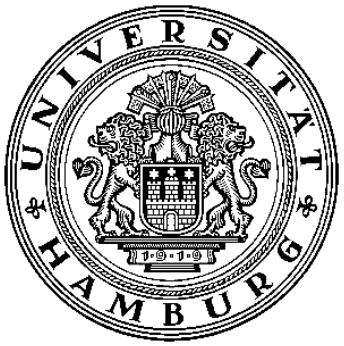

Universität Hamburg
FB Informatik

Information Retrieval Unterstützung zur kooperativen Arbeit mit Bibliographiedatenbanken

Anforderungsanalyse und Systementwurf am Beispiel des BibTeX-Systems

Studienarbeit

eingereicht bei

Prof. Dr. J. W. Schmidt
Arbeitsbereich Softwaresysteme
Technische Universität Hamburg-Harburg

von
André Wittenburg

26. Juni 1998

Inhaltsverzeichnis

1 Einleitung und Motivation **2**

2 BibTeX, eine kurze Einführung **5**
2.1 BibTeX Klassen und Eintragstypen 6
2.2 BibTeX Felder 7

3 Anforderungsanalyse **8**
3.1 Aufgabenbeschreibung 8
3.2 Systemvoraussetzungen 9
3.3 Anwendungsfallanalyse 9
3.4 Klassen und Domänen 9
3.5 Merging-Algorithmen 14

4 Information Retrieval **16**
4.1 String Matching Methoden 17
4.1.1 Edit Distance 17
4.1.2 Soundex und Phonix 18
4.2 Bewertung von IR-Systemen 19
4.2.1 Stoppwort-Elimination und Stammformreduktion 20
4.2.2 Indexing in IR-Systemen 20
4.3 Eine Fallstudie 20

5 Systementwurf **22**
5.1 Tycoon-2 22
5.2 Structured Tycoon Markup Language - STML 23
5.3 Systemarchitektur 25
5.3.1 Verwaltung 25
5.3.2 Applikation 28
5.3.3 Präsentation 31

6 Zusammenfassung **35**

7 Ausblick **37**

Anhang **37**

A BibTeX Eintragstypen **38**

B BibTeX-Stildatei **40**

C UML-Diagramme **43**

D Systembeschreibung **46**
D.1 BibAlgo . . . 46
D.2 BibApplication . . . 47
D.3 BibCheckClass . . . 49
D.4 BibCheck . . . 50
D.5 BibEntry . . . 51
D.6 BibEntryClass . . . 51
D.7 BibFile . . . 51
D.8 BibFileClass . . . 51
D.9 BibMergeClass . . . 51
D.10 BibMerge . . . 52
D.11 BibParameter . . . 52
D.12 BibParameterClass . . . 52
D.13 BibProcessor . . . 53
D.14 BibQuestionClass . . . 53
D.15 BibQuestion . . . 53
D.16 BibRankedList . . . 54
D.17 BibTriple . . . 54

Abbildungsverzeichnis

1.1 Teilergebnis der Suchanfrage: *(latex) and(lamport)* 3

2.1 ein- und ausgehende Dateien für das BibTeX-System 6

3.1 mögliche Ergebnismengen des Systems 8
3.2 Ablaufdiagramm der Anforderungsanalyse 12
3.3 BibTeX-Domänen 13

4.1 Rekurrenzgleichung für Edit Distance 18
4.2 Zusammenhang zwischen relevanten und gefundenen Objekten in IR-Systemen . . 19

5.1 Quelltext der Klassen *MutableListClass* und *MutableList* 23
5.2 Beispiel einer STML-Seite 24
5.3 Schichtenmodell für die Systemarchitektur 25
5.4 Objekte einer BibTeX-Datei – Instanzendiagramm 26
5.5 *aux*-Datei zum Importieren der BibTeX-Dateien 28
5.6 Bildschirmausschnitt - Hauptmenü des BibTeX Merge-Tool 33
5.7 Bildschirmausschnitt - Vergleichen zweier BibTeX-Einträge 34

C.1 Use Case Modell 43
C.2 Klassendiagramm 44
C.3 Ablaufdiagramm des Systementwurfs 45

Kapitel 1

Einleitung und Motivation

Ausgangspunkt für diese Arbeit bildet folgendes Szenario:

> Person A und Person B besitzen eine Datenbank, in der sie getrennt voneinander ihre Literaturreferenzen verwalten. A und B möchten nun zusammen ein Paper schreiben und wollen dazu Ihre Literaturreferenzen abgleichen, um zu sehen, welche Quellen beiden bekannt sind und welche der andere noch nicht kennt.

Im universitären Bereich ist für die Archivierung von Literaturreferenzen das BibTeX-System weit verbreitet, so daß dieses System auf die obige Problemstellung hin genauer untersucht wird.

Einen Suchdienst im WWW bietet die Universität Karlsruhe unter folgender URL an:

> liinwww.ira.uka.de/bibliography/

Hier haben die Benutzer die Möglichkeit, ihre eigene BibTeX-Datei hinzuzufügen und in den Einträgen Anderer zu suchen. Mit der Zeit ist eine große Datenbank aus den verschiedensten BibTeX-Dateien entstanden.

Auf der Suche nach Einträgen zu Information Retrieval erhält man zwar eine Menge von Literaturvorschlägen, doch viele erscheinen mehrfach.
Der Nachteil resultiert aus dem Verfahren, die verschiedenen Dateien zusammenzuführen (mergen). Die einzelnen Einträge werden nicht miteinander verglichen, sondern nach Stichwörtern durchsucht und anschließend mit Angabe des Eigentümers der BibTeX-Datei abgelegt.

Folgender Suchstring könnte als Anfrage gestellt werden:

> *(latex) and (lamport)*

Einen Teil des Suchergebnisses sehen Sie in Abbildung 1.1.
Wenn man aus dem Gesamtergebnisses die Duplikate eliminiert, so erhält man bei insgesamt 20 Einträgen *nur* 6 verschiedene Bücher.
Die Felder einiger Einträge sind fast deckungsgleich (vgl. Einträge 4 und 5). Andere wiederum unterscheiden sich enorm (vgl. Einträge 1 und 6). Wenn man die einzelnen BibTeX Einträge vergleicht, so findet man keine zwei Einträge, in denen alle Felder den gleichen[1] Inhalt haben.

[1] Gleich bezieht sich hier auf die Gleichheit jedes einzelnen Zeichens innerhalb der Felder.

```
1.)
From Bibliography of the library of the Computer Science Department
 at the University of Braunschweig, Germany (1986):
   Leslie Lamport
   LaTeX: A Document Preparation System - User's Guide & Reference
   Manual
   , Addison-Wesley, 1986.
2.)
From Peter B. Danzig's personal bibliographic collection:
   Leslie Lamport
   A Document Preparation System LATEX
   , Addison-Wesley, 1986.
3.)
From Bibliography of the Signal Processing Group at
Rogaland University:
   L. Lamport
   LaTeXUser's Guide and Reference Manual
   , Addison Wesley, 1986 or later.
4.)
From Bibliography on TeX and METAFONT:
   Leslie Lamport
   LaTeX--- A Document Preparation System---User's Guide
   and Reference Manual
   , p. xiv + 242, pub-AW, 1985.
5.)
From Bibliography of publications related to typesetting, primarily
computer-based typesetting:
   Leslie Lamport
   LaTeX: a document preparation system
   , p. xiv + 242, pub-AW, 1986.
6.)
From Paolo Ciancarini's personal bibliography:
   L. Lamport
   LaTeX. User's Guide & Reference Manual.
   , aw, 1986.
7.)
From Bibliography relating to algebra, program specification
and verification, and logic:
   Leslie Lamport
   \LaTeX User Guide and Reference Manual
   , Addison-Wesley, 1985.
```

Abbildung 1.1: Teilergebnis der Suchanfrage: *(latex) and(lamport)*

Es ergibt sich folgendes Problem:

- BibTeX-Datenbanken werden von unterschiedlichen Personen unabhängig voneinander erstellt.
- Zwei oder mehrere BibTeX-Datenbanken sollen zusammengeführt werden.
- Semantische Duplikate sollen gefunden und gegebenenfalls eliminiert werden.

Da die Grundlage für diese Arbeit das BibTeX-System ist, befindet sich in Kapitel 2 eine kurze Einführung in den Aufbau und Strukturierung von BibTeX-Datenbanken.

Die Anforderungen an das System werden in Kapitel 3 beschrieben.

Doch welche Möglichkeiten und Algorithmen bietet die Informatik, um Lösungsansätze für das beschriebene Problem zu finden? Ansätze hierzu bietet das Gebiet des *Information Retrieval*, welches sich unter anderem mit der inhaltlichen Suche in Texten, dem sogenannten *Text Retrieval*, beschäftigt. Welche Grundideen Information Retrieval beinhaltet, und einige Lösungsansätze finden sich in Kapitel 4.
Den Systementwurf innerhalb einer objektorientierten Umgebung befindet sich in Kapitel 5.
Die Zusammenfassung in Kapitel 6 enthält ein Resumee über die Implementation und die genutzten Ressourcen. Abschließend werden im Ausblick mögliche Erweiterungen des Systems vorgestellt.

Kapitel 2

BibTeX, eine kurze Einführung

Das BibTeX-System soll die Erstellung von Literaturverzeichnissen in LaTeX-Dokumenten erleichtern und die Verwaltung von Literaturangaben unterstützen. Hierzu besitzt jeder Benutzer dieses Systems eine oder mehrere Literaturdateien, in denen er seine Literaturdaten verwaltet. Mit Hilfe des *\cite* oder des *\nocite*-Befehls werden Literaturverweise zu einer LaTeX-Datei hinzugefügt. BibTeX übernimmt dann anschließend aus der Hilfsdatei die entsprechenden Daten und erstellt das Literaturverzeichnis. Die Art des Aufbaus der einzelnen Einträge wird durch eine BibTeX-Stildatei angegeben.
Innerhalb der Literaturdateien stehen eine Vielzahl von Einträgen. Ein Eintrag hat zum Beispiel folgende Form:

```
@Book{Lamport95,
  author =      "Leslie Lamport",
  title =       "Das LaTeX Handbuch",
  pages =       "325",
  publisher =   "Addison-Wesley",
  year =        "1995",
  isbn =        "3-89319-826-1",
  price =       "69,90"
      }
```

Zu Beginn jedes Eintrags steht der Eintragstyp. In diesem Fall handelt es sich um den Typ *book*, was durch *@Book* definiert wird. *Lamport95* entspricht dem *Schlüssel*, der in dem LaTeX-Dokument auf dieses Buch referenziert. Dieser Eintrag enthält zusätzlich sieben Felder, wobei `author, title, publisher` und `year` *zwingende* Einträge sind. Die anderen Einträge werden vom BibTeX-System *ignoriert*. Diese Zuordnung, die für die standard Stildateien gültig ist, kann jedoch durch die BibTeX-Stildatei beeinflußt werden. Zusätzlich gibt es noch *optionale* Einträge, die eine Ausgabe erzeugen (vgl. Abschnitt 2.1).
Wichtig für spätere Betrachtungen ist, daß BibTeX Groß- und Kleinschreibung der Buchstaben im Eintragstyp, im Schlüssel und in den Feldnamen ignoriert, so daß ein Parser diese Zeichenketten generell in Kleinbuchstaben übersetzen könnte, ohne daß Information verloren geht. Des weiteren können statt der Anführungszeichen bei Feldern auch geschweifte Klammern verwendet werden oder diese weggelassen werden, wenn es sich bei den Einträgen um Zahlen oder Makros handelt.

Wie BibTeX bestimmte Einträge in LaTeX-Dokumenten darstellt, ist für diese Arbeit nicht relevant, da dies durch die BibTeX-Stildatei beeinflußt wird.

In Abbildung 2.1 sind die Dateien zu sehen, die für das BibTeX-System eine Rolle spielen. Nachfolgend sind die Dateien beschrieben:

- *aux*-Datei:
  LaTeX erzeugt diese Datei. Sie enthält alle Schlüssel zu den gewünschten Litraturreferenzen und Angaben über die zu benutzende Stildatei und BibTeX-Datei.

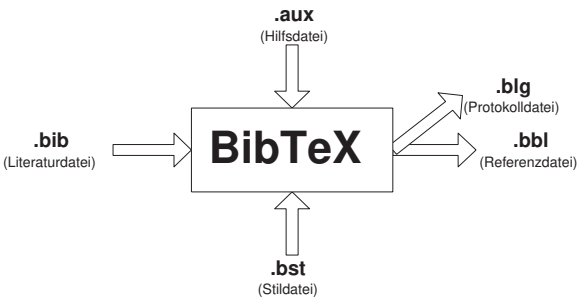

Abbildung 2.1: ein- und ausgehende Dateien für das BIBTEX-System

- *bst*-Datei:
  Die BIBTEX-Stildatei gibt an, wie die Ausgabe formatiert sein soll. Hierzu zählt sowohl die Formatierung der Einträge im Literaturverzeichnis, als auch die Angabe im Text.
- *bib*-Datei:
  Die Literaturdatei enthält alle Literaturreferenzen, die der Benutzer eingetragen hat.
- *bbl*-Datei:
  Die Referenzdatei enthält das fertige Literaturverzeichniss, das im Anschluß vom LATEX-System genutzt werden kann.
- *blg*-Datei:
  Die Logdatei des BIBTEX-Systems enthält alle Fehlermeldungen und Warnungen, die während des Aufrufs entstanden sind.

2.1 BIBTEX Klassen und Eintragstypen

Wie bereits erwähnt, gibt es zu jedem Eintragstyp drei verschiedene Klassen von Feldern, die im folgenden näher beschrieben werden:

zwingend
Wenn ein Feld dieser Klasse ausgelassen wird, so erzeugt BIBTEX beim Übersetzen eine Fehlermeldung. Diese kann jedoch ignoriert werden.

optional
Diese Felder werden im Literaturverzeichnis in LATEX-Dokumenten berücksichtigt und erzeugen eine Ausgabe.

ignoriert
BIBTEX ignoriert diese Klasse von Feldern völlig. Ein Beispiel für diese Klasse ist das Feld `price`.

