadurch können Informationen in ein Formular eingebettet werden, die nicht vom Nutzer änderbar sind. Im folgenden werden diese Felder als versteckte Felder bezeichnet. Ein Problem tritt dann auf, wenn in einer HTML-Seite die nächste Seite über einen Link aufgerufen wird. In diesem Fall muß die URL die Nutzerdaten in der Form

`http://.../`*system*`/`*base*`?QuErY:`*OQL-Anfrage*`UsErDaTa:`*Nutzerdaten*

enthalten. Die *Nutzerdaten* werden aus den einzelnen Namensbestandteilen in der Reihenfolge

`first=`*Vorname*`&von=`*von-Teil*`&last=`*Nachname*`&jr=`*Jr-Teil*

zusammengesetzt. Enthält einer der Namensbestandteile das Zeichen `&`, dann muß es kodiert werden. Die Kodierung erfolgt in der Form `%`*xx*, wobei *xx* für den zweistelligen Hexadezimalwert des Zeichens steht. Die URL kann über die Methode `make_url` der `O2WebAssistant`-Klasse erzeugt werden, der die OQL-Anfrage und die zusammengesetzten, kodierten Nutzerdaten zu übergeben sind. Das Objekt, das als Ergebnis der Anfrage ermittelt wird, kann die Nutzerdaten auswerten. Dies ist möglich, da die Nutzerdaten ein Argument der `html_report`-Methode sind, die den entsprechenden HTML-Text für das Objekt generiert. Aus den Nutzerdaten können dann die einzelnen Namensbestandteile herausgefiltert werden. Anschließend müssen die Namensbestandteile dann dekodiert werden (Umwandlung aller `%`*xx*-Zeichenketten in das entsprechende Zeichen).

6.7 Umsetzung der Funktionalität

Erfolgt der Zugriff auf ein Objekt über einen Link, dann wird die Methode `html_report` dieses Objektes aufgerufen. Enthält eine HTML-Seite ein Formular, dann wird das `ACTION`-Attribut des `FORM`-Tags wie folgt definiert:

Rootname->*Method*($0)

Rootname ist ein persistenter Name, dem ein Objekt zugeordnet wurde, und *Method* ist eine Methode dieses Objektes. Die Werte der Formularfelder werden *Method* übergeben, indem der String $0 durch die Angaben des Formulars substituiert wird. Die Signatur einer Methode, die über ein Formular aufgerufen werden kann, muß die Form

Methodenname(*Argument*: bits): bits

haben und als public definiert sein. Die Formularangaben können durch die Klasse O2WebFormAnalyser ermittelt werden.

Die Tabelle 6.1 zeigt, welche Klassen und Extensionen für die Umsetzung der Funktionalität definiert wurden.

Klasse	Extension	Funktion
o2_set_Abbrevation	abbrevation	Ausgabe, Definition sowie Änderung von Abkürzungen
Query	query	Allgemeine Methoden für die Anfragen
Author_query	author_query	Suche nach Autoren
Key_query	key_query	Suche nach Schlüsselwörtern
Submitter_query	submitter_query	Suche aller Einträge eines Erfassers
Topic_query	topic_query	Suche nach Einträgen, die bestimmten Themen zugeordnet werden können
Userdef_query	userdef_query	Suche nach Einträgen, deren Felder bestimmte Werte enthalten
Copy	copy	Steuerung des Imports einer BibTeX-Datei
Start	start	Generierung der Startseite für die Literaturdatenbank, Eingabe des Nutzernamens
Menu	menu	Ausgabe des Menüs
Reclassify	reclassify	Reklassifizierung von Einträgen
Change	change	Änderung von Einträgen
o2_set_Tmp	tmp	Einfügen der Einträge, die nach dem Import in dieser Extension vorhanden sind

Tabelle 6.1: Klassen für die Umsetzung der Funktionalität

Für jede Klasse muß vor der ersten Benutzung der Literaturdatenbank ein Objekt erzeugt werden, das dann der entsprechenden Extension zugeordnet wird.

Bei der Ausgabe von Werten in den Eingabefeldern eines Formulars müssen die Zeichen > und " kodiert werden, da sie in HTML eine bestimmte Bedeutung haben. Die Kodierung dieser Zeichen erfolgt im Format `&#`*nnn*, wobei *nnn* für den Zahlenwert des betreffenden Zeichens steht.

6.8 Import von BIBTEX-Dateien

Der Import einer BIBTEX-Datei erfolgt in vier Schritten:

1. Im ersten Schritt wird eine HTML-Seite generiert. Diese Seite enthält ein Formular, in dem der Name des Erfassers, die Sprache, in der die Einträge sind, sowie die zu analysierende BIBTEX-Datei angeben werden (siehe Abschnitt 5.3). Wird der Ausführungsknopf des Formulars betätigt, dann generiert der WWW-Client aus den Eingaben und der BIBTEX-Datei eine `multipart/form-data`-Mitteilung.

2. Der WWW-Client ruft eine Methode des Objektes `Copy` auf. Diese Methode liest dann die Mitteilung über die `get_raw_data`-Methode eines `O2WebForm-Analyser`-Objektes.

3. Anschließend werden aus der gelesenen Mitteilung die einzelnen Namensbestandteile des Erfassers, die Sprache und die Orginaldatei herausgefiltert. Die Orginaldatei muß auf dem WWW-Server gespeichert werden, damit sie vom Parser analysiert werden kann.

4. Der BIBTEX-Parser wird aufgerufen. Für die Einträge, die während der Analyse bestimmt wurden, werden Objekte erzeugt. Konflikte, die zwischen den eingelesenen Einträgen und den bereits vorhandenen Einträgen auftreten können, werden in diesem Schritt nicht beachtet, da keine Interaktion zwischen dem Parser-Programm und dem Nutzer möglich ist. Die erzeugten Objekte müssen somit erst „temporär“ gespeichert werden, d.h. sie werden nicht der Extension der Klasse zugeordnet. Damit die Objekte persistent sind, werden sie der Extension `tmp` zugewiesen.

5. Im vierten Schritt werden alle Objekte aus der Extension `tmp`, die dem Nutzer zugeordnet sind, in die Extension der Klasse eingefügt. Für das Einfügen werden Methoden des Objektes `Copy` genutzt. Der Nutzer kann Entscheidungen für die Lösung der Konflikte, die beim Einfügen auftreten können, treffen, da jetzt eine Interaktion mit ihm möglich ist.

Die ersten beiden Schritte wurden bereits im Abschnitt 5.3 beschrieben. Der dritte und der vierte Schritt werden in den folgenden Abschnitten näher betrachtet. Im dritten Schritt werden auch die Grundlagen für den BibTeX-Parser vorgestellt.

6.8.1 Aufruf des Parsers

Bei WWW-Clients, die javafähig sind und die diese Eigenschaft nicht ausgeschaltet haben, wird ein Java-Applet ausgeführt. Das Applet ruft ein Skript auf, in dem wiederum der Parser aufgerufen wird. Das Applet kann den Parser aus Sicherheitsgründen nicht selber aufrufen, da es lokal mit den Rechten des Nutzers ausgeführt wird. Ein Skript dagegen wird auf dem WWW-Server ausgeführt. Die Meldungen vom Parser landen auf der Standardausgabe des Skripts und können somit vom Applet gelesen werden. Dadurch ist das Applet in der Lage, die Parser-Meldungen auszugeben und die Anzahl der eingelesenen Einträge fortlaufend zu aktualisieren.

Kann der Parser kein Applet ausführen, dann wird eine HTML-Seite ausgegeben, die einen Link auf ein CGI-Skript besitzt. Durch die Aktivierung des Links wird das CGI-Skript angestoßen, in dem dann der Parser aufgerufen wird. Die Meldungen des Parsers werden auf der HTML-Seite, die vom Skript generiert wird, ausgegeben.

6.8.1.1 BibTeX-Parser

Der Parser wurde mit den Werkzeugen `lex` und `yacc` entwickelt. `Yacc` liest eine Grammatikspezifikation und generiert daraus einen Parser. Mit `lex` wird der Scanner generiert, der für die Erkennung der Terminale und bestimmter Nichtterminale in der Grammatik verantwortlich ist.

Im folgenden wird die Grammatik für eine BibTeX-Datei angegeben. Für die Grammatik gilt: Nichtterminale, die nicht auf einer linken Regelseite auftreten, sondern über die lexikalische Analyse bestimmt werden, sind in der `Schreibmaschinenschrift` dargestellt.

bib_file	::=	{'@' *entry* | `comment`}
entry	::=	**preamble** preample_entry
	|	**string** string_entry
	|	*bib_entry*
preamble_entry	::=	'{' *value* '}'
		'(' *value* ')'
string_entry	::=	'{' *assignment* '}'
		'(' *assignment* ')'
bib_entry	::=	*entry_type entry_body*

entry_type ::= **article**
| **book**
| **booklet**
| **conference**
| **inbook**
| **incollection**
| **inproceedings**
| **manual**
| **mastersthesis**
| **misc**
| **phdthesis**
| **proceedings**
| **techreport**
| **unpublished**
| `other_type`

entry_body ::= '**{**' [`key`] [',' *assignment_list*] '**}**'
| '(' [`key`] [',' *assignment_list*] ')'

assignment_list ::= *assignment* {',' [*assignment*]}

assignment ::= (**author** | **editor**) author_or_editor
| `abbrev` '=' *value*

value ::= *simple_value* {'#' *simple_value*}

simple_value ::= `abbrev`
| `string`
| `number`

author_or_editor ::= *single_name* {**and** *single_name*}

single_name ::= **other**
| *namen* {*next_part*}
| *von_part last_part* ',' *jr_first_part*

namen ::= *name_part* {*name_part*}

next_part ::= ',' *jr_first_part*
| *von_part last_part*

von_part ::= `lower_name` {`lowername` }

last_part ::= *name_part* {*name_part*}

jr_first_part ::= {*name_part*} [',' *first_part*]

first_part ::= {*name_part*}

name_part ::= `name`
| `name opt_part`
| '**{**' {*braced_name*} '**}**'

braced_name ::= *name_part*
| `lower_name`
| **and**
| **other**
| ','

Wenn ein syntaktischer Fehler in der BibTeX-Datei auftritt, dann kann der Parse-Vorgang beim nächsten `@`-Zeichen fortgesetzt werden. Wie die Nichtterminale, die in der Grammatik vorkommen, bei der lexikalischen Analyse erkannt werden können, wird in der Tabelle 6.2 beschrieben.

Alle eingelesenen Einträge werden in Speicherstrukturen festgehalten. Erst nach dem Einlesen aller Einträge der Datei werden die entsprechenden Objekte für die Einträge erzeugt. Die Gründe dafür sind folgende:

- Der Parser muß bei jedem Eintrag überprüfen, ob das Schlüsselwort eindeutig ist. Es muß also mit den Schlüsselwörtern der zuvor eingelesenen Einträge verglichen werden.
- BibTeX-Einträge können Querverweise enthalten. Der Eintrag, auf den verwiesen wird, muß in der Datei nach dem letzten Eintrag stehen, der auf ihn verweist. Somit kann die Gültigkeit von Querverweisen erst festgestellt werden, wenn alle Einträge der Datei eingelesen wurden.

Das Objekt für einen Eintrag, auf den verwiesen wird, sollte vor dem ersten Eintrag x erzeugt werden, der auf ihn verweist. Dadurch steht bei der Erzeugung des Objektes für den Eintrag x der Wert für das Attribut *crossref* bereits fest.

6.8.1.2 Datenbankschnittstelle

Beim Import ist keine Interaktion zwischen Parser und Nutzer möglich, da der Parser ein eigenes Programm ist und dadurch nicht vom WWW-Client gesteuert werden kann. Es ist besteht nur die Möglichkeit, daß die Ausgaben vom Parser auf den WWW-Client umgeleitet werden. Eine Interaktion ist aber erforderlich, um auftretende Konflikte lösen zu können. Aus diesem Grund wird die Extension `tmp` definiert, in die die erzeugten Objekte für die eingelesen Einträge eingefügt werden. Dabei wird nicht auf Konflikte geachtet.

Wenn ein vorheriger Import-Vorgang vom Nutzer abgebrochen wurde, dann können in `tmp` noch Einträge enthalten sein, die dem Nutzer zugeordnet sind. Solche Einträge müssen vom Parser zu Beginn gelöscht werden.

Zu beachten ist, daß bei der Erzeugung eines Objektes aus der Klasse *Free_field* das Attribut *new_field* auf „wahr“ gesetzt werden muß, da dieses Attribut beim Verschieben eines Eintrags ausgewertet wird.