Die Felder der Klasse *ignoriert* haben nicht automatisch einen niedrigeren Informationsgehalt als *optionale* Felder. Die `ISBN` ist bei keinem Eintragstyp ein *zwingendes* oder *optionales* Feld. Falls jedoch zwei BIBTEX-Einträge die gleiche `ISBN` haben, so ist die Wahrscheinlichkeit recht groß, daß es sich um semantisch gleiche Einträge handelt. Allerdings gilt die Aussage, das die `ISBN` ein *ignoriertes* Feld ist, nur für die Standard Stildateien des BIBTEX-Systems. Verwendet man eine Stildatei, die der Deutschen Industrie Norm (DIN) unterliegt, so ist die ISBN ein optionales Feld und erscheint in den Literaturverzeichnissen von LATEX-Dokumenten.
Im Anhang A befindet sich eine Aufzählung der BIBTEX Eintragstypen, die in den standard Stildateien enthalten sind.

2.2 BIBTEX Felder

Die von BIBTEX unterstützten Felder sind:
`address, annote, author, booktitle, chapter, crossref, edition, editor, howpublished, institution, journal, key, month, note, number, organization, pages, publisher, school, series, title, type, volume, year`

BIBTEX erzeugt, beim Auslassen eines *zwingenden* Feldes nur eine Fehlermeldung, die jedoch zu keinem Abbruch führt. Es ist somit nicht möglich anzunehmen, daß alle *zwingenden* Felder ausgefüllt sind.
Die Felder `key` und `crossref` haben besondere Bedeutungen. Das `key`-Feld kann zur Formatierung verwendet werden und hat nichts mit dem Eintragsschlüssel gemein. Das Feld `crossref` verweist auf einen anderen Eintrag innerhalb der BIBTEX-Datei. Wenn ein BIBTEX-Eintrag dieses Feld enthält, so werden alle nicht spezifizierten Felder des Eintrages aus dem referenzierten Eintrag übernommen.
Der Textteil eines Feldes ist generell mit geschweiften Klammern oder Anführungszeichen eingefaßt. Es gibt desweiteren einige Regeln, wie die Einträge in spezifischen Fällen auszusehen haben, jedoch führt auch diese Nichtbeachtung zu keiner Fehlermeldung, sondern nur zu schlecht formatierten Ausgaben.
Für Namen gibt es zwei Arten der Darstellung:

"Donald E. Knuth" oder "Knuth, Donald E."

Allerdings sind die beiden folgenden Darstellungen nicht äquivalent, da BIBTEX in der ersten Darstellung "Müller" als zweiten Vornamen interpretiert:

falsch:"Johannes Müller Lotze" richtig:"Müller Lotze, Johannes"

Um häufig benutzte Titel, Journale, etc. abzukürzen, können in der Literaturdatei Makros hinzugefügt werden:

```
@String{CACM    = ''Communications of the ACM''}
```

Die Groß- und Kleinschreibung des Makros ist unerheblich, so daß *CACM* und *cacm* die gleiche Bedeutung haben. Das selbe gilt für *@String*.

Folgende Schreibweisen sind somit äquivalent:

journal = "Communications of the ACM"
Journal = CACM
JOURNAL = {Communications of the ACM}
journal = Cacm

Eine ausführliche Einführung in BIBTEX findet sich im LATEX-Handbuch[7] oder in der Dokumentation BIBTEXing[11].

Kapitel 3

Anforderungsanalyse

Als Grundlage für die Gliederung dieses Kapitels dient das Buch *Objektorientierte Softwareentwicklung mit der Unified Modeling Language (UML)* [10].
In der Aufgabenbeschreibung in Kapitel 3.1 sollen die Aufgaben, die das System zu bewältigen hat, kurz beschrieben werden. Das Kapitel 3.2 Systemvoraussetzungen soll die vorhandenen Resourcen und Systeme darlegen. Die einzelnen Anwendungsfälle, die sich aus der Aufgabenbeschreibung ergeben, finden Sie in Kapitel 3.3. Hieraus lassen sich wiederum die ersten Klassen und einige Domänen identifizieren (Kapitel 3.4). Zum Schluß dieses Hauptkapitels werden im Kapitel 3.5 einige Algorithmen beschrieben, die die Grundlage für das Zusammenführen von BibTeX-Einträgen bilden.

3.1 Aufgabenbeschreibung

Das zu entwickelnde System soll den Benutzer bei dem Zusammenführen beziehungsweise Vergleichen zweier BibTeX-Dateien unterstützen. Hierbei sollen semantisch gleiche Einträge vom System erkannt und gegebenenfalls unter Intervention des Benutzers zusammengeführt werden. Die darauf aufbauenden Funktionen des Systems sind (vgl. Abbildung 3.1):

1. *Union*: In der Ergebnisdatei sollen alle Einträge enthalten sein, wobei die semantisch gleichen Einträge nur einmal vorkommen sollen.

2. *Intersection*: Nur Einträge, die in beiden Dateien vorhanden sind, sollen in die Ergebnisdatei übernommen werden.

3. *Minus*: Es sollen nur Einträge der ersten (zweiten) Datei in die Ergebnisdatei übernommen werden, die nicht in der zweiten (ersten) Datei enthalten sind.

4. *Check*: Aus der Datei selbst sollen semantische Duplikate herausgefiltert werden.

Abbildung 3.1: mögliche Ergebnismengen des Systems

Ist es dem System beim Vergleichen zweier Einträge nicht möglich, eine Entscheidung zu treffen, so soll der Benutzer intervenieren und die Entscheidung fällen. Hierbei soll er entweder entscheiden, daß keine Übereinstimmung vorhanden ist oder das Aussehen des zusammengeführten Eintrages bestimmen.

Um die semantisch gleichen Einträge zu finden, sollen Algorithmen des Information Retrieval genutzt werden. Eine allgemeine Einführung und Literatur zu Information Retrieval folgen in Kapitel 4.

3.2 Systemvoraussetzungen

Damit dies System möglichst plattformunabhängig benutzt werden kann, wird das System in das bestehende World-Wide-Web integriert und kann somit von jedem Rechner, der einen Internetzugang und einen WWW-Browser besitzt, genutzt werden. Allerdings ist die Voraussetzung für den WWW-Browser die Unterstützung des Standards *HTML 3.2* [12].

Als Programmiersprache wurde *Tycoon-2* gewählt. Zusätzlich wird die *Structured Tycoon Markup Language (STML)* zur dynamischen Generierung von HTML-Seiten genutzt, um einen interaktiven Zugriff auf die Informationen des Objektspeichers über den WWW-Browser zu erlangen.
Da *Tycoon-2* zu diesem Zeitpunkt nur für bestimmte Plattformen vorliegt, wird eine *SUN Ultra* mit dem Betriebssystem *Solaris 2.5.1* als Server dienen.

3.3 Anwendungsfallanalyse

Das *Use Case Diagramm* in Anhang C zeigt die möglichen Anwendungsfälle und Aktionen des Benutzers. In Tabelle 3.1 werden die einzelnen Objekte (Anwendungsfälle) und die Abhängigkeiten zwischen ihnen dargestellt und genauer beschrieben.
Um nun die einzelnen Anwendungsfälle auf konkrete Aktionen im System abzubilden, ist in Tabelle 3.2 eine Beschreibung der primären Aktionen zwischen dem Benutzer und dem System zu sehen.

Das Ablaufdiagramm in Abbildung 3.2 zeigt die in Tabelle 3.2 diskutierten Aktionen in einem zeitlichen Ablauf, da die einzelnen Vorbedingungen für bestimmte Aktionen innerhalb des Systems sonst nur schwer kenntlich zu machen sind.

3.4 Klassen und Domänen

In diesem Abschnitt werden Fachbegriffe, die sich während der Analysephase ergeben haben, näher erläutert und zu ersten Klassen ausgearbeitet. Ebenso werden die einzelnen Operationen auf die einzelnen Klassen abgebildet und eingebunden.
Zunächst ist der Begriff BibTeX-Eintrag sehr oft gefallen. Allgemein besteht ein BibTeX-Eintrag aus einem Eintragstyp, einem Schlüssel und einer Liste von Feldnamen und Werten (z.B.: author = "Leslie Lamport"). Eine BibTeX-Datei wiederum besitzt einen Dateinamen und eine Liste von BibTeX-Einträgen. Zusätzlich können in einer BibTeX-Datei benutzerdefinierte Makros enthalten sein, die aber bereits beim Importieren berücksichtigt werden sollen.
Die Parameter sind in diesem Kontext eine Liste von Parameternamen und Werten, wobei es systeminterne Parameternamen und Parameternamen aus BibTeX-Einträgen gibt. Genaueres zu diesen Parametern und ihrer Bedeutung folgt in Kapitel 5.

Die Begriffe BibTeX-Eintrag und BibTeX-Datei kann man bereits in ein Klassendiagramm abbilden, welches in Abbildung 3.3 zu finden ist. Zusätzlich sind in diesem Diagramm die Domänen der beiden Klassen zu finden.

Use Case	beteiligte Objekte	Beschreibung
Bibliographiedaten bearbeiten	Benutzer, Einträge vergleichen	*Der zentrale Use Case des Systems.* Bibliographiedaten sollen mit Algorithmen und Unterstützung des Benutzers bearbeitet werden.
Datenbestand erweitern	Bibliographiedaten bearbeiten	Eine der Funktionen, um Bibliographiedaten zu bearbeiten. Entspricht der Funktion *Union*.
Dateien formatieren	Exportieren, Importieren	Um die BibTeX-Dateien für das System lesbar zu machen, beziehungsweise Daten aus dem System in eine Datei zu übertragen, durchlaufen die BibTeX-Dateien einen Parser beziehungsweise Unparser.
Einträge vergleichen	Benutzer	Der Benutzer muß gegebenenfalls einzelne Einträge selbst vergleichen und Entscheidungen fällen.
Exportieren	Benutzer	Damit die Daten aus dem System in eine BibTeX-Datei geschrieben werden können, muß der Benutzer einen Dateinamen vergeben und die Datei einen Unparser durchlaufen.
Importieren	Benutzer	Der Benutzer gibt die Dateinamen der zu bearbeitenden Dateien ein, damit diese für das System sichtbar gemacht werden können.
Konsistentes Zusammenführen	Bibliographiedaten bearbeiten	Eine der Funktionen, um Bibliographiedaten zu bearbeiten. Entspricht der Funktion *Intersection*.
Parameter einstellen	Benutzer	Die Parameter sollen die Interaktionsmöglichkeiten zwischen dem System und dem Benutzer regeln und können somit durch den Benutzer verändert werden.
Unterschiede finden	Bibliographiedaten bearbeiten	Eine der Funktionen, um Bibliographiedaten zu bearbeiten. Entspricht der Funktion *Minus*.

Tabelle 3.1: Alphabetisch sortierte Auflistung und Erklärung der einzelnen Use Cases

Eingehendes Ereignis	Dateneingang (D), Vorbedingung (V)	Erwartetes Verhalten	Nachbedingung	Ausgehende Daten
Benutzer möchte Daten importieren	Dateiname (D)	Wenn die Datei in dem angegebenen Pfad existiert, wird die Datei ausgelesen.	Daten sind für das System sichtbar	-keine-
Benutzer möchte Daten exportieren	Dateiname (D)	Die vom Benutzer ausgewählten Daten werden in eine BibTeX-Datei geschrieben.	-keine-	Daten liegen im BibTeX-Format vor
Parameter einstellen	-keine-	Der Benutzer stellt nach seinen Wünschen die Parameter für das Bearbeiten der Bibliographiedaten ein.	Parameter geändert	-keine-
Bibliographiedaten anzeigen	Auswahl der Datei (D), Daten wurden importiert (V)	Die ausgewählte Datei wird dem Benutzer angezeigt.	-keine-	-keine-
Bibliographiedaten bearbeiten	Funktion wählen (D), Daten wurden importiert (V)	Die gewünschte Funktion wird vom System ausgeführt. Falls das System keine eigene Entscheidung treffen kann, so muß der Benutzer die Einträge vergleichen, wobei das System den Benutzer mit Informationen über die Einträge unterstützen soll.	-keine-	Distanzen einzelner BibTeX-Einträge, Endergebnis

Tabelle 3.2: Aktionen und Ereignisse des Systems

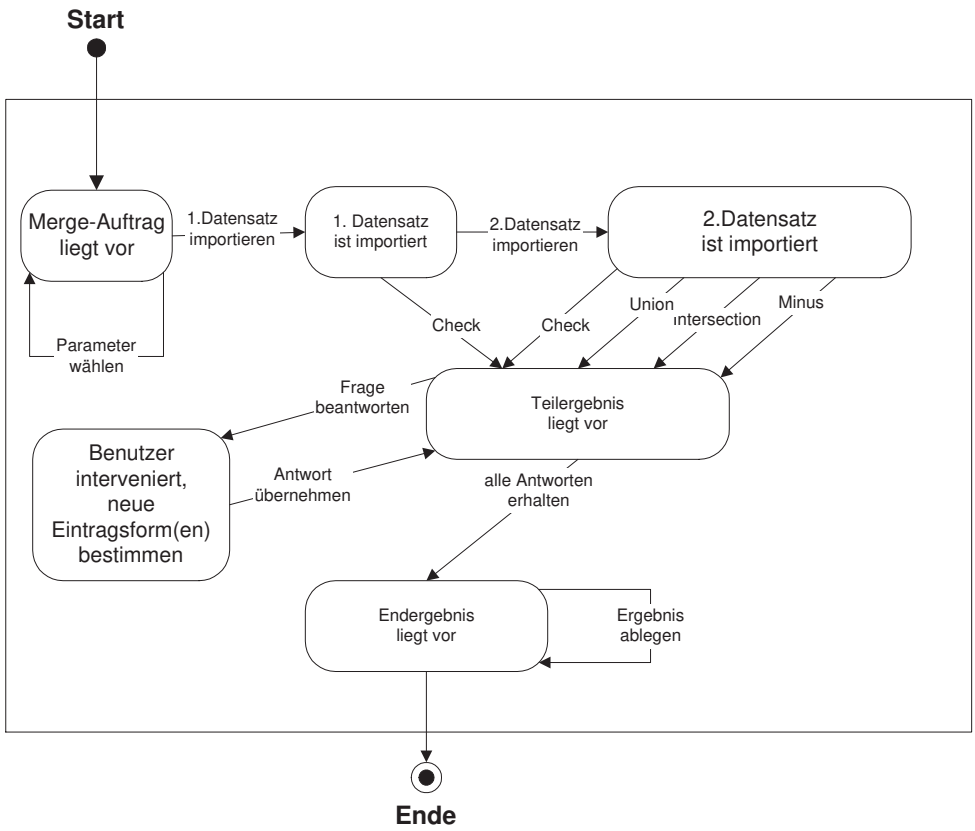

Abbildung 3.2: Ablaufdiagramm der Anforderungsanalyse

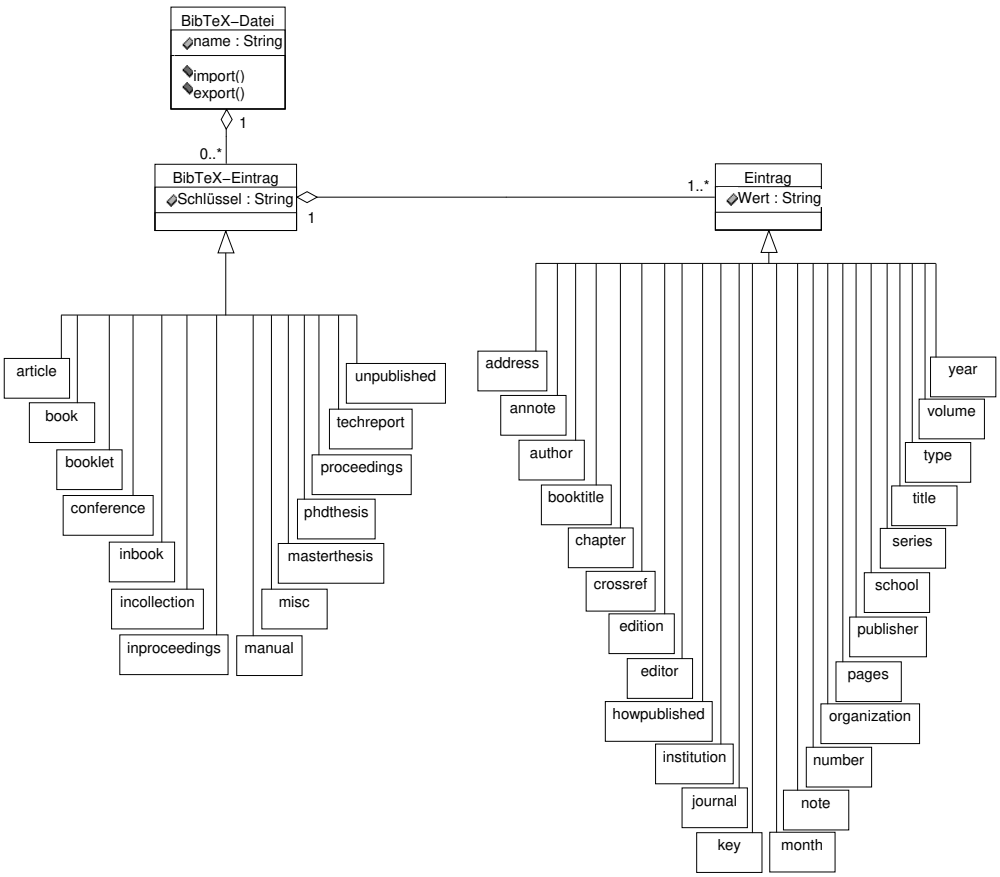

Abbildung 3.3: BibTeX-Domänen

3.5 Merging-Algorithmen

Die zentrale Aufgabe des System soll es sein, BibTeX-Einträge zu vergleichen und gebenenfalls zusammenzuführen. Dieses Kapitel bildet die Grundlage für diese beiden Operationen.
Folgende Fragen sollen in diesem Kapitel beantwortet werden:

- Wie sollen zwei Einträge verglichen werden und was soll das Ergebnis dieses Vergleiches sein?
- Wann soll der Benutzer eingreifen und wie sollen die erwähnten Parameter genutzt werden?
- Welche Hilfsfunktionen werden benötigt, um die gewünschten Funktionen durchführen zu können?
- Gibt es Hilfsfunktionen, die bei mehreren oder gar allen Funkionen genutzt werden können?

Seien $a, b, a_i, \ldots \in M$ Elemente einer beliebigen Menge M.

Definition 3.1 *probabilistische Distanz:*

$$0 \leq a \doteq b \leq 1 \textit{ ist eine probabilistische Distanz,}$$
$$\textit{wenn gilt: } (a \doteq a) = 1 \textit{ (Reflexivität),}$$
$$\textit{und } (a \doteq b) = (b \doteq a) \textit{ (Symmetrie),}$$
$$\textit{und } (a \doteq c) \leq (a \doteq b) + (b \doteq c) \textit{ (Monotonie).}$$

Die *probabilistische Distanz* $a \doteq b$ zwischen zwei Elementen a und b Element der Menge M, gibt als Ergebnis eine Wahrscheinlichkeit zurück, die die Ähnlichkeit der beiden Elemente charakterisiert.
Sei A eine Menge von Elementen, dann gilt:

$$\forall a_i, a_j, i \neq j : a_i \doteq a_j < 1.0$$

Sei R eine Menge von Tupeln $\{r_0, r_1, r_2, \ldots, r_n\}$ mit $r_i = (a, x)$ ($a \in A$ und $x = [0, 1]$)

Definition 3.2 *probabilistischer Elementabgleich:*

$$a \dot{\sqcap} \quad \emptyset \quad = \bot \tag{3.1}$$
$$a \dot{\sqcap} \quad \{(b, 1.0)\} \cup R \quad = a \tag{3.2}$$
$$a \dot{\sqcap} \quad \{(b_1, x_1), (b_2, x_2), \ldots, (b_n, x_n)\} \quad = \begin{cases} \bot \\ c(a, b_i) \end{cases} \tag{3.3}$$

$$a \dot{\sqcup} \quad \emptyset \quad = a \tag{3.4}$$
$$a \dot{\sqcup} \quad \{(b, 1.0)\} \cup R \quad = \bot \tag{3.5}$$
$$a \dot{\sqcup} \quad \{(b_1, x_1), (b_2, x_2), \ldots, (b_n, x_n)\} \quad = \begin{cases} \bot \\ a \end{cases} \tag{3.6}$$

Die Funktionen *cap*($\dot{\sqcap}$) und *cup*($\dot{\sqcup}$) arbeiten grundsätzlich auf einem Element vom Typ X und einer Liste von Elementen vom Typ Y.
In dieser Arbeit handelt es sich bei den Elementen des Typs X um einzelne BibTeX-Einträge. Die Elemente des Typs Y bestehen aus einem Tupel mit einem BibTeX-Eintrag und einer Wahrscheinlichkeit.
Ziel der Funktionen *cap* und *cup* ist es zu entscheiden, ob für den Elementabgleich eine weitere Funktion hinzugezogen werden muß oder ob durch die vorhandenen Angaben die Entscheidung über den Zusammenhang zweier Einträge getroffen werden kann.
Die Funktion *cap* ist eine Hilfsfunktion für die Funktion *Intersection* (Gleichung 3.7).

- Gleichung 3.1 beschreibt den Fall, daß das Element a mit keinem Element aus der Menge R übereinstimmt und nicht in die Ergebnismenge eingehen soll. Somit wird *bottom*($\bot$) zurückgegeben.

- Gleichung 3.2 beschreibt den Fall, daß die probabilistische Distanz zwischen a und einem Element b aus der Menge R 1.0 ist. Dieses Element a wird als Ergebnis zurückgegeben und geht bei *Intersection* in die Ergebnismenge ein.
- Gleichung 3.3 beschreibt den Fall, daß es dem System nicht möglich ist über die Gleichheit des Eintrages a und den Einträgen der Menge R zu entscheiden. Somit soll der Benutzer entscheiden, ob er $bottom(\perp)$ oder neues Element $c(a, b_i)$ zurückgibt.

Die Funktion *cup* ist wiederum die Hilfsfunktion der Funktion *Minus* (Gleichung 3.8).

- Gleichung 3.4 beschreibt den Fall, daß kein Element mit a übereinstimmt und a bei *Minus* in die Ergebnismenge eingeht.
- Gleichung 3.5 beschreibt den Fall, daß ein Element der Menge R mit einer Wahrscheinlichkeit von 1.0 mit a übereinstimmt. Bei *Minus* gehört a somit nicht in die Ergebnismenge.
- Gleichung 3.6 entspricht Gleichung 3.3, jedoch soll der Benutzer bei *Minus* nur entscheiden, ob a in die Ergebnismenge übernommen wird oder nicht.

Die Gleichungen 3.1, 3.2, 3.4 und 3.5 können vollständig vom System bearbeitet werden.

Definition 3.3 *Intersection:*

$$A\dot{\cap}B = \{(a\dot{\sqcap}\{(b, (a \doteq b)) \mid b \in B, a \doteq b \geq p\}) \mid a \in A\}, 0 \leq p \leq 1 \tag{3.7}$$

Diese Funktion spiegelt eine der Funktionen des Systems wider, die bereits in Kapitel 3.1 beschrieben wurde. Es werden zwei Mengen von Elementen verglichen und eine Schnittmenge zurückgegeben. Diese Schnittmenge soll mit Hilfe der Funktion *cap* erstellt werden, so daß alle semantischen Duplikate entfernt werden. Der Parameter p ist ein Schwellwert und soll vom Benutzer beeinflußt werden. Wenn p nahe an 1 liegt, so wird der Benutzer selten gefragt werden. (Bei p nahe bei 0 wird er häufig gefragt.)
Es gilt: $\mid A\dot{\cap}A \mid = \mid A \mid$

Definition 3.4 *Minus:*

$$A\dot{\setminus}B = \{(a\dot{\sqcup}\{(b, (a \doteq b)) \mid b \in B, a \doteq b \geq p\}) \mid a \in A\}, 0 \leq p \leq 1 \tag{3.8}$$

Definition 3.5 *Union:*

$$A\dot{\cup}B = (A\dot{\cap}B) \cup [A\dot{\setminus}(A\dot{\cap}B)] \cup [B\dot{\setminus}(B\dot{\cap}A)] \tag{3.9}$$

Genauso wie die Funktion *Intersection* entsprechen die Funktionen *Minus* und *Union* den zuvor beschriebenen Systemfunktionen, auf denen das System beruht.

Kapitel 4

Information Retrieval

Ein Teilgebiet des Information Retrieval ist der Bereich des Textretrieval, den N. Fuhr [4] als

> inhaltliche Suche in Texten

beschreibt.
Hierbei ist die Quelle der Informationen und Daten zum Beispiel eine Literaturdatenbank, in welcher der Anwender nach Quellen für seine Studien sucht und seine Anfrage in einer natürlichen Sprache eingibt. Das Problem an dieser Stelle ist, wie ein System diese Anfrage verstehen soll. Eine Möglichkeit ist, nach der gesamten Eingabezeichenkette in der Datenbank zu suchen, eine andere nach jedem einzelnen Wort der Anfrage. Doch schon hier erkennt man, daß wohl beide Methoden nur teilweise zu einem erfolgreichen und befriedigenden Ergebnis führen werden.
Zum einen ist die Wahrscheinlichkeit, daß bei einer längeren Anfragezeichenkette genau diese Konstellation der einzelnen Worte innerhalb des Textes eintritt, sehr gering, und die Anzahl der gefundenen Dokumente wird nicht sehr groß sein. Wenn die Anfragezeichenkette jedoch sehr kurz sein sollte, so werden auch Dokumente gefunden, die nur am Rande Informationen zum gesuchten Thema enthalten. Falls man sich sogar innerhalb der Anfrage vertippt haben sollte oder durch mangelndes Wissen ein Schlüsselwort der Anfrage falsch geschrieben hat, so wird man von der Antwort des Systems stark enttäuscht sein.
Bei der zweiten Methode treten ganz andere Probleme auf. Da man innerhalb der deutschen Sprache oftmals Wörter wie "und, der, die ‚das, ..." verwendet, wird man von einer Flut von Antworten überschwemmt werden, die überhaupt nichts mit der eigentlichen Themensuche zu tun hat. Des weiteren ist es sinnvoll, zu versuchen den Wortstamm der verbleibenden Wörter zu finden, um diesen in die Anfrage eingehen zu lassen. Somit wird bei einer Suchanfrage nach *Handbüchern* auch Einträge die das Wort *Handbuch* enthalten, als Ergebnis zurückgegeben.

Wenn man eine Anfrage gegenüber einem Information Retrieval System, wie oben beschrieben, stellt, so wird man keine exakte Antwort erhalten, sondern eine Liste von möglichen Ergebnissen bekommen. Hierbei ist es immer noch möglich, daß völlig irrelevante Dokumente gefunden wurden, die zufällig diese Schlüsselwörter enthielten, oder sogar die wichtigsten und besten Quellen überhaupt nicht als Ergebnis zurückgegeben werden.
An dieser Stelle wird der Unterschied zu klassischen Anfragen gegenüber Datenbank Systemen sichtbar, die sich ausschließlich mit dem sogenannten *Data Retrieval* beschäftigen. Einige Unterschiede zwischen dem klassischen Data Retrieval und dem Information Retrieval hat Rijsbergen [13] wie in Tabelle 4.1 dargestellt.

Den ersten Punkt der Tabelle, das *Matching*, habe ich bereits im Vorweg erläutert.
Bei DR-Systemen handelt es sich um deterministische Modelle, denn ein SQL-Statement wird zum Beispiel immer eine eindeutige Antwort zurückgeben, wobei das Ergebnis dieser Anfrage die Bedingungen vollständig erfüllen muß. Innerhalb eines IR-Systems jedoch kann eine Antwort zum Beispiel mit der Wahrscheinlichkeit 0,66 der Anfrage entsprechen.

	Data Retrieval (DR)	Information Retrieval (IR)
Matching	exakt	partiell, best match
Modell	deterministisch	probabilistisch
Anfragesprache	formal	natürlich
Fragespezifikation	vollständig	unvollständig
gesuchte Objekte	erfüllende	relevante
Reaktion auf Datenfehler	sensitiv	nicht sensitiv

Tabelle 4.1: Data Retrieval vs. Information Retrieval

Die *Anfragesprache* für DR-Systeme ist immer vorab durch eine Syntax definiert, wogegen die Anfrage gegenüber einem IR-System ein beliebiger Text beziehungsweise Zeichenkette sein kann. Ein Beispiel für eine Anfrage gegenüber einem DR-System könnte, wenn es SQL-Statements versteht, folgendermaßen aussehen:

select Fragen from Universum where Antwort = 42;

Bei einem IR-System könnte man diese Anfrage vielleicht wie folgt formulieren:

Suche alle Fragen des Universums, dessen Antwort "42" ist.

Daß die *Fragespezifikation* eines IR-Systems unbegrenzt ist, hängt mit der großen Variationsmöglichkeit innerhalb der natürlichen Sprachen zusammen, so daß diese *unvollständig* ist.