6.8.2 Verschieben der Einträge

Nach Beendigung des Parser-Programms werden die Eintrags-Objekte aus der Extension `tmp`, die dem Erfasser zugeordnet sind, in die Extensionen der Klassen

Nichtterminal	Bedeutung
`comment`	– alle Zeichen außer `@`
`other_type`	– alle Buchstaben und Zeichen außer `" # % ' ( ) , = { }` – das Nichtterminal kann außer an der ersten Stelle auch Ziffern enthalten – das Nichtterminal endet, wenn eines der Zeichen `" # % ' ( ) , = { }` bzw. ein Leerzeichen, ein Zeilenumbruch oder ein Tabulatorzeichen auftritt
`key`	– alle Buchstaben, Ziffern und Zeichen außer Kommata, schließenden geschweiften Klammern, Leerzeichen, Zeilenumbrüchen oder Tabulatorzeichen
`abbrev`	– siehe `other_type`
`string`	– es gilt folgende Vereinbarung: eine öffnende geschweifte Klammer erhöht die Klammerebene und eine schließende geschweifte Klammer erniedrigt die Klammerebene – ein `string` beginnt mit einer öffnenden geschweiften Klammer und endet mit einer schließenden geschweiften Klammer auf der gleichen Klammerebene, oder – der `string` beginnt mit einem Anführungsstrich und endet mit einem Anführungsstrich auf der gleichen Klammerebene
`number`	– zusammenhängende Ziffernfolge
`lower_name`	– an erster Stelle steht ein kleiner Buchstabe, gefolgt von beliebigen weiteren Buchstaben
`name`	– an erster Stelle steht ein großer Buchstabe, gefolgt von beliebige Buchstaben
`opt_part`	– Nichtterminal beginnt mit einer öffnenden eckigen Klammer und endet mit einer schließenden eckigen Klammer, dazwischen können beliebige Buchstaben auftreten

Tabelle 6.2: Bedeutung der Nichtterminale in der BIBTEX-Grammatik

und in `entries` eingefügt. Dies wird über Methoden des Objektes `Copy` aus der Datenbank gesteuert. Vor dem Einfügen eines Eintrags muß geprüft werden, ob Konflikte mit bereits vorhandenen Einträgen auftreten. Tritt ein Konflikt auf, dann wird er in Zusammenarbeit mit dem Nutzer gelöst, da jetzt eine Interaktion mit ihm möglich ist. Im folgenden wird die Vorgehensweise beim Einfügen eines Eintrags beschrieben:

1. Im ersten Schritt wird geprüft, ob der Eintrag ein frei definierbares Feld enthält, das zum ersten Mal vom Erfasser genutzt wurde. Dazu wird das Attribut *new_field* ausgewertet. Ist es „wahr“, dann wurde das freie Feld zum ersten Mal verwendet. In diesem Fall müssen vom Nutzer Informationen zu diesem Feld erfragt werden. Dazu wird eine HTML-Seite generiert, die ein Formular enthält. In diesem Formular kann der Nutzer die Bedeutung dieses freien Feldes spezifizieren und angeben, ob der Wert des Feldes aus einer Liste voneinander unabhängiger Informationen besteht oder nicht. Das Formular muß das Schlüsselwort des Eintrags, den Feldnamen und die einzelnen Namensbestandteile des Erfassers in versteckten Feldern festhalten. Durch das Betätigen des Ausführungsbuttons wird eine Methode des Objektes `Copy` aufgerufen. Diese Methode ermittelt die Eingaben aus dem Formular. Anschließend wird das Objekt für das freie Feld durch den Feldnamen und den Erfasser bestimmt. Den Attributen *meaning* und *multiple_value* dieses Objektes werden dann die entsprechenden Angaben aus dem Formular zugeordnet. Wichtig ist, daß das Attribut *new_field* auf „falsch“ gesetzt wird. Der aktuelle `tmp`-Eintrag wird mit dem Schlüsselwort und dem Erfasser ermittelt. Für diesen Eintrag werden dann die weiteren Überprüfungen vorgenommen.

2. Es wird ermittelt, ob potentiell gleiche Einträge in der Literaturdatenbank vorhanden sind. Dazu wird der Eintrag, der eingefügt werden soll, mit allen Einträgen der Datenbank vergleichen. Für den Vergleich wird die `compare`-Methode des Eintragsobjektes genutzt. Wurden potentiell gleiche Dokumente gefunden, dann werden diese in einem Formular ausgeben. Das Formular muß das Schlüsselwort und die einzelnen Bestandteile des Erfassernamens in versteckten Feldern enthalten. Der Nutzer kann über Radiobuttons angeben, ob ein bzw. welcher Eintrag gleich ist. Wenn ein Eintrag als gleich spezifiziert wurde, dann kann die ID dieses Eintrags über den Wert des Radiobuttons bestimmt werden. Wenn kein gleicher Eintrag angegeben wurde, dann wird das Einfügen des Eintrags im dritten Schritt fortgesetzt.

   Ansonsten müssen die Werte der gleichen Einträge verglichen werden. Der Eintrag, der in die Extension der Klasse engefügt werden soll, wird in `tmp` über das übergebene Schlüsselwort und den Erfasser bestimmt. Er wird im folgenden mit `new_entry` bezeichnet. Der Eintrag, der als gleich definiert

wurde, wird über die ID ermittelt und mit `old_entry` gekennzeichnet. Der Vergleich wird wie folgt durchgeführt:

- Wenn Erfasser nicht als Erfasser in `old_entry` enthalten ist, dann muß das Schlüsselwort überprüft werden (so wie im dritten Schritt beschrieben).
- Die einzelnen Attribute der beiden Objekte werden miteinander verglichen. Treten in den Attributen *fields* gleiche Felder auf, dann werden die Werte dieser Felder verglichen.
- Wenn keine Unterschiede auftreten, dann werden die neuen Felder (wenn vorhanden) in das mengenwertige Attribut *fields* eingefügt. Das Objekt wird aus der Extension `tmp` entfernt, wodurch es seine Persistenz verliert.
- Treten Unterschiede auf, dann wird ein Formular für alle unterschiedlichen Attribute bzw. Felder generiert und auf einer HTML-Seite ausgegeben. Der Erfasser kann dann mit Hilfe von Radiobuttons entscheiden, ob der neue oder der alte Wert verwendet wird. Die ID von `old_entry`, das Schlüsselwort von `new_entry` sowie die neuen Felder müssen als versteckte Felder im Formular enthalten sein. Durch die Betätigung des Ausführungsbuttons wird wiederum eine Methode des Objektes `Copy` aufgerufen.

  Die Methode liest die Angaben aus dem Formular aus. Der Eintrag `old_entry` wird durch die übergebene ID ermittelt und `new_entry` wird in `tmp` mit Hilfe des übergebenen Schlüsselwortes und des Erfassers bestimmt. Wenn das neue Schlüsselwort gewählt wurde und der Erfasser in `old_entry` vorkommt, dann muß die Eindeutigkeit des Schlüsselwortes überprüft werden. Ist es nicht eindeutig, dann wird das alte Schlüsselwort beibehalten. Anschließend werden dann den Attributen *language*, *other_authors*, *other_editors*, *crossref* die neuen Werte zugewiesen, wenn dies vom Erfasser so entschieden wurde. Danach wird für alle Felder in `new_entry` geprüft, ob der Erfasser im Formular für dieses Feld den neuen Wert gewählt hat. Ist dies der Fall, dann wird dieses Feld in `old_entry` geändert bzw. einfügt (wenn es noch nicht vorhanden ist).

  Das Eintrags-Objekt muß aus der Extension `tmp` entfernt werden, damit es von der Freispeicherverwaltung gelöscht werden kann. Bei allen Einträgen, die in `tmp` auf `new_entry` verweisen, wird der Querverweis auf das Objekt `old_entry` gesetzt. Wenn sich die Eintragtypen unterscheiden und vom Erfasser der neue Typ gewählt wurde, dann muß das Objekt `old_entry` reklassifiziert werden (siehe Abschnitt 6.10).

3. In diesen Schritt wird getestet, ob das Schlüsselwort des Eintrags innerhalb der Literaturdatenbank für den Erfasser eindeutig ist. Existiert bereits

ein Eintrag, dem der Erfasser das gleiche Schlüsselwort zugeordnet hat, dann wird ein neues Schlüsselwort benötigt. Es wird eine HTML-Seite mit einem Formular generiert. In diesem Formular kann ein neuer Schlüssel eingegeben werden. Außerdem muß das Formular das alte Schlüsselwort und die einzelnen Bestandteile des Erfassernamens in versteckten Feldern enthalten.

Die Methode, die nach der Eingabe des Schlüsselwortes aufgerufen wird, liest die Eingaben aus dem Formular. Es muß dann geprüft werden, ob das neue Schlüsselwort für den Erfasser eindeutig ist. Ist dies nicht der Fall, dann muß der Nutzer erneut ein anderes Schlüsselwort eingeben. Ansonsten wird der Eintrag, für den das neue Schlüsselwort eingegeben wurde, in `tmp` mit dem altem Schlüsselwort und dem Erfassernamen bestimmt. Anschließend erfolgt dann die Änderung der Schlüsselwortes.

4. Der Eintrag wird in den Namen, der dem Eintragtyp entspricht, und in `entries` eingefügt. Anschließend wird er aus `tmp` entfernt und die ID für den Eintrag bestimmt.

6.8.2.1 Bestimmung des Attributs ID für ein Eintrags-Objekt

Ein Eintrags-Objekt enthält das Attribut *ID*, damit Eintragsobjekte, die in Formularen enthalten sind, einfacher ermittelt werden können (siehe auch Abschnitt 3.3.3.1). Zur Bestimmung des *ID*-Attributs wird die Klasse *Identity* mit den zwei Attributen *ID* und *free_IDs* definiert. Das Attribut *ID* enthält eine Nummer, die einem neuen Eintrag zugeordnet werden kann. Im mengenwertigen Attribut *free_IDs* werden die IDs von gelöschten Einträgen festgehalten.

Das Attribut *ID* des Eintrags-Objektes wird folgendermaßen bestimmt:

- Wenn die Menge *free_IDs* nicht leer ist, dann wird die erste ID aus dieser Menge für den Eintrag genutzt. Anschließend muß diese ID aus *free_IDs* entfernt werden.
- Ist die Menge *free_IDs* leer, dann wird die Nummer in *ID* für den Eintrag genutzt. Anschließen muß die Nummer in *ID* um Eins erhöht werden.

6.9 Änderung eines Eintrags

Bei der Änderung eines Eintrags wird so verfahren, wie es in Abschnitt 5.4 beschrieben wurde. Beim Hinzufügen bzw. Löschen von Autoren, Herausgebern und Feldern wird ein neues Formular generiert. Ein Querverweis wird durch den Erfasser und das zugehörige Schlüsselwort angegeben.

Wenn der zu ändernde Eintrag einen Querverweis besitzt, dann ist zu prüfen, ob der Nutzer in dem Eintrag, auf den verwiesen wird, als Erfasser festgehalten ist. In diesem Fall kann der Nutzer und das zugehörige Schlüsselwort für die Angabe des Querverweises genutzt werden. Ansonsten wird für die Angabe der erste Erfasser und das zugeordnete Schlüsselwort verwendet. Der Querverweis kann geändert werden, indem ein neues Schlüsselwort und gegebenfalls ein anderer Erfasser eingegeben wird. Bei der Änderung des Eintrags muß dann in `entries` nach einem Eintrag gesucht werden, der den angegebenen Erfasser mit dem entsprechenden Schlüsselwort enthält.

Nach der Änderung eines Eintrags ist folgendes zu überprüfen:

- Enthält der Eintrag frei definierbare Felder, die zum ersten Mal genutzt werden?
- Existieren in der Datenbank Einträge, die potentiell gleich sind?
- Wurde ein Schlüsselwort angegeben, dann muß überprüft werden, ob es für den Erfasser eindeutig ist.

Die Überprüfungen erfolgen in einer ähnlichen Art und Weise wie beim Einfügen eines Eintrags in die Datenbank (siehe Abschnitt 6.8.2). Für den Eintrag, der geändert werden soll, wird in den Formularen die ID festgehalten. In den Methoden, die beim Betätigen des Ausführungsbuttons eines Formulars aufgerufen werden, kann der Eintrag dann durch die ID ermittelt werden.

6.10 Reklassifizierung

O_2 unterstützt eine reine Typhierarchie. Objekte können somit nicht in andere Klassen bewegt werden. Aus diesem Grund kann man die Reklassifizierung nicht so durchführen, wie es in Abschnitt 5.6 beschrieben wurde.

Der neue Eintragstyp wird in einem Formular angegeben. Die Reklassifizierung läuft dann in den folgenden Schritten ab:

- Es wird ein Objekt in der Klasse, die dem neuen Eintragstyp entspricht, erzeugt. Konnte für den Eintragstyp keine Klasse bestimmt werden, dann wird ein Objekt in der Klasse *Other_type* erzeugt. Der Eintragstyp muß dann dem Attribut *name* dieses Objektes zugeordnet werden.
- Alle Attribute des Objektes, das reklassifiziert werden soll, werden den entsprechenden Attributen des neuen Objektes zugewiesen.

- Alle Einträge, die auf das zu reklassifizierende Objekt verweisen, werden ermittelt, da den *crossref*-Attributen dieser Einträge das neue Objekt zuzuweisen ist.

- Das neue Objekt wird in die Extension der Klasse und in `entries` eingefügt. Das reklassifizierte Objekt muß aus der zugehörigen Extension und aus `entries` entfernt werden. Dadurch verliert das Objekt die Persistenz und wird von der automatischen Freispeicherverwaltung gelöscht. Der Wert des Attributs *ID* des reklassifizierten Objektes muß in das mengenwertige Attribut *free_IDs* des *Identity*-Objektes eingefügt werden.

6.11 Anfragen

Die OQL-Anfragen werden aus den Angaben in den Formularen für die einzelnen Anfragefunktionen zusammengesetzt. Die Suche nach einem Eintrag kann in `entries` erfolgen bzw. auf die Extensionen, die für die einzelnen Unterklassen von *entry* definiert wurden, eingeschränkt werden.

Die Sortierung des Anfrageergebnisses kann nach dem Eintragstyp, dem Titel, dem ersten Autoren bzw. Herausgeber und nach dem Jahr der Veröffentlichung erfolgen. Für die Sortierung werden die folgenden Methoden in der Klasse *entry* definiert:

- `type_string`: Diese Methode muß in den einzelnen Unterklassen redefiniert werden. Sie gibt den Namen der Klasse und somit den Eintragstyp zurück. In der Unterklasse *Other_type* liefert die Methode den Wert des Attributs *name*.

- `author`: Die Methode gibt den Namen des Autoren, der an erster Stelle angegeben wurde, bzw. eine leere Zeichenkette, wenn kein Autor vorhanden ist, zurück.

- `editor`: Diese Methode liefert den Namen des ersten Herausgebers bzw. eine leere Zeichenkette, wenn kein Herausgeber vorhanden ist.