Der Punkt *gesuchte Objekte* hängt mit dem Matching zusammen, da die Antworten eines DR-Systems die Anfrage erfüllen müssen und die eines IR-Systems die relevanten Objekte enthalten sollen, auch wenn sie die Anfrage nicht genau erfüllen.
Daß die *Reaktion auf Datenfehler* in DR-Systemen sensitiv ist, hängt mit der formalen Anfragesprache zusammen. Wenn in der Anfrage gegenüber dem SQL-System ein Fehler vorliegt und statt 42 dort 44 steht, so wird ein *falsches* Ergebnis zurückkommen. Bei einem IR-Systems könnte sich trotzdem eine ähnliche Antwort ergeben, da diese von der Evaluation der "42" und deren Beachtung bei der Ergebnissuche abhängt.

4.1 String Matching Methoden

Die unterschiedlichen Algorithmen des String Matching lassen sich in zwei Kategorien aufteilen. Zum einen gibt es Algorithmen, die sich ausschließlich mit den einzelnen Zeichen des String (Zeichenkette) beschäftigen und die Bedeutung dieser Zeichen außer achtlassen. Zum anderen gibt es die Kategorie des phonetischen String Matching. Hierbei werden die einzelnen Buchstaben des Alphabets in Gruppen zusammengefaßt und den String in eine Zahlenkolonne überführt. Allerdings zählt man auch einige Algorithmen der ersten Kategorie zu den Algorithmen des phonetischen String Matching, da sich Worte, die ähnlich geschrieben sind, auch ähnlich anhören.
Zusätzlich kann man unterscheiden zwischen Algorithmen, die ein Ranking ermöglichen beziehungsweise nicht ermöglichen. Dies bedeutet, daß das System den einzelnen Ergebnissen eine zusätzliche Gewichtung beifügt und diese in eine Relation zueinander setzt.
Eine Zusammenfassung verschiedener Algorithmen finden Sie bei Zobel und Dart[16].

Aus beiden Kategorien finden Sie in Kapitel 4.1.1 und Kapitel 4.1.2 exemplarische Beispiele.

4.1.1 Edit Distance

Dieser Algorithmus errechnet die Distanz[1] zweier Strings zueinander, wobei man davon ausgeht, daß Strings, die sich ähnlich anhören, auch ähnlich geschrieben werden. Hierbei wird versucht,

[1] Diese Distanz ist nicht mit dem gebräuchlichemBegriff Distanzmaß innerhalb des IR zu verwechseln.

Code:	0	1	2	3	4	5	6
Buchstabe:	a e i o u y h w	b p f v	c g j k q s x z	d t	l	m n	r

Tabelle 4.2: Codierungstabelle für Soundex

einen der beiden Eingabestrings in den anderen zu überführen. Die verwendeten Operationen sind Einfügen, Löschen und Ersetzen einzelner Charakters.
Um die Distanz zweier Strings s und t mit den Längen m und n zu bestimmen, benutzt man die Funktion *edit* aus Abbildung 4.1 mit edit(m, n) auf. Die Hilfsfunktion r liefert den Wert 0, wenn s_i, wobei s_i das i-te Zeichen des Strings ist, und t_j gleich sind, sonst 1.
Die Distanz von *Telefon* und *Telephon* ist bei Edit Distance 2.
Durch die Rückgabe einer Zahl von Edit Distance erhält man zusätzlich die Möglichkeit, ein Ranking zu erstellen.

$$
\begin{aligned}
\text{edit}(0,0) &= 0 \\
\text{edit}(i,0) &= i \\
\text{edit}(0,j) &= j \\
\text{edit}(i,j) &= min[\ \text{edit}(i-1,j)+1, \\
&\qquad\quad \text{edit}(i,j-1)+1, \\
&\qquad\quad \text{edit}(i-1,j-1)+r(s_i,t_j)]
\end{aligned}
$$

Abbildung 4.1: Rekurrenzgleichung für Edit Distance

4.1.2 Soundex und Phonix

Soundex ist einer der phonetischen Matching Algorithmen. Der Eingabestring wird in eine Zahlenkolonne umgeformt, wobei das erste Zeichen unberührt bleibt. Die Tabelle für die Codierung befindet sich in Tabelle 4.2.
Der Algorithmus sieht folgendermaßen aus:

1. Ersetze alle Buchstaben außer den ersten durch seinen phonetischen Code.
2. Lösche alle Nachbarn, die gleich sind.
3. Lösche alle Nullen aus dem codierten Wort (Eliminierung der Vokale)
4. Gebe die ersten vier Zeichen zurück.

Der String *Telefon* wird zum Beispiel reduziert zu *T415* und *Telephon* ebenfalls zu *T415*. Allerdings werden auch Strings, die völlig unterschiedlich klingen und keine gemeinsame Bedeutung haben, als ähnlich erkannt.
Phonix ist eine Variation von Soundex. Die Kodierungstabelle ist leicht verändert, aber der entscheidende Unterschied ist, daß zu Beginn des Algorithmus einzelne Gruppen von Buchstaben umgesetzt werden. Zum Beispiel wird *tjV*, wobei V für einen beliebigen Vokal steht, zu *chV* umgesetzt, wenn *tj* am Beginn eines Strings steht. Es gibt insgesamt 160 dieser Gruppen von Buchstaben, die vorab ersetzt werden.

Die Möglichkeit des Rankings entfällt bei beiden Algorithmen, da die Antwort von Soundex und Phonix nur "ähnlich" oder "nicht ähnlich" ist.

4.2 Bewertung von IR-Systemen

Um die Antworten eines IR-Systems zu bewerten und somit seine Qualität zu beurteilen, muß zuerst unter folgenden Größen unterschieden werden (vgl. Abbildung 4.2):

- Menge aller Objekten (ALL)
- Menge aller relevanten Objekte (REL)
- Menge aller gefundenen Objekte (GEF)

Die Menge der relevanten Objekte muß bestimmt werden, indem sich der Benutzer alle Dokumente selbst anschaut und für jedes einzelne entscheidet, ob es relevant ist oder nicht. Ist dies möglich, so bräuchten wir gar kein IR-System.

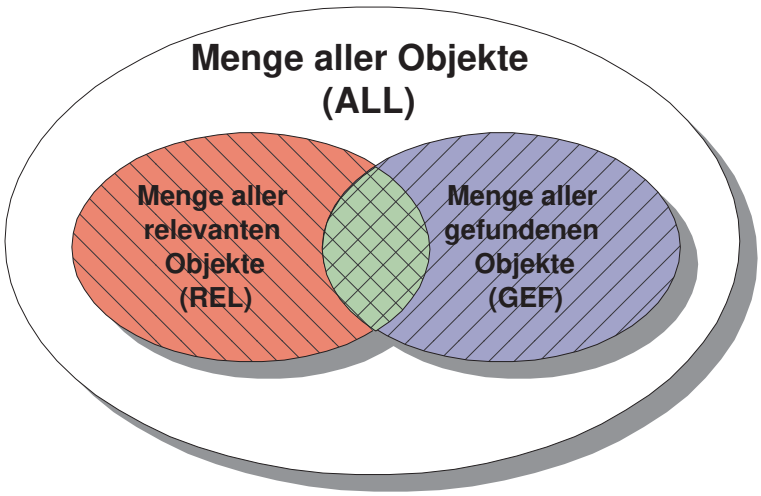

Abbildung 4.2: Zusammenhang zwischen relevanten und gefundenen Objekten in IR-Systemen

Es ergeben sich zusätzlich die folgenden drei Gleichungen, die zur Beurteilung von IR-Systemen beitragen:

- Precision: $p = \frac{|\mathrm{REL} \cap \mathrm{GEF}|}{|\mathrm{GEF}|}$
- Recall: $r = \frac{|\mathrm{REL} \cap \mathrm{GEF}|}{|\mathrm{REL}|}$
- Fallout: $f = \frac{|\mathrm{GEF} - \mathrm{REL}|}{|\mathrm{ALL} - \mathrm{REL}|}$

Precision ist hierbei der Anteil der gefundenen *und* relevanten Objekte an allen gefundenen Objekten. Recall wiederum bezeichnet den Anteil der tatsächlich gefundenen relevanten Objekte. Fallout gibt die Fähigkeit eines Systems, irrelevante Objekte von der Antwortmenge fernzuhalten, wieder.

Precision zu bestimmen ist einfach, da der Benutzer nur alle Objekte der Antwortmenge durchschauen muß und diejenigen dabei aussortiert, die er für irrelevant hält. *Recall* und *Fallout* setzen voraus, daß man die Menge aller relevanten Objekte kennt, so daß dieser Wert zumeist nicht bestimmbar ist, außer man hat einen stark beschränkten Datenbestand und somit die Möglichkeit alle Objekte einmal anzufassen.

Doch welche Möglichkeiten hat man, um einen Text für ein IR-System "verständlich" beziehungsweise "lesbar" zu machen? Hierbei spielen zahlreiche Methoden eine Rolle, von denen an dieser Stelle einige aufgeführt werden sollen.

4.2.1 Stoppwort-Elimination und Stammformreduktion

Stoppwörter haben generell keine Bedeutung für den Inhalt und die Information eines Textes und werden aus dem Text eliminiert. Hierbei handelt es sich vorrangig um Artikel, Füllwörter und Konjunktionen, deren Eliminierung einen Text um bis zu 50% schrumpfen lassen kann. Somit läßt sich der Text auf das Wesentliche reduzieren.
Die Stammformreduktion wirft wesentlich größere Probleme auf. Ein Beispiel aus der englischen Sprache für die Reduktion auf den Wortstamm ergibt sich bei der Endung *ual*. Bei *factual* gibt es keine Probleme und der Wortstamm ist mit *fact* lokalisiert. Wenn es sich jedoch um *equal* handelt und man die Endung *ual* entfernt, so ist der Sinn dieses Wortes verloren. Zumeist gibt man eine untere Schranke für die Länge nach der Reduktion an, so daß solche Probleme auf ein Minimum reduziert werden.

4.2.2 Indexing in IR-Systemen

Nach den zuvor beschriebenen Methoden erhalten wir einen Text, der nur noch die sogenannten *index terms* oder auch *keywords* enthält. Hieraus wird anschließend die sogenannte *index language* entwickelt, die ein Objekt repräsentieren soll.
Es werden zwei Arten von *index languages* unterschieden:

- pre-coordinated: Bei dieser Variante charakterisiert eine Kombination von *index terms* eine Klasse von Objekten. Dies geschieht während der Indexierung der Objekte zur späteren Verarbeitung.
- post-coordinated: Erst während der Anfrage an das IR-Systems wird eine Klasse von Objekten ermittelt, deren *index terms* zu der gestellten Suchanfrage passen.

Bis zu diesem Zeitpunkt haben alle *index terms* die gleiche Gewichtung, doch manche repräsentieren ein Objekt womöglich wesentlich besser als andere. Somit werden die einzelnen *index terms* noch einer Gewichtung unterzogen, bei der zwei Faktoren eine Rollen spielen:

1. Exhaustivity (Erschöpfung), gibt die Anzahl der Themen wieder, die ein *index term* enthält.
2. Specificity (Genauigkeit), gibt die Fähigkeit eines *index term* an, Themen des Objektes zu beschreiben.

Einige dieser beschriebenen Methoden finden im nachfolgenden Anwendungsfall ihre Berücksichtigung. Diese Studie der *University of Virginia* [3] beschäftigt sich mit der Suche in Literaturdatenbanken und wird im folgenden vorgestellt.

4.3 Eine Fallstudie

Grundlage für diese Studie ist die Literaturdatenbank *Astrophysics Data System*, in der versucht wird, Einträge einer Institution zu finden. Als Beispiel werden Artikel der

> University of Virginia, Charlottesville, Virginia, US

gesucht. Hierbei sind im Vorweg per Hand identifiziert, beziehungsweise durch menschliche Suche 21 Varianten und Kombinationen gefunden worden, um Einträge dieser Institution zu beschreiben. Die Variationen ergeben sich sowohl durch Abkürzungen von *University* und *Virginia* als auch durch die unterschiedliche Anordnung und das Auslassen der einzelnen Teileinträge. Beispiele hierfür sind:

- Univ. of Virginia, VA, US
- Virginia, University

Außerdem gibt es auch Rechtschreibfehler innerhalb der einzelnen Einträge, die die Suche noch weiter erschweren.
Um nun möglichst viele Einträge zu finden, die auf den Eingabestring passen, werden folgende Bearbeitungsschritte sequentiell durchgeführt:

1. *lexical cleanup*
   Da sich viele Variationen durch die unterschiedliche Handhabe von Abkürzungen und Akronymen ergeben, werden diese entweder durch ihre eigentliche Bedeutung (z.B.: VA durch Virginia) ersetzt oder vollständig entfernt. Um jedoch einen Informationsverlust weitestgehend zu vermeiden, beschränkt man sich hierbei auf folgende Vorgehensweise:

   (a) entferne US, U.S.A., etc. wenn diese am Ende vorkommen

   (b) entferne US ZIP codes vom Ende

   (c) entferne Abkürzungen für US-Bundesstaaten, die am Ende auftreten

   (d) expandiere alle gewöhnlichen Abkürzungen für University, Institute, etc.

   (e) expandiere anwendungsbezogene Abkürzungen und Akronyme

   (f) entferne alle Staaten-Namen, die am Ende vorkommen

   Diese Vorgehensweise ist allerdings nur auf diesen speziellen Kontext anzuwenden, da Angaben über Staaten in vielen Fällen wichtige Angaben und womöglich der einzige Hinweis sind. Somit müßte diese Vorgehensweise individuell auf die eigenen Algorithmen angepasst werden, um nicht fälschlicherweise Informationen zu verlieren. Allerdings ist das Expandieren von Abkürzungen zu Beginn eines solchen Algorithmus sinnvoll, da keine Informationen verloren gehen. Eine Ausnahme ist jedoch folgendes Szenario:

   > Die beiden Strings “cacm” und “cacn” sollen verglichen werden. (Im zweiten String liegt ein Tippfehler vor.) Nun wird zunächst versucht, alle Abkürzungen und Akronyme zu expandieren. Bei “cacm” verläuft dies erfolgreich, und wir erhalten “Communications of the ACM”. Für das System scheint es jedoch, als wenn für “cacn” keine Expansionsmöglichkeit besteht, und somit bleibt der String bestehen. Wenn man nun die beiden Strings vergleicht, so unterscheiden sie sich mehr voneinander, als vorher. Der Abstand der beiden Ausgangsstrings wird jedoch sehr gering sein, da sie sich nur an einer Stelle unterscheiden.