Die Einführung dieser Methoden ist wichtig, da die Sortierung nur nach atomaren Werten erfolgen kann. Für den Titel und das Jahr wird die Methode `get_value` genutzt, die den Wert des Feldes bzw. eine leere Zeichenkette zurückliefert, wenn das Feld nicht vorhanden ist. Außerdem kann angegeben werden, ob die Sortierung aufsteigend oder absteigend erfolgt.

Weiterhin können die ermittelten Einträge nach dem Eintragstyp und dem Erscheinungsjahr gruppiert werden. Dazu wird die Methode `get_year` in der Klasse

entry definiert, die das Erscheinungsjahr bzw. eine leere Zeichenkette zurückliefert, wenn für den Eintrag kein Erscheinungsjahr angegeben wurde. Diese Methode muß eingeführt werden, da man bei der Gruppierung keine Methoden mit Parametern verwenden darf.

Wenn die Werte, nach denen gesucht wird, die Zeichen \ und " enthalten, dann muß diesen Zeichen ein \ vorangestellt werden.

Für das Ergebnis einer Anfrage wird eine HTML-Seite generiert. Für jeden Eintrag aus dem Anfrageergebnis ist auf dieser Seite ein Link enthalten. Der Text für den Link wird aus den wichtigsten Angaben des Eintrags gebildet. Jede HTML-Seite, auf der ein Anfrageergebnis dargestellt wird, enthält ein Formular, mit dem alle ermittelten Einträge exportiert werden können. In diesem Formular kann angegeben werden, in welches Format der Export erfolgen soll und ob die Kurz- oder die Langform einer Abkürzung beim Export zu nutzen ist. Außerdem enthält das Formular die IDs aller Einträge aus dem Anfrageergebnis.

6.12 Export

Die Einträge, die zu exportieren sind, werden über die verschiedenen Anfragemöglichkeiten ermittelt. Die HTML-Seite, die für das Anfragergebnis generiert wird, enthält ein Formular, mit dem der Export aller ermittelten Einträge veranlaßt werden kann. Dieses Formular enthält versteckte Felder, in denen die IDs aller Einträge aus dem Anfrageergebnis festgehalten sind.

Die Methode, die für den Export aufgerufen wird, ermittelt alle IDs aus dem Formular. Mit diesen IDs können dann die einzelnen Eintragsobjekte aus dem Anfrageergebnis bestimmt werden. Abhängig vom Format, das für den Export gewählt wurde, wird die Methode `export_bibtex`, `export_refdbms` oder `export_ncstrl` des Eintragsobjektes aufgerufen. Die Methode liefert dann den Eintrag im entsprechenden Format.

Die Methoden `export_refdbms` und `export_ncstrl` besitzen noch keine Funktionalität, nur die Methode `export_bibtex` ist voll implementiert.

Kapitel 7

Abschließende Bemerkungen

7.1 Zusammenfassung

In der vorliegenden Arbeit wurde ein Literaturdatenbanksystem entwickelt, das eine adäquate Speicherung von Einträgen im BibTeX-, *refdbms*- oder *NCSTRL*-Format erlaubt. Die Umsetzung des Systems erfolgte mit dem OODBS O_2 und für die WWW-Repräsentation wurde das O_2Web genutzt.

Zunächst wurden die einzelnen Literaturdatenbankformate untersucht. Insbesondere erfolgte eine Untersuchung der Formate in Bezug auf die vorhandenen Eintragstypen und Felder. Ein Vergleich der einzelnen Formate zeigte, daß sich die Eintragstypen und Felder in den verschiedenen Formaten unterscheiden. Diese Konflikte zwischen den Feldern wurden durch die Zuordnung von Feldwerten zu anderen Feldern bzw. durch das Herausfiltern bestimmter Angaben aus Feldwerten gelöst. Die herausgefilterten Angaben werden dann den entsprechenden Feldern zugewiesen.

Da das zentrale Datenmodell ein objektorientiertes ist, wurde ein objektorientiertes Datenbankmodell vorgestellt. Danach erfolgte dann die Entwicklung des Schemas für die Literaturdatenbank. Dabei wurden verschiedene Probleme diskutiert, die bei der Modellierung auftraten.

Auf der Grundlage des Datenbankschemas erfolgte dann die Entwicklung des Datenbanksystem. Besonders detailliert wurden die Import- und die Exportschnittstelle beschrieben. Die Import-Schnittstellen der verschiedenen Formate müssen dafür sorgen, daß die Informationen zu einem Eintrag auf die entsprechenden Felder in der Datenbank abgebildet werden. Diese Abbildungen basieren auf den Lösungen für die Konflikte zwischen den Feldern. Die einzelnen Export-Schnittstellen müssen die Orginalinformationen aus den Angaben, die für einen Eintrag abgespeichert wurden, wiederherstellen.

Anschließend erfolgte die Umsetzung des Datenbanksystems in O_2. Dabei wurden die Architektur des Datenbanksystems sowie O_2 und die weiteren Komponenten von O_2, die für die Umsetzung verwendet werden, vorgestellt. Die Implementie-

rung der Import- und Export-Schnittstelle erfolgte nur für das BibTeX-Format.

Der Import von Literaturdateien erfolgt über das WWW. Dies setzt aber voraus, daß diese Dateien für den WWW-Server zugänglich sind. Aus diesem Grund wurden Möglichkeiten für die Übertragung von Dateien an den WWW-Server betrachtet.

7.2 Ausblick

Bei der Weiterentwicklung des Systems sollten zunächst die Import- und Exportschnittstellen für das *refdbms*- und das *NCSTRL*-Format implementiert werden. Beim Import von BibTeX-Einträgen wäre es von Vorteil, wenn die Präambel-Befehle herausgefiltert werden.

O_2 Java-Binding Es könnte untersucht werden, ob sich einige Funktionen des Systems nicht besser mit dem O_2 Java-Binding umsetzen lassen, da bei der Generierung von HTML-Seiten für die Funktionen folgende Probleme auftreten:

- Eine Transaktion muß immer an Ende einer Methode, die eine entsprechende HTML-Seite generiert, abgeschlossen werden.
- Variablen, die man für die Auswertung eines generierten Formulars benötigt, müssen in versteckten Feldern des Formulars übergeben werden. Bei atomaren Werten ist dies kein Problem, aber bei Objekten: In diesem Fall muß der Schlüssel des Objektes übergeben werden. In der Methode, die vom Formular aufgerufen wird, muß dieses Objekt dann erst wieder mit Hilfe des übergebenen Schlüssels bestimmt werden.

Typabhängige Ausgabe Die Ausgabe der Einträge, die als Ergebnis eines Anfrageergebnisses ermittelt wurden, könnte typabhängig erfolgen. Dabei könnte man sich an der Form orientieren, in der die Einträge einer BibTeX-Datei im Literaturverzeichnis eines LaTeX-Dokumentes ausgeben werden.

Typkennung für Felder Für die Felder sollte eine Typkennung eingeführt werden, die man für die Eingabemasken und die Ausgabe heranziehen kann. Bei den Typen könnte man z.B. zwischen `Person`, `Text`, `Langer Text` unterscheiden, wobei die beiden ersteren in einzelne Eingabezeilen, letzterer aber in ein mehrzeiliges Feld umgesetzt werden könnte. Wenn ein freies Feld definiert wird, dann kann mit einer Drop-down-Liste der entsprechende Typ ausgewählt werden.

Weitere Änderungsmöglichkeiten Es sollte die Änderung von Journal-, Verlags- und Personenobjekten ermöglicht werden. Außerdem könnte man eine Funktion für die Änderung der Bedeutung eines frei definierbaren Feldes bereitstellen.

Import Beim Import sollten die folgenden Tests verfeinert werden:

- Bei der Suche nach potentiell gleichen Dokumenten sollten unterschiedliche Schreibweisen bei den Sonderzeichen (z.B. kann ein ö durch `{\"o}` oder `{\"{o}}` angegeben werden) keine Rolle spielen. Im Moment ist dies noch der Fall, da die Werte der entsprechenden Felder auf Gleichheit getestet werden.
- Die Suche nach gleichen Autoren könnte verbessert werden, denn im Moment stellen `D. E. Knuth` und `Donald E. Knuth` unterschiedliche Personen-Objekte dar, obwohl es sich bei beiden Angaben um den gleichen Autoren handelt. Wurden vom System potentiell gleiche Personen ermittelt, dann muß der Nutzer entscheiden, ob diese Personen wirklich gleich sind.

Verbesserung der Suchmöglichkeiten Die Suchmöglichkeiten sollten verbessert werden. Wenn nach Einträgen gesucht wird, die bestimmte Informationen enthalten, dann sollten Sonderzeichen wie Umlaute und ß in den Formularfelder angegeben werden können. Wird z.B. bei der Suche nach einem Autoren im Formularfeld der Name `Müller` eingegeben, dann sollten auch die Einträge ausgegeben werden, die eine Autorenangabe `M{\"u}ller` oder `M{\{"u}}ller` enthalten.

Ausgabe von Feldwerten Die Angaben in den *ftp* und *http*-Feldern sollten als Links ausgeben werden. Außerdem könnte man bei diesen Feldern die Gültigkeit der Angaben prüfen. Weiterhin sollten LaTeX-Sonderzeichen wie Umlaute und ß bei der Ausgabe in die entsprechenden HTML-Sonderzeichen umgewandelt werden.

Automatischer Export Man könnte für das System eine Funktion einführen, mit der alle Literatureinträge, auf die man sich in einem LaTeX-Dokument bezieht, automatisch exportiert werden. Dies wäre wie folgt möglich:

- Bei der Übersetzung eines LaTeX-Dokumentes wird eine `aux`-Datei erzeugt. Für jeden `\cite`-Befehl im Dokument wird in der `aux`-Datei die Zeile

  `\citation{`*Schlüsselwort*`}`

  generiert. Das *Schlüsselwort* wird dem `\cite`-Befehl entnommen.

- Diese `aux`-Datei muß dann an den WWW-Server gesendet werden.
- Anschließend müssen die `\citation`-Befehle aus der Datei herausgefiltert werden.
- Jeder Eintrag, bei dem das dem Nutzer zugeordnete Schlüsselwort mit einem der in den `\citation`-Befehlen enthaltenen Schlüsselwörter übereinstimmt, wird dann exportiert.

Ein Problem tritt dann auf, wenn das LaTeX-Dokument `\include`-Befehle enthält: Für die Dateien, die mit diesem Befehl in das Dokument eingefügt werden, erzeugt LaTeX bei der Übersetzung eigene `aux`-Dateien, in die dann die jeweiligen `\citation`-Befehle geschrieben werden.

Gruppierung von Anfrageergebnissen Für OQL-Anfragen wird in O_2C eine Systemfunktion genutzt. Bei der Gruppierung eines Anfrageergebnisses nach mehreren Kriterien ergeben sich Probleme, da der Ergebnistyp statisch vorgegeben werden muß. Dieses Problem kann gelöst werden, indem die Anfrage so formuliert wird, daß man Objekte und keine Werte erhält. Dazu muß statt des `Group by`-Operators von OQL einfach eine Anfrage der Form

```
select Gruppe(Gruppierungswert:  g, Elemente:
     select distinct e
     from Entries e
     where e.<X>=g
     order by e.<Y>)
from g in
     (select distinct e.<X>
     from Entries e)
```

verwendet werden. `<X>` ist das Attribut, nach dem gruppiert werden soll, und nach dem Attribut `<Y>` wird sortiert. Statt der direkten Attributzugriffe und Vergleiche sollte man Methoden verwenden, z.B. könnte statt `e.<X>=g` die Methode `e.get_field('X').compare_with(g)` aufgerufen werden. Die Definition der `compare`-Methode muß dann für alle Klassen von Feldtypen erfolgen. Da `get_field('X')` mehrere freie Felder liefern kann, muß noch eine Aggregation vorgenommen werden.

Der Typ der Klasse `Gruppe` ist

```
tuple(Gruppierungswert:  string, Elemente:  list(Entry)).
```

Bei Mehrfachgruppierungen ist der Typ von `Elemente` entsprechend anzupassen (`list(Gruppe)`). Für die Klasse muß dann die Methode `html_report`-Methode definiert werden, in der die Ausgabe des Anfrageergebnisses implementiert wird.

Literaturverzeichnis

[BCG97] Baldonado, M.; Chang, C.-C. K.; Gravano, L.: Metadata for Digital Libraries: Architecture and Design Rationale. Technischer Bericht SIDL-WP-1997-0055, Stanford University, 1997.
`http://www-diglib.stanford.edu/cgi-bin/WP/get/-SIDL-WP-1997-0055`.

[BCGP96] Baldonado, M.; Chang, C.-C. K.; Gravano, L.; Paepcke, A.: The Stanford Digital Library Metadata Architecture. Technischer Bericht SIDL-WP-1996-0051, Stanford University, Oktober 1996.
`http://www-diglib.stanford.edu/cgi-bin/WP/get/-SIDL-WP-1996-0051`.

[Bee93] Beebe, N. H. F.: Bibliography prettyprinting and syntax checking. *TUGBoat*, Band 14, Nr. 4, S. 395–422, 1993.

[Con97] Conrad, S.: *Föderierte Datenbanksysteme: Konzepte der Datenintegration*. Springer, Berlin, 1997.

[CORa] *Dublin Core Metadata Element Set: Reference Description*.
`http://purl.org/metadata/dublin_core_elements`.

[CORb] *Metadaten: Inhaltliche Beschreibung des Dublin Core Element Set*.
`http://www.swbv.uni-konstanz.de/wwwroot/metadata/-kv_dc004.html`.

[GCGMP96] Gravano, L.; Chang, C.-C. K.; García-Molina, H.; Paepcke, A.: STARTS: Stanford Proposal for Internet Meta-Searching. Technischer Bericht SIDL-WP-1996-0043, Stanford University, August 1996.
`http://www-diglib.stanford.edu/cgi-bin/WP/get/-SIDL-WP-1996-0043`.

[Gep97] Geppert, A.: *Objektorientierte Datenbanksysteme: ein Praktikum*. Verlag für digitale Technologie, Heidelberg, 1997.

[GMS94] Gossens, M.; Mittelbach, F.; Samarin, A.: *Der LATEX-Begleiter*. Addison-Wesley, Bonn; Paris, 1994.

[Her92] Herold, H.: *lex und yacc: Lexikalische und syntaktische Analyse.* UNIX und seine Werkzeuge. Addison-Wesley, Bonn; München, Paris, 1992.

[Heu97] Heuer, A.: *Objektorientierte Datenbanken - Konzepte, Modelle, Standards und System.* Addison-Wesley-Longmann, Bonn, 2. Auflage, 1997.

[HS95] Heuer, A.; Saake, G.: *Datenbanken - Konzepte und Sprachen.* International Thomson Publishing, Bonn, 1. Auflage, 1995.

[Kop96] Kopka, H.: *LATEX: Einführung Band1.* Addison-Wesley, Bonn, 1996.

[Les78] Lesk, M. E.: Some applications of inverted indexes on the UNIX system. Computing Science technical report 69, Bell Laboratories, Murray Hill, Juni 1978.

[LMB92] Levine, J. R.; Mason, T.; Brown, D.: *lex & yacc.* UNIX Programming Tools. O'Reilly & Associates, Inc., Sebastopol, 1992.

[LV96] Lausen, G.; Vossen, G.: *Objekt-orientierte Datenbanken: Modelle und Sprachen.* Oldenbourg, München, 1996.

[MK96] Musciano, C.; Kennedy, B.: *HTML-The Definitive Guide.* O'Reilly & Associates, Inc., Bonn, Paris, 1996.

[NCSa] *Description of bibs.refer. NCSTRL*-Dokumentation `http://www.ncstrl.org/Dienst/htdocs/Info/bibformat.html`.

[NCSb] *Beschreibung des Eingabeformats. NCSTRL*-Dokumentation `http://www.ink.tu-harburg.de/literatur/bibformat.html`.

[$O_2$98a] O_2 Technology: *O_2C Beginner's Guide*, Februar 1998. Release 5.0.

[$O_2$98b] O_2 Technology: *O_2Web User Manual*, Februar 1998. Release 5.0.

[$O_2$98c] O_2 Technology: *ODMG C++ Binding Guide*, Februar 1998. Release 5.0.

[$O_2$98d] O_2 Technology: *ODMG OQL User Manual*, Februar 1998. Release 5.0.

[Pat88] Patashnik, O.: *BIBTEXing*, Februar 1988. BibTEX-Dokumentation.

[RFCa] RFC-2045 multipurpose internet mail extensions (mime) part one: Format of internet message bodies. `http://www.brattberg.se/activex/essmtp/rfc/rfc2045.htm`.

[RFCb] RFC-822 Date and Time Specification: Section 5. `http://andrew2.andrew.cmu.edu/rfc/rfc822.html#sec-5`.

[RG94] Richard Golding, J. W.: *The refdbms bibliography database user guide and reference manual*, 1994. *refdbms*-Dokumentation.

[Sch97] Schader, M.: *Objektorientierte Datenbanken: Die C++-Anbindung des ODMG-Standards*. Springer, Berlin, 1997.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

Beispiele für Literatureinträge

Im folgenden sind Beispieleinträge für die Literaturdatenbanken BibTeX, *refdbms* und *NCSTRL* angegeben.

A.1 BibTeX

```
@Book{BanDelKan92,
    author        = "Banchilon, F. and Delobel, C. and
                     Kanellakis, P.",
    title         = "Building an Object-Oriented Database
                     System -- The Story Of O2",
    publisher     = {Morgan Kaufmann Publishers},
    year          = 1996,
    OPTisbn       = "1-55860-169-4",
    annote        = "OODBS",
}
```

A.2 *refdbms*

```
%z   Article
%K   Lamport78
%A   Leslie Lamport
%T   Time, clocks and the ordering of events in a distributed system
%J   CACM.
%V   21
%N   7
%D   1978
%P   558 565
%x   The concept of one event happening before another in a distributed
```

```
%x  system is examined, and is shown to define a partial ordering of the
%x  events.  A distributed algorthm is given for synchronizing a system
%x  of logical clocks which can be used to totally order the events.  The
%x  use of the total ordering is illustrated with the method for solving
%x  synchronization problems.  The algorithm is then spezialized for
%x  synchronizing physical clocks, and a bound is derived on how far out
%x  of synchrony the clock can become.
%k  causal consistency, asynchrony, happens before
%k  clock synchronization
                        *** Leerzeile ***
```

A.3 *NCSTRL*

```
%A  Son, Sang H.
%A  Agarwal, Nipun
%T  Synchronization of Temporal Constructs in Distributed Multimedia Systems
%R  CS-93-57
%D  October 28, 1993
%Z  Tue, 29 Aug 1995 20:09:55 GMT
%K  Multimedia, Synchronization, ATM, Temporal Constructs
%Y  H.5.1 Multimedia Information Systems
    B.4.1 Data Communications Devices
%X  With the inception of technology in communication networks such as
    ATM it will be possible to run multimedia applications on future
    integrated networks.  Synchronization of the related media data is one
    of the key characteristics of a multimedia system.  In this paper we
    present a scheme for synchronization of multimedia data across a
    network where the accuracy of detecting asynchronization and
    predicting the future asynchrony is variable and can be tailored to
    the intended application.  The protocol has been designed keeping in
    mind characteristics of ATM networks such as the absence of global
    synchronized clocks and utilizing features as the QOS promised by
    them.  The multimedia data when sent across the network may also be
    stored at an intermediate node and later retrieved for dis- play.  We
    extend the scheme and present a mechanism wherein synchronization of
    all the possible temporal constructs is supported and not restricted
    to the ïn-parallel" construct which is only one of the thirteen
    possible temporal relations.
%U  ftp://ftp.cs.virginia.edu/pub/techreports/CS-93-57.ps.Z
%I  Department of Computer Science, University of Virginia
                        *** Leerzeile ***
```

Anhang B

Erstellung der Datenbank und zugehöriger Programme

B.1 Laden der Datenbank

Der Name des Datenbankschemas ist `bibliography_schema` und die Datenbasis hat den Namen `bibliography_base`. Wenn diese Namen geändert werden sollen, dann sind Änderungen beim BibTeX-Parser vorzunehmen (siehe Anhang B.4).

Die Datenbank ist in den folgenden Schritten zu erstellen:

1. Das in der Datei `schema_def` enthaltene Schema laden.
2. Die Methoden-Implementierungen laden. Diese sind in den Dateien

```
abbrev_code
author_query_code
change_code
copy_code
entry_code
entry_types_code
ffield_list_code
fname_code
fvalue_code
htmltool_code
identity_code
key_query_code
list_code
menu_code
o2webinteractor_code
person_code
query_code
```

```
reclassify_code
start_code
submitter_query_code
tmp_code
topic_query_code
urltool_code
user_query_code
```

enthalten.

3. Die Datei `setup_schema` laden, um Initialisierungen für die Datenbank vorzunehmen.

B.2 Benötigte Dateien auf dem WWW-Server

Die folgenden Dateien müssen in ein Verzeichnis kopiert werden, das dem WWW-Server zugänglich ist:

`Parser.class`	Java-Applet für Ausgaben des BibTeX-Parsers
`back.gif`	
`exec.cgi`	CGI-Skript, das vom Applet aufgerufen wird
`exec_nojava.cgi`	CGI-Skript, das aufgerufen wird, wenn das Applet nicht gestartet werden konnte
`green_ball.gif`	
`index.gif`	
`next.gif`	
`nojava.gif`	
`orange_ball.gif`	

Der Quelltext für das Java-Applet ist in der Datei `Parser.java` enthalten.

B.3 Konfiguration der Datenbank

Die folgenden Verzeichnisse können geändert werden:

- `parserdir`: Dieses Verzeichnis enthält den BibTeX-Parser.
- `parserappletdir`: In diesem Verzeichnis befinden die Dateien aus dem Abschnitt B.2.
- `wwwservertmpdir`: In diesem Verzeichnis werden die Import-Dateien temporär gespeichert.

Der WWW-Server muß auf die angegebenen Verzeichnisse zugreifen können. Die Verzeichnisse sind Attribute eines Objektes der Klasse `Configuration`, das über den Namen `configuration` bestimmt werden kann. Sie können auch in der Datei `setup_schema` geändert werden, wenn die Datenbank neu eingespielt wird.

B.4 Kompilierung des BIBTEX-Parsers

Für die Kompilierung des BIBTEX-Parsers werden die folgenden Dateien benötigt:

`bibtex_o2.cc`	Datenbank-Funktionen
`bibtex.cf`	Konfigurations-Datei
`parser/`	
`Makefile`	
`bibtex.h`	
`bibtex.l`	`lex`-Datei
`bibtex.y`	`yacc`-Datei
`bibtex_parse.cc`	Funktionen zum Prüfen und Speichern von Einträgen
`bibtex_parse.h`	
`bibtex_parse_def.h`	
`message.h`	
`site.cf`	Definitionen für die Compiler-Umgebung

Der Parser ist in den folgenden Schritten zu erstellen:

- Im Verzeichnis `parser` muß der Befehl `make` ausgeführt werden.
- Danach wird das Kommando

  ```
  o2makegen -deffile site.cf bibtex.cf
  ```

  ausgeführt, um aus der Konfigurations-Datei ein Makefile zu generieren.
- Anschließend wird mit dem Befehl `make` der BIBTEX-Parser erzeugt. Das ausführbare Programm hat den Namen `import_bibtex`.

Der BIBTEX-Parser muß in ein Verzeichnis kopiert werden, auf das der WWW-Server zugreifen kann.

Wichtig: Der WWW-Server muß ein O_2-Client sein, damit der BIBTEX-Parser ausgeführt werden kann.

Wenn der Name des Datenbankschemas bzw. der Datenbasis geändert werden soll, dann müssen folgende Anpassungen vorgenommen werden:

- In der Datei `parser/bibtex.h` muß die Zeile

  `#define DATABASENAME "`*Name*`"`

  geändert werden, wobei *Name* der neue Name der Datenbasis ist.

- Der neue Name des Datenbankschemas ist in der Datei `bibtex.cf` in der Zeile

  `O2Schema=`*Name*

  anzugeben.

B.5 O_2Web-Server-Einstellungen

Die Umgebungsvariable `O2WEBCOMMIT_FREQUENCY` muß den Wert 1 haben, wenn der O2Web-Server gestartet wird.

Anhang C

Nutzerhandbuch

C.1 Einstieg in die Datenbank

Der Einstieg in die Datenbank erfolgt mit der URL

```
http://wwwdb.informatik.uni-rostock.de/cgi-bin/o2web_gateway/
demo/bibliograpy_base?start
```

Auf der Startseite ist der Name des Benutzers anzugeben. Der Nutzername muß in der Reihenfolge *Vorname*, *von-Teil*, *Nachname*, *Jr-Teil* spezifiziert werden. Zum *von-Teil* gehören alle kleingeschriebenen Wörter der Namensangabe. Der *Jr-Teil* umfaßt Angaben wie `Jr.` oder `Sr.`.

C.2 Funktionen der Datenbank

Über das folgende Menü werden die einzelnen Funktionen des Datenbanksystems aufgerufen.

Das Datenbanksystem umfaßt

- den Import von BibTeX-Dateien,

- das Auflisten vorhandener Abkürzungen und deren Änderung,
- das Hinzufügen von Abkürzungen,
- verschiedene Anfragefunktionen, mit denen die Suche nach
  - den Einträgen bestimmter Autoren,
  - Einträgen, denen von einem angegebenen Erfasser bestimmte Schlüsselwörter zugeordnet wurden,
  - allen Einträgen eines Erfassers,
  - Einträgen, die bestimmten Themen zugeordnet werden können, sowie
  - Einträgen, die bestimmte Feldangaben enthalten

  möglich ist, und
- den Export von Einträgen, die über Anfragen ermittelt wurden.

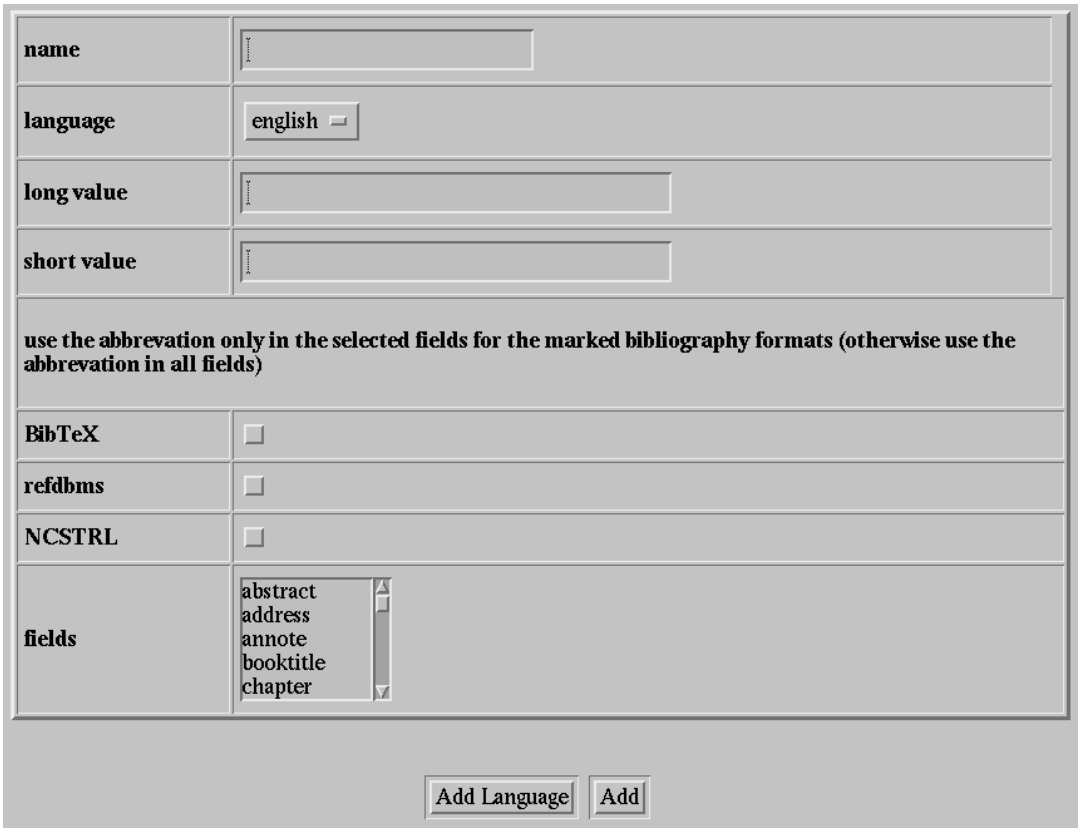
Die einzelnen Funktionen werden im Menü über den entsprechenden Link aufgerufen.