2. *Edit distance clustering*
   Nachdem die Abkürzungen und Akronyme behandelt wurden, werden die einzelnen Strings verglichen. Hierbei kommt der Algorithmus *Edit Distance* aus Kapitel 4.1 zum Zuge. Ab einem bestimmten Schwellwert wird nun angenommen, daß es sich um den gleichen String handelt. Anschließend gruppiert man die Strings nach Übereinstimmung und nimmt als repräsentatives Element jenes mit dem häufigsten Vorkommen in einer Gruppe. Die Wahl des Schwellwertes spielt in diesem Kontext eine wichtige Rolle, da bei einer falschen Wahl entweder semantisch gleiche Strings nicht erkannt oder unterschiedliche als gleich deklariert werden.

3. *Word-extraction and lexicographically sort*
   Zum Schluß werden aus dem String die enthaltenen Wörter extrahiert. Die entstehende Liste wird nun lexikographisch sortiert. Diese Änderung ergibt für diesen Kontext einen deutlichen Fortschritt, da sowohl “Virginia, University”, als auch “University, Virginia” gebräuchlich sind.
   Dieses Verfahren könnte man auch bei Namen anwenden, denn je nach Gebrauch wird die Reihenfolge von Vornamen und Nachnamen vertauscht.

Kapitel 5

Systementwurf

Basis der Implementation stellen Tycoon-2 und STML dar (vgl. Abschnitt 5.1 und 5.2). Das erstellte System verwaltet BibTeX-Einträge (vgl. Kapitel 2), stellt Algorithmen auf den BibTeX-Einträgen zur Verfügung (vgl. Kapitel 3.5) und bietet eine grafische Benutzeroberfläche (vgl. Abschnitt 5.3.3).

5.1 Tycoon-2

Bei Tycoon-2 [15] handelt es sich um den indirekten Nachfolger der Programmierumgebung Tycoon [9]. Tycoon-2 ist jedoch im Gegensatz zu Tycoon vollständig objektorientiert [14]. Gemeinsame Eigenschaften sind Persistenz und parametrischer Polymorphismus. Des weiteren erreicht Tycoon-2 eine Plattformunabhängigkeit, die mit der von JAVA [8] zu vergleichen ist.

In Tycoon-2 sind Klassen Objekte und verstehen somit Nachrichten, die über Methoden an sie geschickt werden. Diese Methoden befinden sich in den sogenannten Metaklassen. Will man also Klassenobjekten Nachrichten schicken, so sind die zugehörigen Methoden in der Metaklasse zu finden. Im allgemeinen werden Metaklassen mit der Endung *class* gekennzeichnet (vgl. Abbildung 5.1).

Um Typfehler und Meldungen wie *Method not understood* zu vermeiden, gibt es einen *Typechecker*, der bereits beim Übersetzen Fehler im Code erkennt. Dies beinhaltet, daß der Typ jedes Objektes bereits zur Übersetzungszeit feststeht. Somit zählt Tycoon-2 zu den strikt und statisch typisierten Systemen, ebenso wie Tycoon.

Im Unterschied zu objektorientierten Sprachen wie Smalltalk [6] und JAVA [2] verfügt Tycoon-2 über parametrischen Polymorphismus. Cardelli und Wegner [1] unterscheiden zwischen dem *universal* und dem *ad-hoc* Polymorphismus, die sich wiederum aufspalten lassen.
Tycoon-2 unterstützt jedoch keine der beiden Arten des *ad-hoc* Polymorphismus.[1] *Universal* Polymorphismus unterteilt sich in *parametric* und *inclusion* Polymorphismus. Die Besonderheit beim *inclusion* Polymorphismus ist, daß ein Objekt zu mehreren Klassen gehören kann, so daß ein Objekt sowohl der Klasse *Person*, also auch der Klasse *Student* angehören kann und ein Objekt alle

[1] Der *ad-hoc* Polymorphismus spaltet sich auf in *overloading* und *coercion* Polymorphismus. Allgemein kann es beim *ad-hoc* Polymorphismus eine Funktion geben, die auf unterschiedlichen Typen arbeitet und sich je nach Typ des Objektes auch völlig unterschiedlich verhält.
Sollten zwei Methoden mit dem gleichen Namen existieren, die aber auf Objekten unterschiedlichen Typs arbeiten, so kann beim *overloading* entschieden werden, welche Methode aufgerufen werden soll. Beim *coercion* wiederum wird eine semantische Operation benötigt, um den Typ eines Objektes zu verändern, damit er mit den erwarteten Typen einer Methode übereinstimmt.
Bei der Methode `+`, die in diesem Beispiel auf Objekten des Typs Real arbeiten soll, müßte beim Aufruf `5+6.0` das Objekt `5` umgesetzt werden in ein Objekt des Typs Real.

Methoden dieser beiden Klassen versteht.
Beim parametrischen Polymorphismus besitzen Klassen und Methoden einen Typparameter über den der Typ der Argumente festgelegt wird. Ein Beispiel für parametrischen Polymorphismus ergibt sich, wenn man die parametrisierte Klasse *MutableList* aus Tycoon-2 benutzt (ein Objekt der Klasse MutableList entspricht einer Liste im imperativen Sinne). Wenn zwei Objekte dieser Klasse erzeugt werden, eins mit dem Parameter *Int* und eins mit dem Parameter *String*, so sind die Methoden, mit denen auf diesen Objekten gearbeitet wird, gleich, aber die Objekte können nicht aufeinander zugewiesen werden. In Abbildung 5.1 finden Sie einen Ausschnitt des Quelltextes der Klasse *MutableList* und seiner Metaklasse *MutableListClass*.

```
class MutableListClass(E <:Object)
super AbstractListClass(E, MutableList(E))
metaclass MetaClass
public methods
new :MutableList(E)
  {
  EmptyList.new
  }
...
```

```
class MutableList(E <:Object)
super AbstractList(E,MutableList(E)), MutableSequence(E)
public methods
asList :List(E)
  {
  self
  }
...
```

Abbildung 5.1: Quelltext der Klassen *MutableListClass* und *MutableList*

5.2 Structured Tycoon Markup Language - STML

Bei der Structured Tycoon Markup Language (STML) handelt es sich genauso wie bei HTML um eine SGML. Das heißt, daß ein STML-Dokument einen SGML konformen Aufbau besitzt.
Eine vollständige Beschreibung der STML finden Sie bei Gawecki und Wienberg [5].
HTML biete keine Möglichkeiten, WWW-Seiten dynamisch zu erstellen, so daß in Abhängigkeit von Abfragen oder Zuständen bestimmter Objekte eine unterschiedliche Ausgabe entsteht. Die einzige Möglichkeit, dies zu erreichen, besteht in der Nutzung des Common Gateway Interface (CGI) und Skriptsprachen (z.B. JavaScript).

Um dynamische WWW-Seiten mit Tycoon-2 zu erstellen, hat man die Möglichkeit, über zusätzliche Steuerelemente HTML-Seiten zu erweitern (vgl. Abbildung 5.2). STML bietet folgende Erweiterungen zu HTML:

- `<if true='argument'>-<then>-<else>:`

  Entspricht dem *if-then-else*-Statement.

- `<define>-<ref>-<apply>:`

  Durch `<define>` besteht die Möglichkeit, Variablen mit Werten zu belegen oder Funktionen zu erzeugen. Mit `<ref>` greift man auf Variablen zu. Wenn man innerhalb des `<define>`

Elements mit `<fun>` eine Funktion erzeugt, so ruft man diese mit `<apply>` auf.

- `<send receiver='objekt' selector='method'>-<arg>:`

  Durch `<send>` kann man Objekten bestimmte Nachrichten schicken, die durch `<arg>` übergeben werden.

- *Backquotes*:
  Backquotes werden benutzt, um in HTML Attributen, wie zum Beispiel *IMG="..."* oder *TYPE="..."*, Tycoon-2 Ausdrücke einzubetten.

- `<tycoon>:`

  Innerhalb dieser Umgebung können vollständige Tycoon-2 Ausdrücke eingebettet werden. Hier können also sowohl Methodenaufrufe als auch Abfragen auftauchen.

Mit Hilfe des *Tycoon Web Server* ist es möglich, STML-Dokumente in HTML-Dokumente zu übersetzen. Hierzu wird zunächst das STML-Dokument auf Fehler überprüft. Zum einen wird die SGML-Konformität und zum anderen der enthaltene Tycoon-2 Code mit dem Typechecker überprüft.
Mit Hilfe einer Applikationsklasse wird dem *Tycoon Web Server* die Möglichkeit gegeben, auf ausgewählte Objekte der Anwendung zuzugreifen. Diese Applikationsklasse enthält insbesondere Methoden und Slots, die für die Anwendung relevant sind. Nähere Informationen finden sich bei Gawecki und Wienberg [5].

```
<!doctype stml system>
<stml processor=BibProcessor>
<TITLE>BibTeX-Merge-Tool</TITLE>
...
<BODY TEXT="#000000" BGCOLOR="#FFFFFF">
...
 <FORM ACTION="question_end.stml">
  <define name=output><fun param='zeile :BibTeXArray'>
   <TR><TD BGCOLOR=''question.colourFirst'' VALIGN="MIDDLE">
   <tycoon>out << zeile[0]</tycoon> 
   ...
  </fun></define>
  <TABLE BORDER=1 WIDTH="100%" BGCOLOR="#CCCCCC">
   <TR>...</TR>
   <send receiver='table' selector='do'><arg><ref name=output></ref></send>
   ...
  <TABLE>
   <TR><TD ALIGN="CENTER" WIDTH="150">
    <if true='nextButtonAppears'>
     <B>n&auml;chster Eintrag</B><BR>
     <A HREF=''"question_main.stml?number="+next''>
     <IMG BORDER=0 ALT="next" SRC="/icons/down.gif"></A>
    <else> 
    </if>
    ...
</BODY>
```

Abbildung 5.2: Beispiel einer STML-Seite

5.3 Systemarchitektur

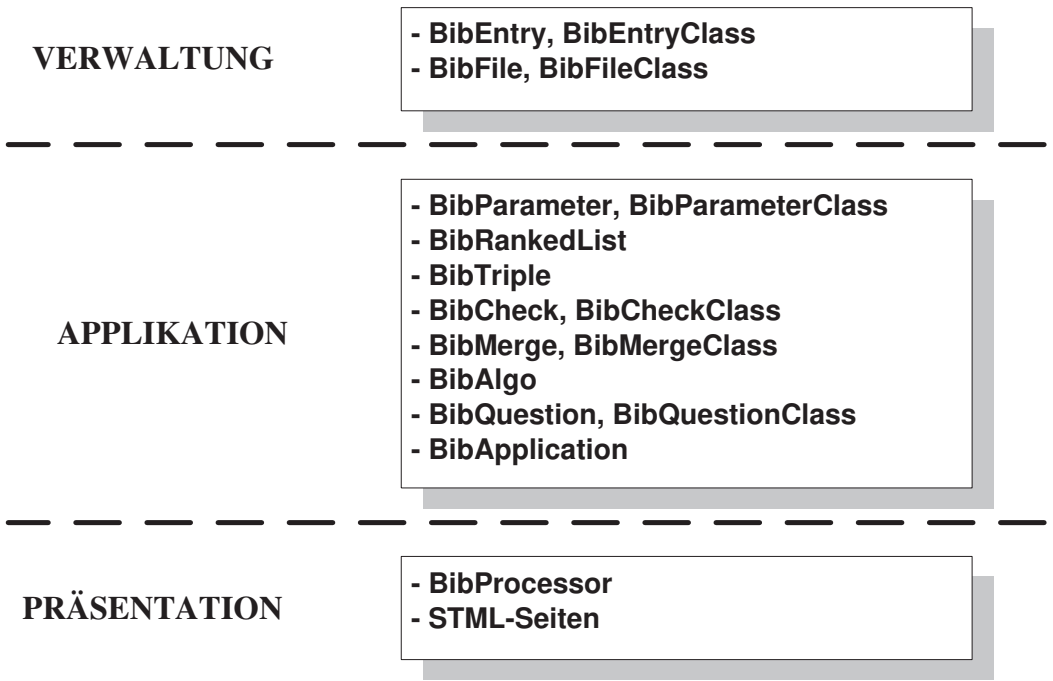

Abbildung 5.3: Schichtenmodell für die Systemarchitektur

In Abbildung 5.3 sehen Sie die Gliederung des Unterkapitels Systemarchitektur. Im Abschnitt Verwaltung (vgl. Kapitel 5.3.1) werden die Klassen beschrieben, die BibTeX-Dateien und BibTeX-Einträge repräsentieren. Das Kapitel 5.3.2 enthält die Klassen, die für die Verarbeitung der Daten zuständig sind. Im Kapitel 5.3.3 werden die Komponenten für die Präsentation beschrieben.

5.3.1 Verwaltung

Zur Verwaltung der Daten innerhalb des Systems dienen die Klassen *BibEntry*, die einen BibTeX-Eintrag, und *BibFile*, die eine BibTeX-Datei repräsentiert (vgl. Abbildung 5.4).

BibEntry

Eine Instanz der Klasse *BibEntry* stellt einen BibTeX-Eintrag dar.
Die Klasse *BibEntry* besitzt vier Slots. Die beiden öffentlichen Slots *type* und *key* entsprechen dem BibTeX-Eintragstyp und dem BibTeX-Schlüssel. In Abbildung 3.3 der Anforderungsanalyse sind Unterklassen für die einzelnen Eintragstypen zu finden. Jedoch ist es dem Benutzer möglich, eigene Eintragstypen zu definieren, so daß diese Klasse ständig erweitert werden müßte. Um dies zu verhindern, wurde die Implementation über den Slot *type* gewählt.
Der dritte öffentliche Slot *file* dient als Informationsquelle, zu welcher BibTeX-Datei ein Eintrag gehört, da die einzelnen Dateien in der Präsentation farblich unterschiedlich dargestellt werden. Zwar kann man auch über die Objekte der Klasse *BibFile* herausfinden, zu welcher BibTeX-Datei ein Eintrag gehört, doch aus Gründen der Performanz ist es sinnvoller, einem BibTeX-Eintrag diese Information mitzugeben.
Der private Slot *_dict* ist vom Typ *Dictionary(String,String)*, wobei ein Dictionary aus einer Menge von *(name, value)*-Paaren besteht. Dieses Dictionary enthält die einzelnen BibTeX-Felder des

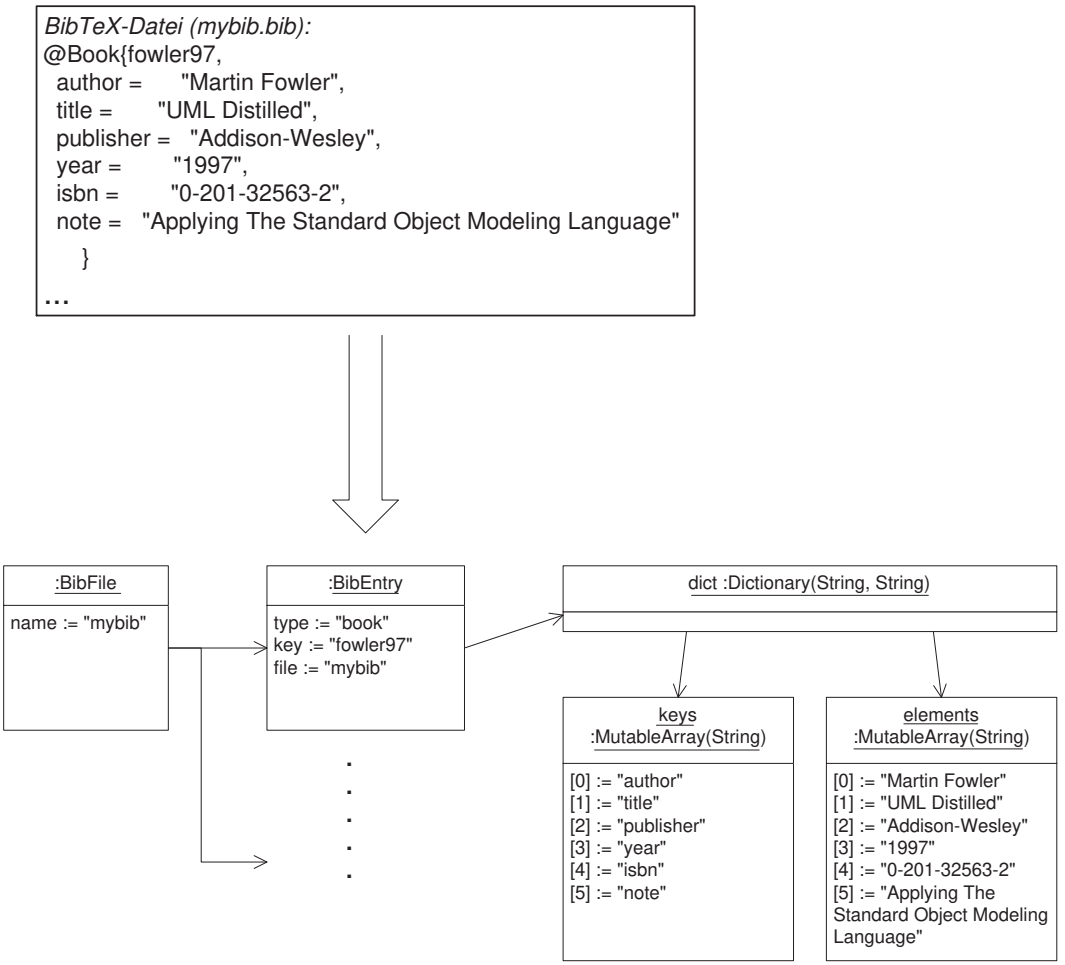

Abbildung 5.4: Objekte einer BibTeX-Datei – Instanzendiagramm

Eintrags und kann mit den Methoden der Klasse bearbeitet werden. Die einzelnen Methoden dieser Klasse delegieren die Nachrichten an die Methoden der Klasse *Dictionary* weiter.
Auf Grund dieser Definition sind die einzelnen BibTeX-Felder Objekte der Klasse *String*. Prinzipiell wäre es zum Beispiel auch möglich, eine Klasse *author* und *title* zu bilden. Jedoch können BibTeX-Datenbanken beliebige Feldnamen enthalten, so daß das System für jeden Benutzer gegebenenfalls erweitert werden müßte. Da dies aber nicht sinnvoll erscheint, wird an dieser Stelle auf die Unterscheidung der einzelnen Feldtypen verzichtet und erst beim Vergleichen zweier Einträge diese Unterscheidung herbeigeführt.
Die Methode *keysAndElementsDoSorted* ist eine Erweiterung der Methode *keysAndElementsDo* (sequentielle Bearbeitung von einem *Dictionary*), wobei die einzelnen Felder alphabetisch sortiert abgearbeitet werden. Dies ist für die Präsentation wichtig, damit die Ausgabe der einzelnen Felder eines BibTeX-Eintrags nicht durch die Speicherung innerhalb der Hash-Tabellen beeinflußt wird, sondern diese alphabetisch sortiert ausgegeben werden können.
Die zugehörige Metaklasse *BibEntryClass* enthält zum einen die Methode *new*, die als Eingabe keine Parameter erwartet, und zum anderen *newWith*, bei der der Typ, Schlüssel und das zugehörige File als Parameter beim Aufruf übergeben werden.

BibFile

Ein *BibFile* entspricht einer BibTeX-Datei mit Einträgen (*BibEntry*).
Der öffentliche Slot *name* speichert den Namen des entsprechenden Objektes. Dieser entspricht im allgemeinen dem Dateinamen. Der private Slot *list* ist vom Typ *List(BibEntry)* und entspricht der Aggregation aus Abbildung 3.3. Somit besteht eine BibTeX-Datei für das System aus einem Namen und einer Liste von BibTeX-Einträgen.
In einer BibTeX-Datei könnten allerdings auch noch Makros enthalten sein. Doch diese werden beim Importieren bereits auf die einzelnen BibTeX-Einträge angewendet und müssen somit nicht gespeichert werden.

1. **Importieren von Daten (*import(fileName :String) :Bool*)**
   Die Methode *import* liest eine BibTeX-Datei ein. Hierzu werden folgende Schritte durchgeführt:

   (a) Der Slot *name* des Objektes wird mit dem mitgelieferten Parameter *fileName* belegt.

   (b) Das Programm BibTeX wird mit der *aux*-Datei, die von der Methode selbst erzeugt wird, aufgerufen (Abbildung 5.5). Innerhalb der *aux*-Datei wird die BibTeX-Stildatei *merge* verwendet (vgl. Anhang B). Der Parameter für *bibdata* wird durch den Parameter *fileName* der *import*-Methode ersetzt.
   Durch den Aufruf von BibTeX entsteht eine temporäre Datei *tmp.bbl*, die in BNF-Notation folgendes Aussehen hat:

   {bibtex_type:*type*@%key_of_entry: *key*{@%*field_name*:*field_value*}@%}

   Durch den Aufruf des Programms BibTeX werden alle Feldnamen (z.B. *ISBN, Author*) in Kleinbuchstaben übersetzt und alle Makros aus der Stildatei und der BibTeX-Datei expandiert. Falls bestimmte Felder und Eintragstypen nicht in der Stildatei enthalten sind, so muß diese erweitert werden, da diese Felder sonst verloren gehen.

   (c) Die temporäre Datei *tmp.bbl* wird gelesen und mit Hilfe eines Scanners, der den regulären Ausdruck, den man aus der obigen BNF-Notation erhält, für die Datei kennt, werden die einzelnen BibTeX-Einträge als Elemente in den Slot *_list* des Objektes eingefügt.

2. **Exportieren von Daten (*export(:String) :Bool*)**
   Hierzu dient die Methode *export*, die die in dem Objekt enthaltenen Daten in eine Datei schreibt. Als Parameter erhält diese Methode den Dateinamen, der gegebenenfalls von dem

```
\relax
\citation{*}
\bibstyle{merge}
\bibdata{''fileName''}
```

Abbildung 5.5: *aux*-Datei zum Importieren der BibTeX-Dateien

im Slot *name* gespeicherten Wert abweichen kann.
Die einzelnen BibTeX-Einträge, die in dem Slot *_list* gespeichert sind, werden zum Exportieren vorab alphabetisch sortiert und dann in die Datei geschrieben. Das Aussehen der einzelnen BibTeX-Einträge entspricht dem in Kapitel 1 beschriebenen BibTeX-Format.

3. **Integritätsprüfung eines *BibFile*-Objektes (*checkIntegrity() :BibEntry*)**
   Da beim Mergen der Dateien nicht auf die Schlüssel der einzelnen BibTeX-Einträge geachtet wird, ist es möglich, daß mehrere verschiedene BibTeX-Einträge den gleichen Schlüssel erhalten haben. Um diesen Mangel zu beseitigen, wird dem entsprechenden Objekt, das beim Mergen entstanden ist, die Nachricht *checkIntegrity* geschickt. Diese parameterlose Methode liefert entweder *nil* oder einen BibTeX-Eintrag zurück. Wird *nil* zurückgegeben, so liegen keine Integritätsprobleme vor. Wird jedoch ein BibTeX-Eintrag zurückgegeben, so sollte man anschließend diesem Objekt der Klasse *BibEntry* die Nachricht *changeKey(:String, :BibEntry)* schicken. Die Parameter sind der BibTeX-Eintrag und der neue Eintragsschlüssel für dieses Objekt.

5.3.2 Applikation

Für die Verarbeitung der Daten, die in den Objekten der Klassen *BibFile* und *BibEntry* enthalten sind, dienen die Klassen, die in diesem Unterkapitel beschrieben werden. Sie übernehmen das Mergen von BibTeX-Datensätzen und das Vergleichen einzelner Einträge.

BibParameter

Die Klasse BibParameter ist eine einfache Struktur und enthält nur den privaten Slot *_param*, der sich von der parametrisierten Klasse *Dictionary* ableitet und vom Typ *Dictionary(String,Real)* ist. Allerdings spielen die Objekte dieser Klasse eine wichtige Rolle beim Vergleichen zweier BibTeX-Einträge.
Innerhalb des Systems gibt es folgende vordefinierte Parameter:

- *Upper_CUT*: Falls die Wahrscheinlichkeit (im Folgenden mit p bezeichnet), mit der zwei BibTeX-Einträge übereinstimmen, größer als der *Upper_CUT* ist, so nimmt das System automatisch an, daß diese beiden BibTeX-Einträge gleich sind.
  Dieser Wert sollte nahe bei 1 liegen, da ein zu niedriger *Upper_CUT* dazu führt, daß Einträge als gleich angenommen werden, die dies möglicherweise gar nicht sind.

- *CUT_Point*: Wenn gilt $CUT_Point < p < Upper_CUT$, so stellt dies für das System eine nicht lösbare Aufgabe dar, und der Benutzer muß intervenieren, um die BibTeX-Einträge zu vergleichen.
  Der *CUT_Point* muß auf jeden Fall kleiner sein als der *Upper_CUT*. Sollte der Benutzer jedoch versuchen dies zu ändern, so berücksichtigt das System zunächst den *CUT_Point* und benutzt anschließend den *Upper_CUT*, um die Ergebnismenge weiterzuverarbeiten.
  Ein Wert zwischen 0,6 und 0,8 hat sich für diesen Parameter als günstig erwiesen. Sollten die Ähnlichkeitsalgorithmen noch einmal verfeinert werden, so könnte man diesen Wert weiter erhöhen.

Durch diese Parameter erhält der Benutzer die Möglichkeit, in die Systemabläufe einzugreifen. Setzt er zum Beispiel den *CUT_Point* sehr hoch, so wird der Benutzer bei den gleichen Dateien seltener intervenieren müssen, als wenn der *CUT_Point* sehr niedrig gewählt wurde.

Zusätzlich gibt es noch zwei Parameter, die den Informationsgehalt des Eintragstyps und des Eintragsschlüssels widerspiegeln. Diese beiden Parameter sind vordefiniert, da jeder BibTeX-Eintrag diese Informationen enthält. Hierbei bedeutet ein Wert nahe bei 1, daß beim Vergleichen zweier BibTeX-Einträge dieses Feld besonders stark gewichtet werden soll. Liegt der Wert nahe bei 0, so gilt das entsprechende Gegenteil. Die systeminternen Bezeichnungen für diese Parameter sind *BibTeX_Typ* und *BibTeX_Key*. Für den Parameter *BibTeX_Typ* scheint ein Wert um 0,4 sinnvoll. Bei dem Parameter *BibTeX_Key* sollte man jedoch einen Wert nahe bei oder gleich 0 wählen, da im allgemeinen jeder Benutzer seine eigenen Konventionen besitzt, um seine Eintragsschlüssel sinnvoll zu belegen.
Des weiteren werden nach dem Importieren einer BibTeX-Datei alle Datensätze auf neue Eintragsfelder durchsucht und diese zu dem Slot *_param* hinzugefügt. Somit sind nach dem Importieren zweier BibTeX-Dateien alle Eintragsfelder, die mindestens einmal in einer der beiden Dateien vorkamen, genau einem Parameter im Slot *_param* zugeordnet. Die Werte dieser Parameter haben die gleiche Bedeutung, wie die des *BibTeX_Typ* und *BibTeX_Key*. Also bedeutet auch hier ein Wert nahe bei 1, daß dieses Feld stark gewichtet werden soll.

BibCheck

Die Methoden und Slots dieser Klasse und ihrer Metaklasse *BibCheckClass* unterstützen die Funktion *Check* (vgl. Kapitel 3.1).

Die Methoden der Metaklasse *BibCheckClass*:

- *_compareBibFile(:BibFile, :BibParameter) :MutableList(BibRankedList)*
  Innerhalb dieser Methode werden die einzelnen Objekte der Klasse *BibEntry* mit Hilfe der Methode *_compareBibEntries* wechselseitig verglichen.

- *_compareBibEntries(:BibEntry, :BibEntry, :BibParameter) :BibTriple*
  Aus dem mitgelieferten Objekt der Klasse BibParameter werden die nötigen Parameter herausgezogen. Hierzu zählen zunächst die vordefinierten Parameter *BibTeX_Key* und *BibTeX_Typ*. Zusätzlich werden Eintragsfelder gesucht, die in beiden BibTeX-Einträgen vorkommen. Zu diesen Eintragsfeldern werden die entsprechenden Parameter ausgelesen und gespeichert. Anschließend werden die einzelnen Parameter normalisiert, damit anschließend die Summe der benötigten Parameter 1 ergibt.
  Nun werden die einzelnen Domänen des BibTeX-Eintrags identifiziert und die Inhalte der BibTeX-Felder mit den entsprechenden Algorithmen verglichen. Abschließend können diese Ergebnisse mit dem entsprechenden Parameter multipliziert und aufsummiert werden. Ein Beispiel folgt bei der Beschreibung der Klasse *BibAlgo*.