C.3 Abkürzungen

Die Abkürzungen, die in der Datenbank enthalten sind, werden beim Import von Einträgen verwendet. Feldwerte, die mit dem Namen einer in der Datenbank enthaltenen Abkürzung übereinstimmen, werden durch die entsprechende Abkürzung ersetzt. Der Wert einer Abkürzung ist von der Sprache, die für den Eintrag spezifiziert wurde, abhängig, und er kann in einer Kurz- und Langform angegeben werden. Bei der Ausgabe von Einträgen erfolgt die Darstellung der Abkürzungen immer in der Langform. Beim Export von Einträgen kann der Benutzer entscheiden, ob die Abkürzungen durch die jeweilige Kurz- oder Langform ersetzt werden.

C.3.1 Einfügen von Abkürzungen

Die Liste der vorhandenen Abkürzungen ist erweiterbar. Dazu wird im Menü die Funktion `Add Abbrevations` aufgerufen. Es erscheint dann das folgende Eingabeformular:



Als erstes ist der Name der Abkürzung einzugeben. Die Sprache (englisch oder deutsch), in der Kurz- und Langform der Abkürzung angegeben sind, kann über eine Auswahlliste spezifiziert werden. Sind die Checkbuttons in den Feldern `bibtex`, `refdbms` und `NCSTRL` aktiviert, dann werden die Angaben im Feld `fields` beim Import von Einträgen im jeweiligen Format ausgewertet. In der Liste `fields` können Felder ausgewählt werden, in denen die Abkürzungen beim Import zu verwenden sind. Bei deaktivierten Checkbuttons kann die Abkürzung in allen Felder eines Eintrags genutzt werden.

Über den Button `Add Language` können weitere Werte für die Abkürzung in einer anderen Sprache angegeben werden. Die Abkürzung wird über den Button `Add` in die Datenbank eingefügt.

C.3.2 Ändern vorhandener Abkürzungen

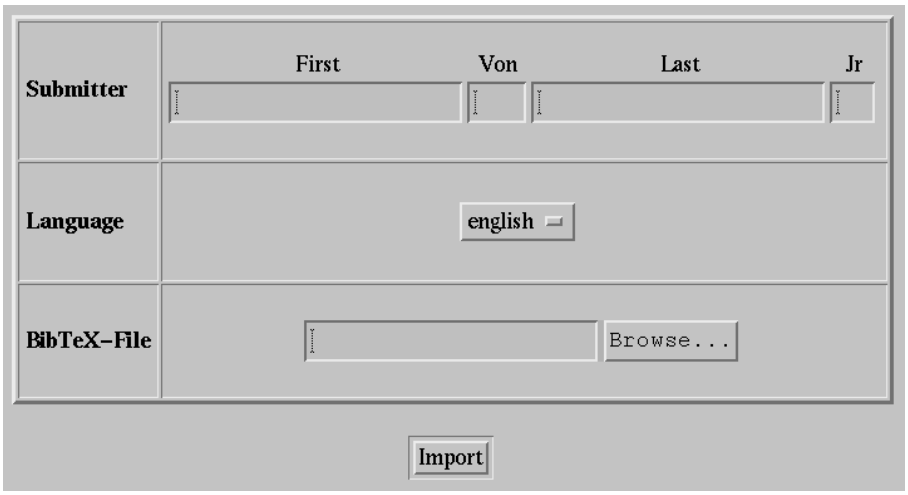
Wenn eine vorhandene Abkürzung geändert werden soll, dann ist zunächst die Funktion `Show Abbrevations` aufzurufen. Mit dieser Funktion erfolgt die Auflistung aller vorhandenen Abkürzungen in alphabetischer Reihenfolge. Anschließend ist der Name der Abkürzung, die geändert werden soll, anzuklicken. Es erscheint ein Formular, das die aktuellen Werte der Abkürzung enthält. Diese

können dann geändert werden. Die Bedeutung der einzelnen Formularfelder wurde bereits beim Einfügen von Abkürzungen im vorherigen Abschnitt beschrieben.

Über den Button `Add Language` kann man Werte in einer weiteren Sprache angeben (wenn noch keine deutsch- und englischsprachigen Werte spezifiziert wurden). Bei der Betätigung des `Change`-Buttons werden die Änderungen übernommen.

C.4 Import von BibTeX-Dateien

Über die Menü-Funktion `Import BibTeX-File` kann eine BibTeX-Datei importiert werden. Es erscheint dann das folgende Formular:



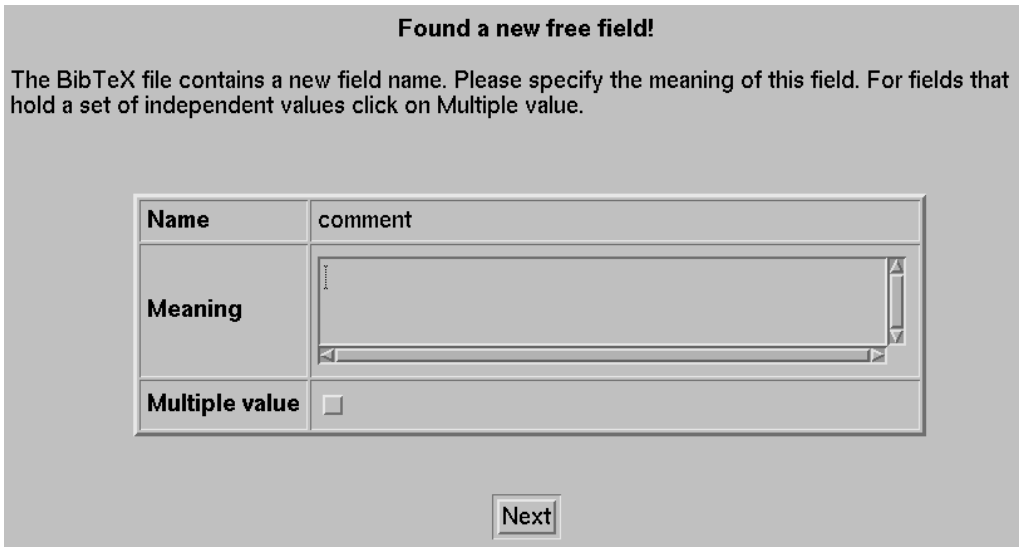
Im ersten Formularfeld ist der Erfasser anzugeben. Die einzelnen Bestandteile des Erfassernamens wurden bereits in Abschnitt C.1 beschrieben. Über eine Auswahlliste erfolgt die Auswahl der Sprache (englisch oder deutsch), die dann allen Einträgen aus der Datei zugeordnet wird. Im Feld `BibTeX-File` ist der Name der BibTeX-Datei, die importiert werden soll, anzugeben. Bei Betätigung des Buttons `Browse...` wird eine Dialogbox für die Auswahl einer Datei im lokalen Dateisystem geöffnet.

Wichtig: Die angegebene Sprache wird allen Einträgen, die in der Datei enthalten sind, zugeordnet. Aus diesem Grund sollte darauf geachtet werden, daß die Datei nur Einträge der gewählten Sprache enthält.

Wenn der Browser javafähig ist bzw. wenn diese Eigenschaft nicht deaktiviert wurde, dann erfolgt die Ausgabe der Meldungen, die beim Einlesen der BibTeX-Datei generiert werden, in einem Java-Applet. In diesem Applet wird außerdem

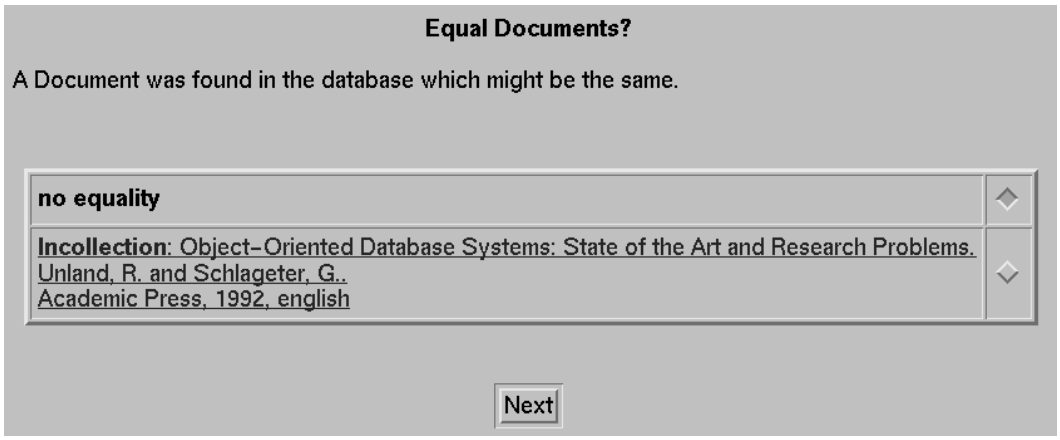
dargestellt, wieviele Einträge zu einem bestimmten Zeitpunkt eingelesen wurden. Ist der Browser nicht javafähig, dann werden die Meldungen auf einer HTML-Seite ausgegeben.

Nach der Beendigung des Einlesevorgangs ist der `Next`-Button zu betätigen. Die Einträge werden dann in die Datenbank eingefügt. Enthält ein Eintrag ein frei definierbares Feld, das vorher noch nicht genutzt wurde, erscheint das folgende Formular:



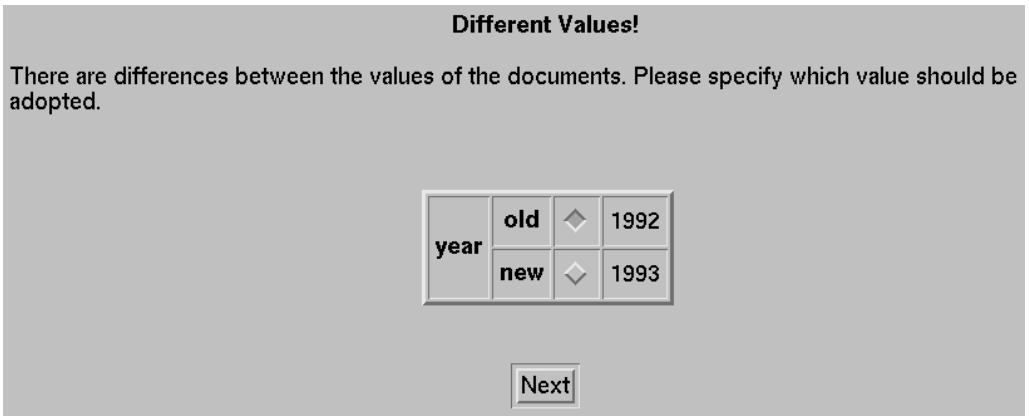
In diesem Formular muß die Bedeutung des Feldes angegeben werden. Besteht das Feld aus einer Liste von Angaben, die unabhängig voneinander sind, dann ist der Checkbutton `Multiple value` zu aktivieren.

Wenn für einen Eintrag potentiell gleiche Dokumente in der Datenbank gefunden wurden, wird das folgende Formular ausgegeben:



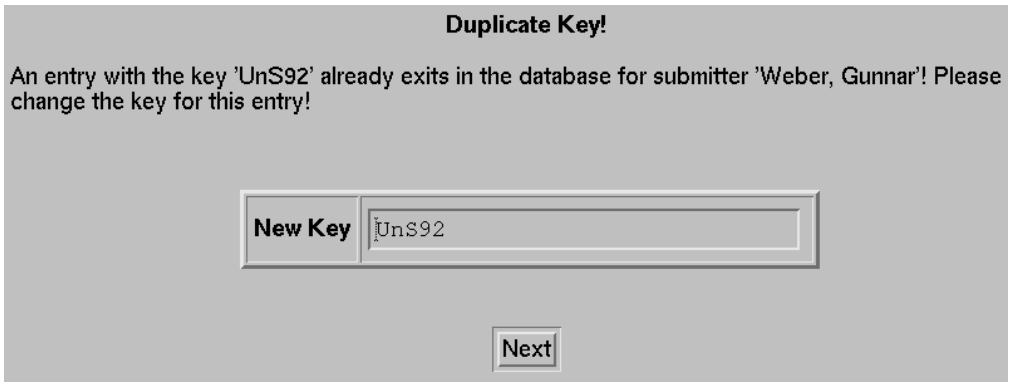
Das Formular enthält alle potentiell gleichen Einträge. Beim Klicken auf einen dieser Einträge wird ein Fenster geöffnet, in dem alle Angaben des entsprechenden Eintrags dargestellt sind. Wenn ein Eintrag wirklich gleich ist, dann muß der Radiobutton, der diesem Eintrag zugeordnet ist, aktiviert werden. Ist keiner der Einträge gleich, dann ist der Radiobutton im Feld `no equality` zu aktivieren.