Die Methoden der Klasse *BibCheck*:

- *cleanUp() :Void*
  Der Slot *_result* enthält nach Ausführung der *new*-Methode bereits ein Zwischenergebnis. Dieses Zwischenergebnis wird weiter bearbeitet, um die einzelnen BibTeX-Einträge entweder in den Slot *newBibFile* oder *questions* zu schreiben. Hierzu wird die Liste des Slots *_result* sequentiell abgearbeitet. Wenn man ein Objekt dieser Liste betrachtet, so gehört es der Klasse *BibRankedList* an. Also enthält es einen *BibEntry* und eine Liste von *BibTriple*. Sollte diese Liste leer sein, so konnte das System keinen Eintrag entdecken, der mit diesem Eintrag des Objektes übereinstimmt. Also kann dieser Eintrag in dem Slot *newBibFile* abgelegt werden. Sollte die Liste nicht leer sein und die Wahrscheinlichkeiten innerhalb der Objekte Klasse *BibTriple* kleiner sein als der *upperCut* und größer als der *cutPoint*, so wird der Eintrag in

dem Slot *questions* verschoben. Sollte eine Wahrscheinlichkeit größer als der *upperCut* sein, so wird nur der entsprechende Eintrag in *newBibFile* übernommen und der andere gelöscht. Sind alle Wahrscheinlichkeiten kleiner als der *cutPoint*, so wird der Eintrag in *newBibFile* abgelegt.

- *_addEmptyElementsFromBibFile(:MutableList(BibRankedList)) :MutableList(BibRankedList)* Durch das Verändern des Slots *questions* kann eine *RankedList* entstehen, dessen Felder zum Teil leer geworden sind, so daß die Elemente in den Slot *newBibFile* geschrieben werden können.

BibAlgo

Diese Klasse enthält alle Methoden, mit denen einzelne Felder der BibTeX-Einträge verglichen werden. Will man die Algorithmen verbessern, so hat dies in dieser Klasse zu geschehen. Falls man zusätzlich einzelne Domänen der BibTeX-Einträge noch weiter unterscheiden will, so muß man die *_compareBibEntries*-Methode der Klassen *BibCheckClass* und *BibMergeClass* ebenfalls erweitern.

Als kurzes Anwendungsbeispiel werden die beiden folgenden BibTeX-Einträge mit den implementierten Algorithmen verglichen. Es liegen zusätzlich die folgenden Parameter vor, wobei die Werte in Klammern bereits normalisiert sind:

Parameter	Wert
Upper_Cut	0.99
Cut_Point	0.7
BibTeX_Key	0.0 (0.0)
BibTeX_Typ	0.4 (0.129)
author	0.8 (0.258)
title	0.7 (0.226)
publisher	0.5 (0.161)
year	0.3 (0.097)
isbn	0.4 (0.129)
note	0.3

Erster Eintrag:

```
@Book{fowler97,
  author =      "Martin Fowler",
  title =       "UML Distilled",
  publisher =   "Addison-Wesley",
  year =        "1997",
  isbn =        "0-201-32563-2",
  note = "Applying The Standard Object Modeling Language"
      }
```

Zweiter Eintrag:

```
@Book{uml_fowler97,
  author =      "Fowler, Martin",
  title =       "UML-Distilled",
  publisher =   "Addison",
  year =        "1996",
  isbn =        "0-201-32563-2",
      }
```

In beiden Einträgen kommen die Felder *BibTeX_Key*, *BibTeX_Typ*, *author*, *title*, *publisher*, *year* und *isbn* vor. Somit sind nur diese Felder für die Normalisierung der Parameter relevant.
Die einzelnen Distanzen der Felder ergeben sich wie folgt:

- *BibTeX_Key* = 4/12 (Die Distanz der Strings wird über *Edit Distance* errechnet und durch die Länge dividiert.)
- *BibTeX_Typ* =0.0 (Die Eintragstypen sind gleich)
- *author* = 0.0 (Die beiden Felder werden vom System als gleich erkannt.)
- *title* = 1/13 (– siehe *BibTeX_Key* –)
- *publisher* = 7/15 (– siehe *BibTeX_Key* –)
- *year* = 0.5 (– siehe *distanceYear* im Anhang D –)
- *isbn* = 0.0

Nun ergibt sich folgende Gleichung:

$$d = 4/12 * 0.0 + 0.0 * 0.129 + 0.0 * 0.258 + 1/13 * 0.226 + 7/15 * 0.168 + 0.5 * 0.097 + 0.0 * 0.129 = 0.144$$

Daraus ergibt sich die Wahrscheinlichkeit $p = 1 - d = 0.856$.

5.3.3 Präsentation

Die Präsentation findet auf einem WWW-Browser statt. Somit müssen die Daten, die der Benutzer erhält, über die STML-Seiten aufgearbeitet werden.

BibProcessor

Bei jedem *Http-Request* wird ein neues Prozessor Objekt erzeugt, das unter anderem spezifische Daten der Anfrage und einen Verweis auf die Applikation enthält. Durch Festlegen der parametrisierten Superklasse *HtmlFormProcessor*, welche als Parameter stets die Applikationsklasse erhält, ist der Verweis auf die Applikation gegeben.
Der *BibProcessor* erbt vom *HtmlFormProcessor*, so daß auch die *Form-Felder* der HTML-Seiten beachtet werden. Zusätzlich enthält er die Methode *printReal*, damit Zahlen der Klasse *Real* nicht nur in wissenschaftlicher Notation, sondern auch als Gleitkommadarstellung ausgegeben werden können.

STML-Seiten

Einen Bildschirmausschnitt des Hauptmenüs finden Sie in Abbildung 5.6. Hierin enthalten ist auch das zentrale Mengendiagramm für die Applikation.

In diesem Mengendiagramm sind die folgenden vier Mengen enthalten:

- *A'* und *B'*:
  Die Menge *A'* (*B'*) enthält alle Elemente, die in dem Slot *newBibFileA* (*newBibFileB*) enthalten sind, also alle Elemente, zu denen es keinen semantisch gleichen Eintrag aus B (A) gibt.
- *AB*:
  Die Menge *AB* wiederum enthält die Einträge, zu denen es semantisch gleiche in beiden Mengen gibt. Diese Elemente sind in dem Slot *newBibFileAB* gespeichert.

- *?*:
  Hierin sind alle Elemente enthalten, die nicht eindeutig zugeordnet werden können. Durch die Intervention des Benutzers werden diese Elemente entweder nach *A'*, *B'* oder *AB* verschoben. Sollten zwei Einträge gleich sein, so wird entweder ein neuer Eintrag erstellt oder einer von beiden übernommen und der Eintrag in *AB* hinzugefügt. Anderenfalls wird der Eintrag je nach Zugehörigkeit in die Menge *A'* oder *B'* verschoben.

Wie aus Abbildung 5.6 zu entnehmen ist, wurden bereits BibTeX-Dateien importiert. Die jeweilige Anzahl der enthaltenen Elemente steht hinter dem Dateinamen in Klammern.
Auch die *Merge*-Funktion wurde über die entsprechende Funktion angestoßen, da den einzelnen Mengen bereits Elemente zugeordnet worden sind. Die Anzahl der Elemente, die in den einzelnen Mengen enthalten sind, stehen in der Mitte des Bildschirmausschnittes.
Die Funktion *"EDIT PARAMETER"* sollte vor den Funktionen *"MERGE"* und *"CHECK"* ausgeführt werden, damit die benutzerdefinierten Parameter bei beiden Operationen zum Tragen kommen.
Über die Funktion *"EDIT ?"* müßte der Benutzer nun die offenen Fragen beantworten, damit die derzeit 11 Elemente der Menge *?* in die entsprechenden Mengen verschoben werden können, beziehungsweise neue Einträge entstehen.
Über die Funktion *"ASSIGN TO C"* wird die entsprechende Funktion ausgewählt. Dies ist zwar auch möglich, wenn die Menge *?* nicht leer ist, doch der Benutzer muß bedenken, daß ihm Einträge verloren gehen, wenn er diese Operation ausführt, bevor er alle offenen Fragen beantwortet hat.
Hinter der Funktion *"CHECK"* verbirgt sich die entsprechende *Check*-Funktion des Systems. Je nachdem bei welcher Menge die Funktion aufgerufen wird, wird die Methode *Check* der Applikation mit dem richtigen Parameter aufgerufen. Die Funktion *"KEYS"* wiederum stößt die Methode *checkIntegrity* an und überprüft die Ergebnismenge auf eventuelle Integritätsprobleme.

In Abbildung 5.7 sehen Sie den Bildschirmausschnitt, den der Benutzer erhält, wenn er zwei Einträge miteinander vergleichen soll.
In diesem Fall hat das System zwei Einträge gefunden, die mit dem Eintrag aus der ersten Spalte übereinstimmen könnten. Über die Funktion *nächster Eintrag* unter der Tabelle erhält der Benutzer den zweiten Eintrag.
Die Wahrscheinlichkeit $p = 0.97065$ ist der vom System errechnete Wert.
Über die Funktion "->" hat der Benutzer die Möglichkeit, den Feldinhalt direkt in das entsprechende Formfeld in der rechten Spalte zu übertragen. Hierzu verbirgt sich hinter der Funktion eine *JavaScript*-Methode.
Sollte die gerade angezeigten Einträge übereinstimmen, so muß der Benutzer den neuen Eintrag erstellen und mit *"YES"* bestätigen. Stimmen keine der vorgeschlagenen Einträge überein, so verneint er entsprechend über die *"NO"*-Funktion.

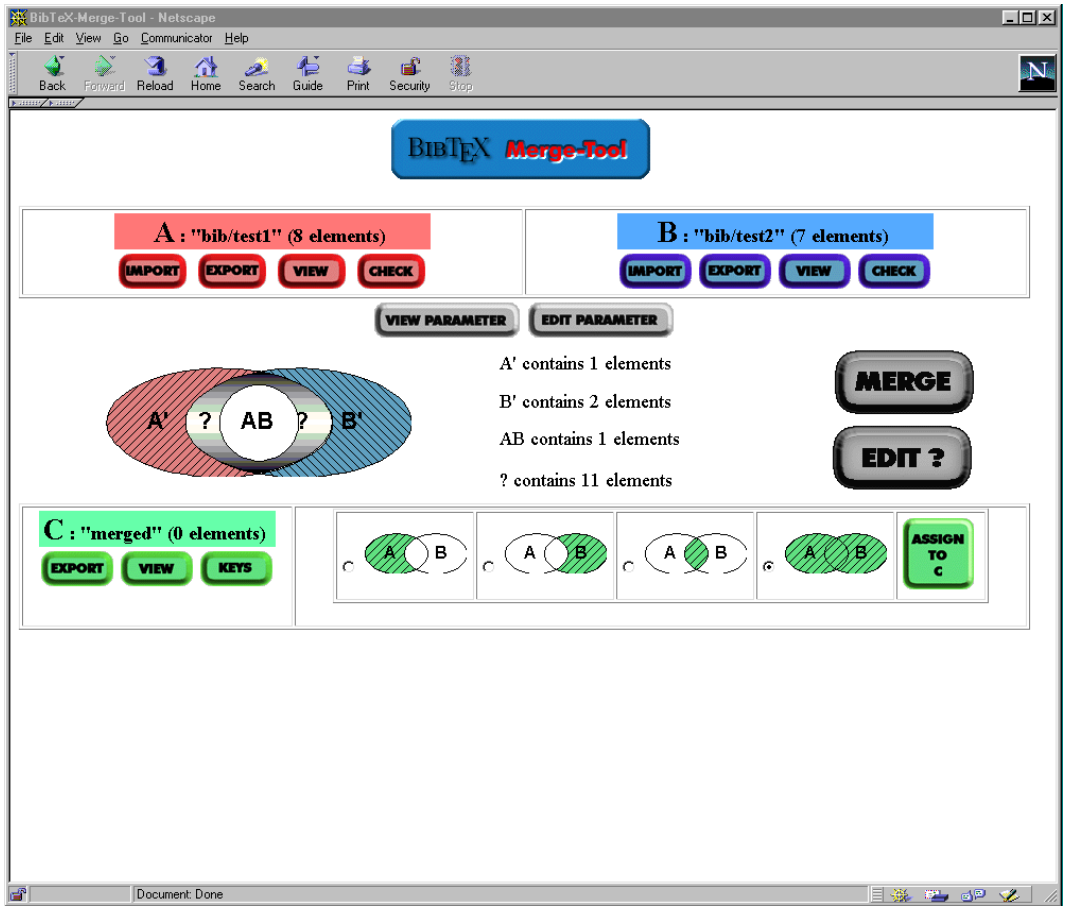

Abbildung 5.6: Bildschirmausschnitt - Hauptmenü des BIBTEX Merge-Tool

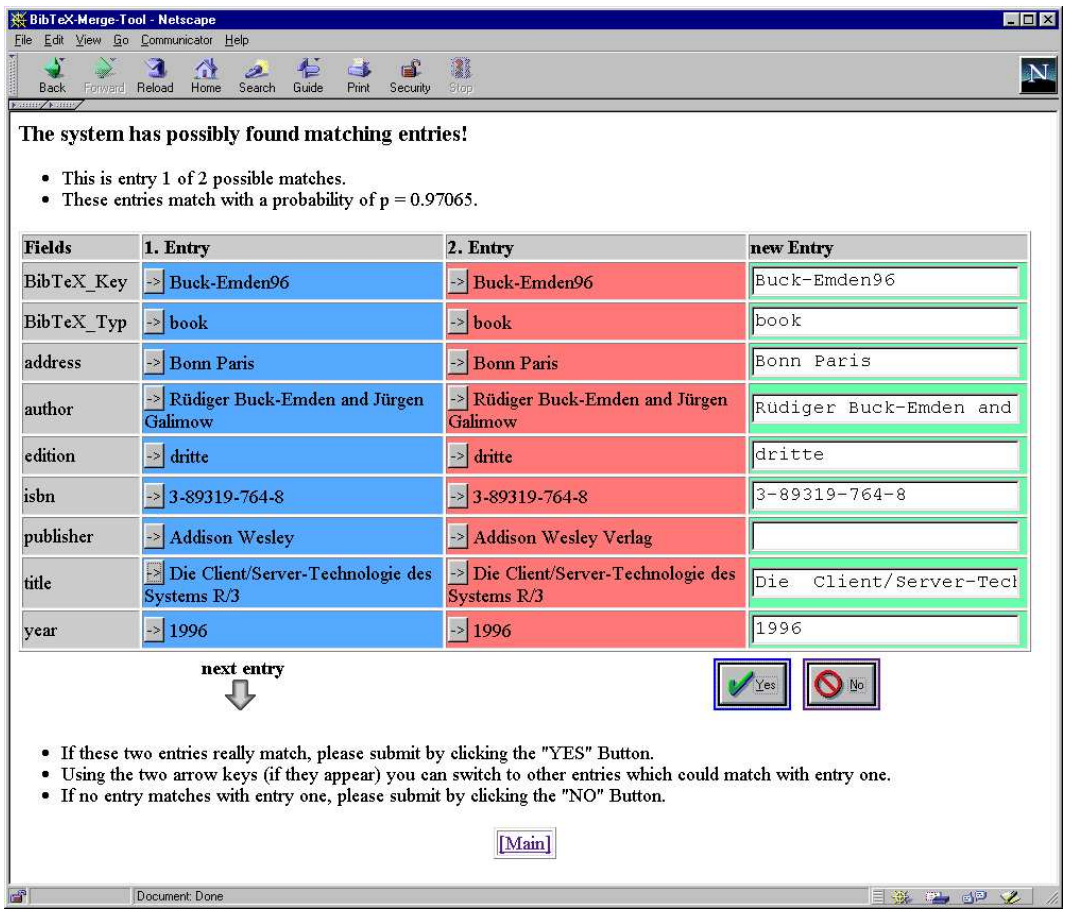

Abbildung 5.7: Bildschirmausschnitt - Vergleichen zweier BibTeX-Einträge

Kapitel 6

Zusammenfassung

Ausgangspunkt dieser Arbeit bildet die Aufgabenstellung, mehrere BibTeX-Datenbanken zu vergleichen und zusammenzuführen. In der Anforderungsanalyse werden die domänenspezifischen Anforderungen für das Zusammenführen von BibTeX-Einträgen analysiert. Insbesondere finden Methoden des Information Retrieval ihre Anwendung, um die Ähnlichkeiten von BibTeX-Einträgen festzustellen. Die Implementierung eines Prototypen greift auf Tycoon-2 als Implementierungssprache und STML zur Visualisierung zurück.