Wurde ein Eintrag als gleich definiert, dann können Unterscheide zwischen den Angaben des Eintrags, der eingefügt werden soll, und dem bereits vorhandenen Eintrag auftreten. Alle Unterschiede sind in einem Formular dargestellt.



In dem Formular sind die einzelnen Eigenschaften aufgelistet, bei denen Unterschiede auftreten. Für jede Eigenschaft ist der alte und der neue Wert angegeben. Die Auswahl, ob der Eigenschaft der alte oder der neue Wert zugewiesen werden soll, erfolgt über die zugeordneten Radiobuttons.

Wenn in der Datenbank ein Eintrag gefunden wurde, dem der Erfasser das gleiche Schlüsselwort zugeordnet hat wie dem Eintrag, der eingefügt werden soll, dann ist ein neues Schlüsselwort einzugeben.

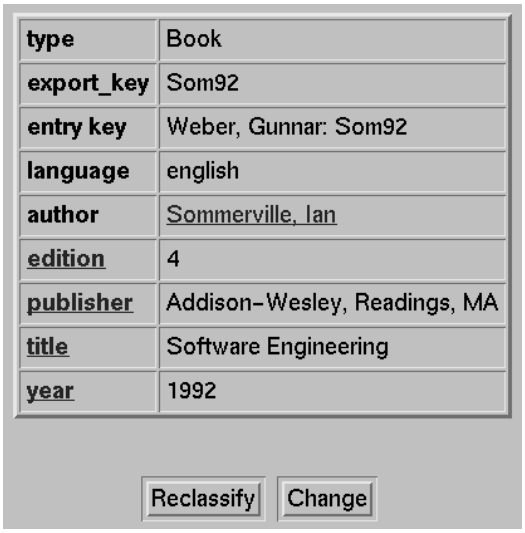


C.5 Anfragen

Das Datenbanksystem stellt verschiedene Anfragemöglichkeiten bereit. Bei allen Anfragefunktionen können die folgenden Angaben spezifiziert werden:

In der Liste `Entry Type` erfolgt die Auswahl, ob alle Einträge der Datenbank bei der Suche berücksichtigt werden sollen oder ob die Suche auf einen bestimmten Eintragstyp beschränkt werden soll. Im Feld `Sort by...` kann man verschiedene Felder auswählen, nach denen das Anfrageergebnis sortiert wird. Die Festlegung der Sortierreihenfolge (aufsteigend oder absteigend) erfolgt dann im Feld `Sort-Order`. Ein Kriterium, nach dem das Ergebnis gruppiert werden soll, kann man in der Liste `Group by ...` auswählen. Das Anfrageergebnis kann sich abhängig von der Anzahl der ermittelten Einträge über mehrere HTML-Seiten erstrecken. Die Anzahl der Einträge, die auf einer Seite enthalten sind, wird über das Feld `Documents per Page` beeinflußt. Der Standardwert für dieses Feld ist `20`.

Die Einträge werden im Anfrageergebnis in einer Kurzform mit den wichtigsten Angaben eines Eintrags dargestellt. Diese Kurzformen sind als Links gestaltet. Wenn man so einen Link aktiviert, dann wird der gesamte Eintrag in der folgenden Form ausgegeben:

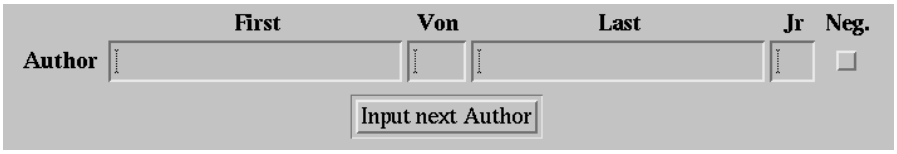


Im Feld `type` ist der Typ des Eintrags enthalten. Das Schlüsselwort, das dem Eintrag beim Export in das BibTeX- bzw. *refdbms*-Format zugeordnet wird, ist im Feld `export_key` angegeben. In `key` sind die Schlüsselwörter aller Erfasser enthalten. Außerdem werden die einzelnen Felder, die der Eintrag enthält, und die zugehörigen Werte ausgegeben.

Über den `Reclassify`-Button kann man den Eintrag reklassifizieren (siehe Abschnitt C.8). Soll der Eintrag geändert werden, dann ist der Button `Change` zu betätigen (siehe Abschnitt C.7).

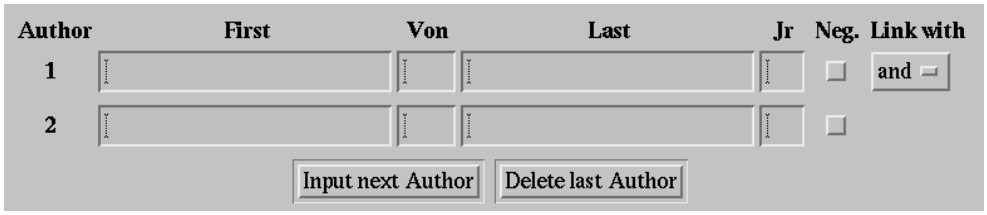
C.5.1 Suchen nach Autoren

Diese Funktion dient der Suche nach Einträgen, deren Autorenlisten bestimmte Autoren enthalten bzw. nicht enthalten. Sie wird im Menü über den Link `Author` aufgerufen. Der Autor, nach dem gesucht werden soll, ist im folgenden Formular anzugeben:



Die Angabe eines Autorennamens erfolgt in der Reihenfolge *Vorname*, *von-Teil*, *Nachname*, *Jr-Teil*. In den einzelnen Feldern können Wildcards (`*`) verwendet werden. Ist der Checkbutton `Neg` aktiviert, dann darf der angegebene Autor nicht in der Autorenliste eines Eintrags vorkommen.

Über den Button `Input next Author` können weitere Autoren hinzugefügt werden. Es erscheint dann das folgende Formular:

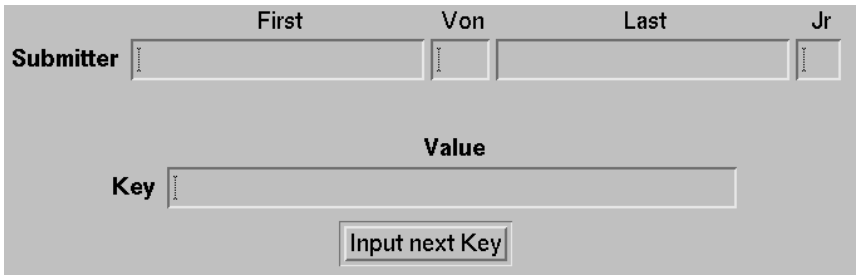


Der nächste Autor wird in der gleichen Form wie der erste Autor angegeben. Im Feld `Link with` ist zu spezifizieren, wie die einzelnen Autorenangaben miteinander verknüpft werden sollen. Dieses Feld enthält eine Liste, aus der die Verknüpfung `and` oder `or` gewählt werden kann. Über den Button `Delete last Author` kann man den letzten Autoren wieder löschen.

Die Suche wird durch die Betätigung des Buttons `Search` ausgelöst.

C.5.2 Suchen nach Eintragsschlüsseln für einen Erfasser

Über die Menü-Funktion `Entry key` können Einträge ermittelt werden, denen von einem angegebenen Erfasser bestimmte Schlüsselwörter zugeordnet wurden. Dazu wird das folgende Formular genutzt:

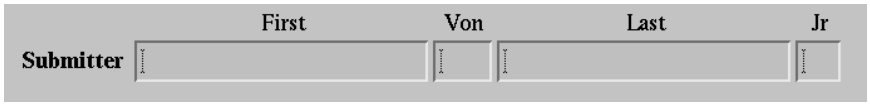


Die Angabe des Erfassernamens erfolgt in der Reihenfolge *Vorname*, *von-Teil*, *Nachname* und *Jr-Teil*. Das Schlüsselwort ist im Feld `key` anzugeben.

Über den Button `Input next key` können weitere Schlüsselwörter spezifiziert werden.

C.5.3 Suchen aller Einträge für einen Erfasser

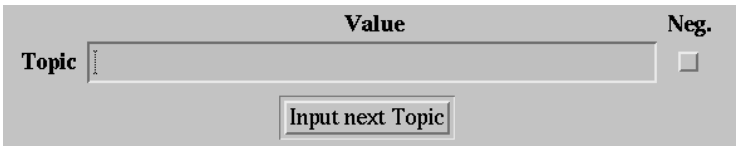
Mit der Funktion `Submitter` können alle Einträge ermittelt werden, die von einer bestimmten Person erfaßt wurden.



Der Name des Erfassers ist durch die einzelnen Bestandteile anzugeben.

C.5.4 Suchen nach Themen

Die Menü-Funktion `Topic` dient der Suche nach Einträgen, die bestimmten Themen zugeordnet werden können.



Das Thema wird im Feld `Topic` angegeben. Die Suche nach dem Thema erfolgt in den Felder *title*, *booktitle*, *abstract*, *keywords*, *note* und *annote*. Bei aktiviertem Checkbutton `Neg` umfaßt das Anfrageergebnis alle Einträge, die diesem Thema nicht zugeordnet werden können.

Über den Button `Input next Topic` können weitere Thema angegeben werden. Die Verknüpfung der einzelnen Themen wird im Feld `Link with` spezifiziert. Dieses Feld enthält eine Liste, aus der die Verknüpfung `and` oder `or` gewählt werden kann.

C.5.5 Nutzerdefinierte Anfragen

Über die Funktion `Userdefined` können Einträge ermittelt werden, deren Felder bestimmte Bedingungen erfüllen. Für nutzerdefinierte Anfragen wird das folgende Formular genutzt:

Ein Feld, nach dem gesucht werden soll, muß aus der Liste `Name` ausgewählt werden. In dieser Liste sind sowohl die Standardfelder als auch die frei definierbaren Felder enthalten. Der Wert, den das Feld haben soll, wird in `Value` spezifiziert. Bei der Angabe des Feldwertes können Wildcards (`*`) verwendet werden. Ist der Checkbutton `Neg` aktiviert, dann wird nach Einträgen gesucht, bei denen sich der Wert des Feldes vom angegebenen Wert unterscheidet.

Über den Button `Input next Field` kann man weitere Felder angeben. Die Verknüpfung der einzelnen Feldangaben ist im Feld `Link with` zu spezifizieren. Dieses Feld enthält eine Liste, aus der die Verknüpfung `and` oder `or` gewählt werden kann.

C.6 Export

Die Einträge, die exportiert werden sollen, sind über Anfragen zu ermitteln. Auf den Seiten, die das Anfrageergebnis enthalten, ist ein Formular vorhanden, über das der Export aller ermittelten Einträge veranlaßt werden kann. In dem Formular erfolgt die Auswahl des Formats, in das die Einträge exportiert werden sollen. Außerdem ist in diesem Formular anzugeben, ob die Abkürzungen durch die Kurz- oder Langform zu ersetzen sind.

Durch die Betätigung des `Export`-Buttons wird der Export gestartet. Dazu ist der Name der Datei anzugeben, in der die exportierten Einträge gespeichert werden sollen. Der Dateiname, der vom WWW-Browser vorgegeben wird, kann leider nicht beeinflußt werden.

C.7 Ändern von Einträgen

Der Eintrag, der geändert werden soll, ist über eine Anfrage (siehe Abschnitt C.5) zu ermitteln.

Für die Änderung eines Eintrags wird ein Formular genutzt, das die folgenden Komponenten enthält:

- Wenn der Benutzer den Eintrag erfaßt hat, dann ist in `key` das entsprechende Schlüsselwort enthalten. Ansonsten kann der Benutzer ein Schlüsselwort für den Eintrag angeben.
- Die Sprache, die dem Eintrag zugeordnet wurde, wird im Feld `language` ausgegeben. Eine andere Sprache kann in der zugehörigen Liste ausgewählt werden.
- Besitzt der Eintrag einen Querverweis, dann ist in `crossref` das Schlüsselwort und zugehörige Erfasser des Eintrags, auf den verwiesen wird, enthalten. Der Erfassername ist in der Reihenfolge *Vorname*, *von-Teil*, *Nachname* und *Jr-Teil* angegeben. Bei der Definition eines Querverweises muß man das Schlüsselwort und den zugeordneten Erfasser des Eintrags, auf den verwiesen werden soll, angeben.
- Wurden für den Eintrag Autoren angegeben, dann werden diese in `author` aufgelistet. Jedem Autoren ist ein Radiobutton zugeordnet. Will man einen Autoren löschen, dann muß der Radiobutton für den entsprechenden Autoren aktiviert und anschließend der Button `Delete marked Authors` betätigt werden. Über den Button `Add Author` kann man weitere Autoren hinzugefügen.

- Ist der Checkbutton `other authors` aktiviert, dann bedeutet dies, das die Autorenliste abgekürzt wurde. Bei Ausgaben wird an das Ende der Autorenliste `and other` angehängt.

- Alle Herausgeber, die für den Eintrag angegeben wurden, werden in `editor` aufgelistet. Will man einen Herausgeber löschen, dann muß der Radiobutton, der dem Herausgeber zugeordnet ist, aktiviert und anschließend der Button `Delete marked Editors` betätigt werden. Über den Button `Add Editor` kann man einen weiteren Herausgeber angeben.

- Ein aktivierter Checkbutton `other editors` steht für eine abgekürzte Herausgeberliste. In diesem Fall wird bei Ausgaben an das Ende der Herausgeberliste `and other` angehängt.

- In den weiteren Formularkomponenten sind die Felder, die für den Eintrag angegeben wurden, mit den zugeordneten Werten enthalten. Felder können gelöscht werden, indem der zugehörige Radiobutton aktiviert und anschließend der Button `Delete marked Fields` betätigt wird.

  Bei einigen Feldern werden die Werte für jeden Erfasser separat gespeichert. Für solche Felder gilt: Ein Benutzer kann nur den Wert ändern, der von ihm erfaßt worden ist.

Über den Button `Add Standard Field` kann man ein weiteres Standardfeld für den Eintrag angeben. Wenn man ein Feld spezifizieren will, das nicht in der Standardfeld-Liste enthalten ist, dann ist der Button `Add Free Field` zu betätigen.

Wird der `Change`-Button betätigt, dann werden die Änderungen in der Datenbank gespeichert. Bei der Abspeicherung des Eintrags kann es vorkommen, daß der Benutzer weitere Angaben vornehmen muß, wenn

- ein freies Feld angegeben wurde, das der Benutzer vorher noch nicht verwendet hat,

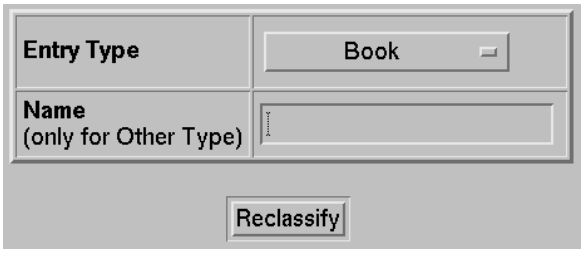
- in der Datenbank Einträge existieren, die eventuell mit dem geänderten Eintrag übereinstimmen, oder

- in der Datenbank Einträge vorhanden sind, denen der Benutzer das gleiche Schlüsselwort zugeordnet hat.

Die Formulare, die in diesen Fällen ausgegeben werden, wurden bereits in Abschnitt C.4 beschrieben.

C.8 Reklassifizierung von Einträgen

Die Änderung des Eintragstyps wird als Reklassifizierung bezeichnet. Der Eintrag, der reklassifiziert werden soll, ist über eine Anfrage (siehe Abschnitt C.5) zu bestimmen. Das Formular, mit dem die Reklassifizierung vorgenommen werden kann, sieht folgendermaßen aus:



Der neue Typ wird aus der Liste `Entry Type` ausgewählt. Wurde der Eintragstyp `Other_type` gewählt, dann muß im Feld `Name` der neue Typ eingegeben werden.

Anhang D

Datenbankschema

Im folgenden sind die Klassen- und Methodendefinitionen aufgelistet, die im Datenbankschema enthalten sind. Aus dem Schema `o2web` müssen die Klassen `O2WebFormItem`, `O2WebFormAnalyser`, `O2WebAssistant` und `O2WebInteractor` importiert werden.

D.1 Klassen des Datenbankschemas

D.1.1 Personen

```
class Person inherit Object public type
   tuple(first_part: string,
         von_part  : string,
         last_part : string,
         jr_part   : string)
method
   private encode_name(name: string): string,
   public init(f_part: string, v_part: string, l_part: string,
               j_part: string),
   public get_query: string,
   public report(userdata: string): bits,
   public short_report: bits,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string): bits
end;
```

D.1.2 Felder

Field_name

```
class Field_name inherit Object public type
```

```
   tuple(name: string, meaning: string, multiple_value: boolean)
method
   public is_standard: boolean,
   public is_submitter_depend: boolean,
   public is_free: boolean,
   public get_submitter: Person,
   public html_footer(query:string, userdata:string): bits,
   public get_query: string
end;
```

Standard_field

```
class Standard_field inherit Field_name public type
   tuple(refdbms_tag     : string,
         ncstrl_tag      : string,
         submitter_depend: boolean,
         bibtex          : boolean,
         refdbms         : boolean,
         ncstrl          : boolean)
method
   public init(n: string, m: string, r_t: string, n_t: string,
               s: boolean, mv: boolean, bib: boolean, ref: boolean,
               nc: boolean),
   public is_standard: boolean,
   public is_submitter_depend: boolean,
   public html_report(query: string, userdata: string): bits,
   public get_query: string
end;
```

Free_field

```
class Free_field inherit Field_name public type
   tuple(new_field: boolean,
         submitter: Person)
method
   public init(n: string, m: string, mv: boolean, nf: boolean,
               s: Person, persistent: boolean),
   public is_free: boolean,
   public get_submitter: Person,
   public html_report(query: string, userdata: string): bits,
   public get_query: string
end;
```

D.1.3 Feldwerte

Field_value

```
class Field_value inherit Object public
method
   public get_exp_value(language: string,
                        abbrv_format: integer): string,
   public get_exp_submitter_value(language: string,
                                  abbrv_format: integer,
                                  submitter: Person): string,
   public get_submitter_value(language: string, first: string,
                              von: string, last: string,
                              jr: string): string,
   public get_submitter_value_obj(first: string, von: string,
                                  last: string,
                                  jr: string): Field_value,
   public get_value(language: string): string,
   public check_value(value: string, like: boolean,
                      negation: boolean): boolean,
   public cp_value(new_value: Field_value, submitter: Person,
                   overwrite: boolean): boolean,
   public change_submitter_value(value: string,
                                 submitter: Person): Field_value,
   public delete_submitter_value(submitter: Person): Field_value
end;
```

Abbrevation

```
class Abbrevation inherit Field_value public type
   tuple(name   : string,
         value  : set(tuple(language: string, long_value : string,
                            short_value: string)),
         bibtex : boolean,
         refdbms: boolean,
         ncstrl:  boolean,
         fields : set(Field_name))
method
   public init(n: string, l: string, lv: string, sv: string,
               bib: boolean, ref: boolean, ncstrl: boolean,
               f: set(Field_name)),
   public add_value(l: string, lv: string, sv: string),
   public change_value(l: string, lv: string, sv: string),
   public get_exp_value(language: string,
                        abbrv_format: integer): string,
```

```
   public get_value(language: string): string,
   public check_value(value: string, like: boolean,
                      negation: boolean): boolean,
   public is_field(name: string): boolean,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string):bits
end;
```

Publisher

```
class Publisher inherit Field_value public type
   tuple(name   : Field_value,
         address: string,
         misc   : string)
method
   public init(n: Field_value, a: string, m: string),
   public get_value(language: string): string,
   public check_value(value: string, like: boolean,
                      negation: boolean): boolean
end;
```

Journal

```
class Journal inherit Field_value public type
   tuple(name: Field_value)
method
   public init(n: Field_value),
   public get_value(language: string): string,
   public check_value(value: string, like: boolean,
                      negation: boolean): boolean
end;
```

Submitter_depend_value

```
class Submitter_depend_value inherit Field_value public type
   set(tuple(value: string, submitter: Person))
method
   public init(v: string, s: Person),
   public add_value(v: string, s: Person),
   public get_submitter_value(language: string, first: string,
                              von: string, last: string,
                              jr: string): string,
   public get_submitter_value_obj(first: string, von: string,
                                  last: string,
```

```
                                jr: string): Field_value,
   public get_exp_submitter_value(language: string,
                                abbrv_format: integer,
                                submitter: Person): string,
   public get_value(language: string): string,
   public check_value(value: string, like: boolean,
                      negation: boolean): boolean,
   public cp_value(new_value: Field_value, submitter: Person,
                   overwrite: boolean): boolean,
   public change_submitter_value(value: string,
                                 submitter: Person): Field_value,
   public delete_submitter_value(submitter: Person): Field_value
end;
```

Single_value

```
class Single_value inherit Field_value public type
   tuple(value: string)
method
   public init(v: string),
   public get_value(language: string): string,
   public check_value(value: string, like: boolean,
                      negation: boolean): boolean
end;
```

D.1.4 Eintrag

```
class Entry inherit Object public type
   tuple(ID           : integer,
         language     : string,
         fields       : set(tuple(f_name: Field_name,
                                  f_value: Field_value)),
         crossref     : Entry,
         submitter_key: set(tuple(key: string, submitter: Person)),
         authors      : set(tuple(author: Person, note: string,
                                  organization: string)),
         editors      : set(tuple(editor: Person, note: string)),
         other_authors: boolean,
         other_editors: boolean)
method
   private author_report(authors: set(tuple(author: Person,
                                            note: string,
                                            organization: string)),
                         other: boolean, userdata: string,
```

```
                      link: boolean): bits,
private editor_report(editors: set(tuple(editor: Person,
                                         note: string)),
                      other: boolean, userdata: string,
                      link: boolean): bits,
public wrap_str(value: string, f_name_length: integer): string,
public init(l: string, c: Entry, k: string, s: Person),
public add,
public delete,
public author: string,
public editor: string,
public compare_entry: list(Entry),
public add_field(name: Field_name, value: Field_value),
public get_field(name: string): Field_value,
public get_whole_field(name: string, submitter: Person):
                       tuple(f_name: Field_name,
                             f_value: Field_value),
public get_authors: string,
public get_author_set: set(Person),
public get_editor_set: set(Person),
public get_submitter_value(field: Field_name, first: string,
                           von: string, last: string,
                           jr: string): string,
public get_submitter_value_obj(field: tuple(f_name:Field_name,
                                            f_value: Field_value),
                               first: string, von: string,
                               last: string,
                               jr: string): Field_value,
public get_value(field: string): string,
public cp_value(field: tuple(f_name: Field_name,
                             f_value: Field_value),
                submitter: Person, overwrite: boolean),
public check_value(field: string, value: string, like: boolean,
                   negation: boolean): boolean,
public reclassify(new_class: string, name: string),
public export_bibtex(abbrv_format: integer,
                     submitter: Person): bits,
public export_refdbms(abbrv_format: integer,
                      submitter: Person): bits,
public export_ncstrl(abbrv_format: integer,
                     submitter: Person): bits,
public get_query: string,
public add_author(author: Person, note: string,
                  organization: string),
public add_editor(editor: Person, note: string),
```

```
   public set_other_author(bool: boolean),
   public set_other_editor(bool: boolean),
   public report(query_str: string, userdata: string): bits,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_string: string,
   public type_nr: integer,
   public html_header(query:string, userdata:string):bits,
   public html_footer(query:string, userdata:string):bits,
   public html_report(query: string, userdata: string): bits
end;
```

D.1.5 Eintragstypen

Article

```
class Article inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Book

```
class Book inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Booklet

```
class Booklet inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
```

```
   public add,
   public delete,
   public bibtex_title: string,
   public type_nr: integer
end;
```

Conference

```
class Conference inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public type_nr: integer
end;
```

Inbook

```
class Inbook inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Incollection

```
class Incollection inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public type_nr: integer
end;
```

Inproceedings

```
class Inproceedings inherit Entry public
```

```
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Manual

```
class Manual inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Mastersthesis

```
class Mastersthesis inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public type_nr: integer
end;
```

Miscellaneous

```
class Miscellaneous inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Phdthesis

```
class Phdthesis inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Proceedings

```
class Proceedings inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Techreport

```
class Techreport inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public refdbms_title: string,
   public type_nr: integer
end;
```

Unpublished

```
class Unpublished inherit Entry public
method
   public init(l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
```

```
   public refdbms_title: string,
   public type_nr: integer
end;
```

Other_type

```
class Other_type inherit Entry public type
   tuple(name: string)
method
   public init(n: string, l: string, c: Entry, k: string, s: Person),
   public add,
   public delete,
   public bibtex_title: string,
   public type_string: string,
   public type_nr: integer
end;
```

D.2 Klassen für die Datenbankfunktionen

Identity

```
class Identity private type
   tuple(ID: integer, free_IDs: unique set(integer))
method
   public init,
   public get_ID: integer,
   public add_free_ID(ID: integer)
end;
```

Configuration

```
class Configuration inherit Object public type
   tuple(parserdir      : string,
         parserappletdir: string,
         wwwservertmpdir: string)
end;
```

o2_set_Abbrevation

```
class o2_set_Abbrevation public type
   set(Abbrevation)
method
```

```
   public add(userdata: string): bits,
   public add_abbrev(params: bits): bits,
   public add_language1(params: bits): bits,
   public change(params: string): bits,
   public change_abbrev(params: bits): bits,
   public add_language(params: bits): bits,
   public html_header(query:string, userdata:string): bits,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string): bits
end;
```

Query

```
class Query
method
   public search_def(t: string, n: integer, s: list(string),
                     so: string): bits,
   public sort(sort: list(string), order: string): string,
   public compare(value: string): string,
   public convert(value: string): string,
   public is_like(value: string): boolean,
   public print_entry(e: Entry, n: integer, out: boolean,
                      param: string): bits,
   public html_prolog(query:string, userdata:string):bits,
   public html_header(query:string, userdata:string):bits,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string):bits,
   public export(params: bits): bits,
   public export_entry(entry: Entry, exp_format: integer,
                       abbrv_format: integer,
                       submitter: Person): bits,
   public export_table: bits,
   public search(params: bits): bits,
   public search_first(query: string, typ: string,
                       params: bits): bits,
   public search_next(params: bits): bits
end;
```

Author_query

```
class Author_query
method
   private single_column(c: integer, n: string, f: string, v: string,
                         l: string, j: string, neg: string,
```

```
                        link: string): bits,
   private single_author(t: string, n: integer, s: list(string),
                        so: string, f: string, v: string, l: string,
                        j: string, a_f: string, a_v: string,
                        a_l: string, a_j: string, neg: string): bits,
   private search_author(params: bits): bits,
   public html_prolog(query:string, userdata:string):bits,
   public html_header(query:string, userdata:string):bits,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string):bits,
   public search(params: bits): bits
end;
```

Key_query

```
class Key_query
method
   private submitter(f: string, v: string, l: string,
                     j: string): bits,
   private single_column(c: integer, n: string,
                        key: string): bits,
   private single_value(t: string, n: integer, s: list(string),
                       so: string, f: string, v: string, l: string,
                       j: string, key: string, s_f: string,
                       s_v: string, s_l: string, s_j: string): bits,
   private query(params: bits): bits,
   public html_prolog(query:string, userdata:string):bits,
   public html_header(query:string, userdata:string):bits,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string):bits,
   public search(params: bits): bits
end;
```

Submitter_query

```
class Submitter_query
method
   public html_prolog(query:string, userdata:string):bits,
   public html_header(query:string, userdata:string):bits,
   public html_report(query_str: string, userdata: string): bits,
   public html_footer(query:string, userdata:string):bits,
   public search(params: bits): bits
end;
```

Topic_query

```
class Topic_query
method
   private single_column(c: integer, n: string, topic: string,
                         neg: string, link: string): bits,
   private single_value(t: string, n: integer, s: list(string),
                        so: string, f: string, v: string, l: string,
                        j: string, topic: string, neg: string): bits,
   private query(params: bits): bits,
   private add_field(field: string, topic: string, like: string,
                    neg: string): string,
   public html_prolog(query:string, userdata:string):bits,
   public html_header(query:string, userdata:string):bits,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string):bits,
   public search(params: bits): bits
end;
```

Userdef_query

```
class Userdef_query
method
   private header(prolog: boolean): bits,
   private single_column(c: integer, n: string, f_name: string,
                         ffield: string, f_value: string, neg: string,
                         link: string): bits,
   private single_value(t: string, n: integer, s: list(string),
                        so: string, f: string, v: string, l: string,
                        j: string, f_name: string, ffield: string,
                        f_value: string, neg: string): bits,
   private query(params: bits): bits,
   public html_prolog(query:string, userdata:string):bits,
   public html_header(query:string, userdata:string):bits,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string):bits,
   public search(params: bits): bits
end;
```

Copy

```
class Copy
method
   public html_header(query:string, userdata:string):bits,
```

```
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string): bits,
   public parser(params: bits): bits
end;
```

Start

```
class Start
method
   public html_header(query:string, userdata:string):bits,
   public html_report(query: string, userdata: string): bits
end;
```

Menu

```
class Menu
method
   public  html_header(query:string, userdata:string):bits,
   public  html_report(query: string, userdata: string): bits,
   public  menu_report(params: bits): bits
end;
```

Reclassify

```
class Reclassify
method
   public reclassify(params: bits): bits,
   public new_class(params: bits): bits
end;
```

Change

```
class Change
method
   private person_table(header: boolean, prolog: string, p: Person,
                       del: boolean): bits,
   private person_table1(header: boolean, prolog: string,
                        first: string, von: string, last: string,
                        jr: string, del: boolean): bits,
   public  change(params: bits): bits,
   public  change_entry(params: bits): bits,
   public  change_entry_values(params: bits): bits,
   public  write_publisher(entry: Entry, submitter: Person,
```

```
                        publisher: tuple(name: string,
                                         address: string)):
                        tuple(new_field: boolean,
                              field: tuple(f_name: Field_name,
                                           f_value: Field_value)),
public  write_field(entry: Entry, field: tuple(name: string,
                                               value: string),
                    submitter: Person):
                    tuple(new_field: boolean,
                          field: tuple(f_name: Field_name,
                                       f_value: Field_value)),
private make_field_value(fname: Field_name, value: string,
                         language: string,
                         submitter: Person): Field_value,
private make_field_name(name: string,
                        submitter: Person): Field_name,
private get_abbrevation(name: string): Abbrevation,
private get_abbrev_value(value: string): Abbrevation,
private make_publisher(name: string, address: string,
                       language: string): Publisher,
private make_journal_field(name: string,
                           language: string): Journal,
private make_abbrev_field(value: string,
                          language: string): Field_value,
private make_normal_field(name: Field_name, value: string,
                          language: string,
                          submitter: Person): Field_value,
private make_free_field(value: string, language: string,
                        submitter: Person): Field_value,
private header(prolog: boolean): bits,
private error(errmsg: string, first: string, von: string,
              last: string, jr: string): bits,
private display_free_field(name: string): bits,
private check_entry(key: string, e: Entry,  first: string,
                    von: string, last: string,
                    jr: string): bits,
private is_double_key(key: string, ID: integer, first: string,
                      von: string, last: string,
                      jr: string): boolean,
private double_key(key: string,  first: string, von: string,
                   last: string, jr: string): bits,
public  free_field(params: bits): bits,
public  equal_entries(params: bits): bits,
public  cp_change_key(params: bits): bits,
public  change_key(params: bits): bits,
```

```
   private table_col(name: string, old_value: string,
                     new_value: string): bits,
   private copy_entry(first: string, von: string, last: string,
                      jr: string, e: Entry, key: string,
                      ID_str: string): bits,
   private cp_check_key(first: string, von: string, last: string,
                        jr:string, key: string, e: Entry,
                        ID: string): bits,
   public  cp_entry(params: bits): bits,
   private cp_entry_values(entry: Entry, old_entry: Entry,
                           key: tuple(key: string, submitter: Person),
                           diffs: set(tuple(f_name: Field_name,
                                            f_value: Field_value))),
   private check_crossref(entry: Entry, old_entry: Entry),
   private copy_entry_values(entry: Entry, old_entry: Entry,
                             first: string, von: string, last: string,
                             jr: string, params: bits)
end;
```

Tmp

```
class Tmp inherit Object public type
   tuple(entry: Entry)
method
   public init(e: Entry),
   public delete_obj,
   public move,
   public cp_entry(old_entry: Entry,
                   key: tuple(key: string, submitter: Person),
                   diffs: set(tuple(f_name: Field_name,
                                    f_value: Field_value))),
   public check_crossref(old_entry: Entry),
   public copy_entry(old_entry: Entry, params: bits)
end;
```

o2_set_Tmp

```
class o2_set_Tmp inherit Object public type
   set(Tmp)
method
   public  delete_submitter_obj(submitter: Person),
   private header(prolog: boolean): bits,
   private error(errmsg: string, first: string, von: string,
                 last: string, jr: string): bits,
```

```
   private move_entry(first: string, von: string, last: string,
                      jr:string, tmp_obj: Tmp): bits,
   private check_free_field(first: string, von: string, last: string,
                            jr:string, e: Entry): bits,
   private check_entry(first: string, von: string, last: string,
                       jr:string, e: Entry): bits,
   private table_col(name: string, old_value: string,
                     new_value: string): bits,
   private copy_entry(first: string, von: string, last: string,
                      jr:string, t: Tmp, ID_str: string): bits,
   private cp_check_key(first: string, von: string, last: string,
                        jr:string, key: string, e: Entry,
                        ID: string): bits,
   private check_key(first: string, von: string, last: string,
                     jr:string, e: Entry): bits,
   public  free_field(params: bits): bits,
   public  equal_entries(params: bits): bits,
   public  cp_entry(params: bits): bits,
   public  change_key(params: bits): bits,
   public  cp_change_key(params: bits): bits,
   public  next(params: bits): bits,
   public  move(params: bits): bits,
   public  html_prolog(query:string, userdata:string):bits,
   public  html_header(query:string, userdata:string):bits,
   public  html_footer(query:string, userdata:string): bits,
   public  html_report(query: string, userdata: string): bits
end;
```

D.3 Klassen für die HTML-Generation

URLtool

```
class URLtool public type
   set(tuple(item: string, value: string))
method
   public init(userdata: string),
   public hex_decode(str: string): string,
   public encode(str: string): string,
   public hex_encode(str: string): string,
   public get_value(name: string): string,
   public is_item(name: string): boolean
end;
```

O2WebInteractor

```
rename class O2WebInteractor as ImportedO2WebInteractor;
class O2WebInteractor inherit ImportedO2WebInteractor
method
   public prolog: string,
   public header: bits,
   public footer: bits,
   public error(kind: integer): bits
end;
```

HTMLtool

```
class HTMLtool
method
   public named_form_begin(name: string, enctype: string,
                           action: string): bits,
   public form_begin(enctype: string, action: string): bits,
   public form_end: bits,
   public input_field(name: string, length: integer,
                      value: string): bits,
   public check_field(name: string, value: string,
                      checked: boolean): bits,
   public radio_field(name: string, value: string,
                      checked: boolean): bits,
   public check_sel_field(name: string, value: string,
                          compare: string): bits,
   public hidden_field(name: string, value: string): bits,
   public textarea(name: string, rows: integer, cols: integer): bits,
   public clickbutton(value: string, onclick: string): bits,
   public submit_button(value: string): bits,
   public submit_clickbutton(value: string, onclick: string): bits,
   public submit_name_button(value: string, name: string): bits,
   public submit_name_clickbutton(value: string, name: string,
                                  onclick: string): bits,
   public list_image(image: string): bits,
   public get_form_value(formtool: O2WebFormAnalyser,
                         formitems: list(O2WebFormItem),
                         value: string): string,
   public get_form_set_value(formtool: O2WebFormAnalyser,
                             formitems: list(O2WebFormItem),
                             value: string): list(string),
   public sbox_begin(name: string, multiple: boolean,
                     size: integer): bits,
   public sbox_option(value: string, selected: boolean,
```

```
                        name: string): bits,
   public sbox_sel_option(value: string, compare: string,
                          name: string): bits,
   public sbox_sel_comp_option(value: string, compare: string,
                               compare1: string,
                               name: string): bits,
   public sbox_sel_set_option(value: string, compare: list(string),
                              name: string): bits,
   public sbox_end: bits,
   public navigation_button(url: string, image: string,
                            alt: string): bits,
   public navigation_clickbutton(url: string, image: string,
                                 alt: string,
                                 onclick: string): bits,
   public submit_image_button(image: string, alt: string): bits,
   public table_begin(border: integer, spacing: integer,
                      padding: integer): bits,
   public table_end: bits,
   public table_row_begin: bits,
   public table_row_end: bits,
   public table_column(value: string): bits,
   public table_column_ext(ext: string, value: string): bits,
   public table_header(value: string): bits
end;
```

FField_list

```
class FField_list inherit Object
method
   public html_header(query: string, userdata: string): bits,
   public html_report(query: string, userdata: string): bits,
   public html_footer(query:string, userdata:string): bits
end;
```

D.4 Klassen für das ODMG C++-Bindung

o2_set_Object

```
class o2_set_Object inherit Object public type
   set(Object)
end;
```

Abbrev_list

```
class Abbrev_list inherit Object public type
   tuple(abbrevs: list(Abbrevation))
method
   public add(a: Abbrevation): integer,
   public get(i: integer): Abbrevation
end;
```

Field_list

```
class Field_list inherit Object public type
   tuple(fields: list(Field_name))
method
   public add(a: Field_name): integer,
   public get(i: integer): Field_name,
   public get_name(n: string): Field_name
end;
```

Cross_list

```
class Cross_list inherit Object public type
   set(tuple(key: string, entry: Entry))
method
   public add(k: string, e: Entry),
   public get(k: string): Entry,
   public get_number: integer
end;
```

D.5 Definierte Extensionen

```
name identity: Identity;
name configuration: Configuration;
name person: set(Person);
name standard_field: set(Standard_field);
name free_field: set(Free_field);
name abbrevation: o2_set_Abbrevation;
name publisher: set(Publisher);
name journal: set(Journal);
name entries: set(Entry);
name article: set(Article);
name book: set(Book);
name booklet: set(Booklet);
```

```
name conference: set(Conference);
name inbook: set(Inbook);
name incollection: set(Incollection);
name inproceedings: set(Inproceedings);
name manual: set(Manual);
name masterstheses: set(Masterstheses);
name miscellaneous: set(Miscellaneous);
name phdthesis: set(Phdthesis);
name proceedings: set(Proceedings);
name techreport: set(Techreport);
name unpublished: set(Unpublished);
name other_type: set(Other_type);
name query: Query;
name author_query: Author_query;
name key_query: Key_query;
name submitter_query: Submitter_query;
name topic_query: Topic_query;
name userdef_query: Userdef_query;
name copy: Copy;
name start: Start;
name menu: Menu;
name reclassify: Reclassify;
name change: Change;
name ffield_list: FField_list;
name tmp: o2_set_Tmp;
name TheO2WebInteractor: O2WebInteractor;
```

Erklärung

Ich erkläre, daß ich die vorliegende Arbeit selbstständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, 01.Juni 1997 Gunnar Weber

Thesen

1. In den Literaturdatenbanken BibTeX und *refdbms* werden die einzelnen Einträge bestimmten Typen zugeordnet. Die Eintragstypen unterscheiden sich teilweise im BibTeX- und im *refdbms*-Format, sie sind aber aufeinander abbildbar.

2. Zwischen den Feldern, die in den Literaturdatenbanken BibTeX, *refdbms*, und *NCSTRL* definiert sind, treten Konflikte auf. Diese Konflikte können durch die Zuordnung von Feldwerten bzw. bestimmter Teilinformationen aus den Feldwerten zu anderen Feldern gelöst werden.

3. Die in den Literaturdatenbanken vorhandenen Eintragstypen werden durch eigene Klassen dargestellt. Dies hat den Vorteil, daß die Methoden in Abhängigkeit vom Eintragstyp implementiert werden können.

4. Die Felder der Einträge werden nicht in eigenen Klassenattributen, sondern in einem mengenwertigen Attribut gespeichert, da die Menge der Felder, die man für einen Eintrag angeben kann, sehr variabel ist.

5. Der Wert einiger Felder ist vom Erfasser abhängig. In solchen Felder müssen die Werte separat für jeden Erfasser gespeichert werden.

6. Die Voraussetzung für den Import einer Datei mit Literaturreferenzen ist die Übertragung dieser Datei an den WWW-Server. Die Übertragung kann über eine `multipart/form-data`-Mitteilung erfolgen, die aus einem Formular generiert und anschließend an den WWW-Server gesendet wird.

7. Für OQL-Anfragen wird in O_2C eine Systemfunktion genutzt. Bei der dynamischen Zusammenstellung von Anfragen über Formulare ergeben sich Probleme, da der Ergebnistyp statisch vorgegeben werden muß.