Nach der Implementierung des ersten Prototypen ergaben sich einige neue Gesichtspunkte, so daß von dem Ablaufdiagramm (Abbildung 3.2) der Anforderungsanalyse abgewichen wurde. Das Ablaufdiagramm der zweiten Iteration finden Sie im Anhang C.
Zu diesem Entschluß führte die Tatsache, daß die Funktionen des Systems und seine Leistung bisher vor dem Benutzer total verborgen blieben. Die Möglichkeiten der qualitativen Analyse waren ebenso stark eingeschränkt. Zusätzlich sollte dem Benutzer die Möglichkeit gegeben werden, verschiedene Ergebnismengen zu erhalten, ohne die gesamte Prozedur zu wiederholen.
In dem Prototyp war es ebenso nicht möglich, während der Benutzerintervention abzubrechen, falls es dem Benutzer zu viele Fragen sind, die ihm das System stellt. Jetzt kann dies ohne Ergebnisverlust und mit der Option den Vorgang fortzusetzen durchgeführt werden.

Die Anforderungsanalyse und der Systementwurf beschränken sich in dieser Arbeit ausschließlich auf das in Kapitel 2 beschriebene BibTeX-System. Die in Kapitel 4 beschriebenen Algorithmen zur Ähnlichkeitsfindung lassen sich jedoch auf viele Datenbanken ausweiten.
Während der Anforderungsanalyse hat sich herausgestellt, daß man die speziellen Domänen eines Systems im Vorweg identifizieren muß, da es sonst für Information Retrieval Systeme nicht möglich ist, ausreichende Informationen aus den Daten zu ziehen. Es gibt zwar bereist kommerzielle Systeme, die versuchen, Information Retrieval auf bestehende Datenbanksysteme anzuwenden, doch diese gehen von sehr allgemeinen Annahmen aus, so daß auch hier eine Verbesserung durch Spezifikation der individuellen Domänen zu erwarten ist.

Während der Test-Phase des Systems haben die verwendeten Algorithmen, speziell der *Edit Distance*, sehr gute Ergebnisse gezeigt. Voraussetzung hierfür war jedoch, daß inbesondere die Felder eines BibTeX-Eintrags sinnvoll ausgefüllt sind. Sollten zum Beispiel Felder, dessen Inhalt dem Benutzer nicht bekannt ist, mit Fragezeichen ausgefüllt sein, so sinkt die vom System errechnete Wahrscheinlichkeit vehement. Dies könnte jedoch durch Erweiterung der Algorithmen umgangen werden. Felder vom Typ *ISBN* könnten zum Beispiel als eine Zahlenkolonne mit Trennzeichen interpretiert werden und somit könnte bei fehlerhafter Belegung eine Ausnahmebehandlung erfolgen.

Insgesamt sollte es sich bei dieser Anwendung um ein generisches System handeln, das stetig erweitert wird. Speziell die BibTeX-Felder der einzelnen Benutzer sollten ständig hinzugefügt werden. Somit können auch die Algorithmen erweitert werden, da auch die Domänen erweitert werden.

Um die BibTeX-Felder der einzelnen Benutzer zu ergänzen muß die Stildatei (Anhang B) erweitert werden, da das System durch diese Datei die Daten importiert.
Um die Algorithmen zu erweitern beziehungsweise zu verfeinern muß in der Klasse *BibAlgo* die entsprechende Änderung vorgenommen werden. Falls neue Domänen und Eintragstypen hinzukommen, so müssen diese zusätzlich in der Methode *_compareBibEntries* der Klassen *BibCheck* und *BibMerge* bekanntgemacht werden.

Für mich persönlich war es das erste Projekt in der objektorientierten Welt, so daß es im Nachhinein scheint, daß man durch eine ausführlichere und tiefgehendere Anforderungsanalyse bessere Ergebnisse erzielen könnte.

Kapitel 7

Ausblick

Während der Implementation und Test-Phase des Systems haben sich bereits Veränderungen gegenüber der Anforderungsanalyse ergeben, so daß ein Teil der ursprünglichen Ideen verworfen wurde.
Doch wie kann man dieses System noch erweitern?
Die Oberfläche kann durch die STML-Seiten weiter verfeinert werden. Auch die Anbindung eines JAVA-Applets ist denkbar. Sollen andere Systeme verwendet werden oder das System durch externe Systeme angestoßen werden, so müsste man die Oberfläche dahingehend erweitern.
Auch die Personalisierung von BibTeX-Daten ist denkbar, so daß der Benutzer selektiert, welche BibTeX-Einträge beim Mergen berücksichtigt werden sollen.
Des weiteren könnte man vom Benutzer zusätzliche Eingaben erwarten. Wenn der Benutzer die verwendeten Sprachen eingeben würde, so könnte man phonetische Algorithmen wie *Phonix* oder *Soundex* verwenden, die zusätzliche Ergebnisse liefern.

Innerhalb der Applikation könnte man die Algorithmen verbessern. Inbesondere die Performanz des Systems ist noch verbesserungsfähig, da der *Edit Distance* momentan sehr oft Verwendung findet und dieser eine mittlere Komplexität von $\Theta(nm)$ besitzt. Doch gerade dieser Algorithmus führt zu den guten Ergebnissen.
Auch eine Datenbank zur phonetischen Suche, die ständig erweitert werden könnte, kann eingebunden werden. Ebenso können Fingerprints, die den einzelnen BibTeX-Einträgen angehängt werden, helfen den Benutzer zu entlasten. Sollte ein Benutzer seinen Datenbestand stetig erweitern wollen, so würden Antworten, die schon einmal gemacht wurden, als bekannt erkannt werden und Fragen vom Benutzer ferngehalten.

Die Verwaltung der Daten könnte durch einen eigene Parser verbessert werden, der auch Makros mit importiert. Somit kann der Benutzer seine selbst definierten Makros weiterbenutzen.

Anhang A

BibTeX Eintragstypen

Die folgende Aufzählung enthält alle Eintragstypen und Felder, die in den standard Stildateien vordefiniert sind:

- article
  zwingend: `author, title, journal, year`
  optional: `volume, number, pages, month, note`
- book
  zwingend: `author` oder `editor, title, publisher, year`
  optional: `volume` oder `number, series, address, edition, month, note`
- booklet
  zwingend: `title`
  optional: `author, howpublished, address, month, year, note`
- conference
  entspricht *inproceedings*
- inbook
  zwingend: `author` oder `editor, title, chapter` und/oder `pages, publisher,year`
  optional: `volume` oder `number, series, type, address, edition, month, note`
- incollection
  zwingend: `author, title, booktitle, publisher, year`
  optional: `editor, volume` oder `number, series, type, chapter, pages, address, edition, month, note`
- inproceedings
  zwingend: `author, title, booktitle, year`
  optional: `editor, volume` oder `number, series, pages, address, month, organization, publisher, note`
- manual
  zwingend: `title`
  optional: `author, organization, address, edition, month, year, note`
- masterthesis
  zwingend: `author, title, school, year`
  optional: `type, address, month, note`
- misc
  zwingend: keine
  optional: `author, title, howpublished, month, year, note`

- phdthesis
  zwingend: `author, title, school, year`
  optional: `type, address, month, note`
- proceedings
  zwingend: `title, year`
  optional: `editor, volume` oder `number, series, address, month, organization, publisher, note`
- techreport
  zwingend: `author, title, institution, year`
  optional: `type, number, address, month, note`
- unpublished
  zwingend: `author, title, note`
  optional: `month, year`

Zusätzlich gibt es noch ein optionales `key`-Feld, welches zur Sortierung verwendet wird, doch keinen weiteren Regeln unterliegt! Dieses Feld steht in keinem Zusammenhang mit dem *Eintragsschlüssel*, wird jedoch oft mit diesem verwechselt.

Anhang B

BibTeX-Stildatei

```
%% *** BibTeX-MergeTool ***
%% Ausgabe: bibtex_type: (type)@%key_of_entry: (key){@%(field): (entry)}@&

ENTRY
  { address
    author
    booktitle
    chapter
    edition
    editor
    howpublished
    institution
    isbn
    journal
    key
    month
    note
    number
    organization
    pages
    price
    publisher
    school
    series
    title
    type
    volume
    year
  }
  {}
  { label }

FUNCTION {output}
{
 duplicate$ empty$
    'pop$
    'write$
  if$
  "@%" write$
```

```
}

FUNCTION {output.bibitem}
{
  "key_of_entry: " write$
  cite$ write$
  "@%" write$
  ""
}

FUNCTION {fin.entry}
{
duplicate$ empty$
    'pop$
    'write$
  if$
  "@&" write$
  newline$

}

% Ausgabe der Felder

FUNCTION {print.fields}
{
    output.bibitem
    "address: " write$ address output
    "author: " write$  author output
    "booktitle: " write$  booktitle output
    "chapter: " write$  chapter output
    "edition: " write$  edition output
    "editor: " write$  editor output
    "howpublished: " write$ howpublished output
    "institution: " write$ institution output
    "isbn: " write$  isbn output
    "journal: " write$ journal output
    "key: " write$ key output
    "month: " write$ month output
    "note: " write$  note output
    "number: " write$  number output
    "organization: " write$  organization output
    "pages: " write$  pages output
    "price: " write$  price output
    "publisher: " write$ publisher output
    "school: " write$  school output
    "series: " write$  series output
    "title: " write$  title output
    "type: " write$  type output
    "volume: " write$  volume output
    "year: " write$  year output
     fin.entry
}

FUNCTION {article} {"bibtex_type: article@%"  write$ print.fields}
```

```
FUNCTION {book} {"bibtex_type: book@%"  write$ print.fields}
FUNCTION {booklet} {"bibtex_type: booklet@%" write$ print.fields}
FUNCTION {conference} {"bibtex_type: conference@%" write$ print.fields}
FUNCTION {inbook} {"bibtex_type: inbook@%" write$ print.fields}
FUNCTION {incollection} {"bibtex_type: incollection@%" write$ print.fields}
FUNCTION {inproceedings} {"bibtex_type: inproceedings@%" write$ print.fields}
FUNCTION {manual} {"bibtex_type: manual@%" write$ print.fields}
FUNCTION {masterthesis} {"bibtex_type: masterthesis@%" write$ print.fields}
FUNCTION {misc} {"bibtex_type: misc@%" write$ print.fields}
FUNCTION {phdthesis} {"bibtex_type: phdthesis@%" write$ print.fields}
FUNCTION {proceedings} {"bibtex_type: proceedings@%" write$ print.fields}
FUNCTION {techreport} {"bibtex_type: techreport@%" write$ print.fields}
FUNCTION {unpublished} {"bibtex_type: unpublished@%" write$ print.fields}
FUNCTION {default.type} {"bibtex_type: unknown@%" write$ print.fields}

% Macros:
MACRO {jan} {"Januar"}
MACRO {feb} {"Februar"}
MACRO {mar} {"Maerz"}
MACRO {apr} {"April"}
MACRO {may} {"Mai"}
MACRO {jun} {"Juni"}
MACRO {jul} {"Juli"}
MACRO {aug} {"August"}
MACRO {sep} {"September"}
MACRO {oct} {"Oktober"}
MACRO {nov} {"November"}
MACRO {dec} {"Dezember"}

MACRO {acmcs} {"ACM Computing Surveys"}
MACRO {acta} {"Acta Informatica"}
MACRO {cacm} {"Communications of the ACM"}
MACRO {ibmjrd} {"IBM Journal of Research and Development"}
MACRO {ibmsj} {"IBM Systems Journal"}
MACRO {ieeese} {"IEEE Transactions on Software Engineering"}
MACRO {ieeetc} {"IEEE Transactions on Computers"}
MACRO {ieeetcad} {"IEEE Transactions on Computer-Aided Design of Integrated Circuits"}
MACRO {ipl} {"Information Processing Letters"}
MACRO {jacm} {"Journal of the ACM"}
MACRO {jcss} {"Journal of Computer and System Sciences"}
MACRO {scp} {"Science of Computer Programming"}
MACRO {sicomp} {"SIAM Journal on Computing"}
MACRO {tocs} {"ACM Transactions on Computer Systems"}
MACRO {tods} {"ACM Transactions on Database Systems"}
MACRO {tog} {"ACM Transactions on Graphics"}
MACRO {toms} {"ACM Transactions on Mathematical Software"}
MACRO {toois} {"ACM Transactions on Office Information Systems"}
MACRO {toplas} {"ACM Transactions on Programming Languages and Systems"}
MACRO {tcs} {"Theoretical Computer Science"}

% Ausfuehren :
READ
ITERATE {call.type$}
```

Anhang C

UML-Diagramme

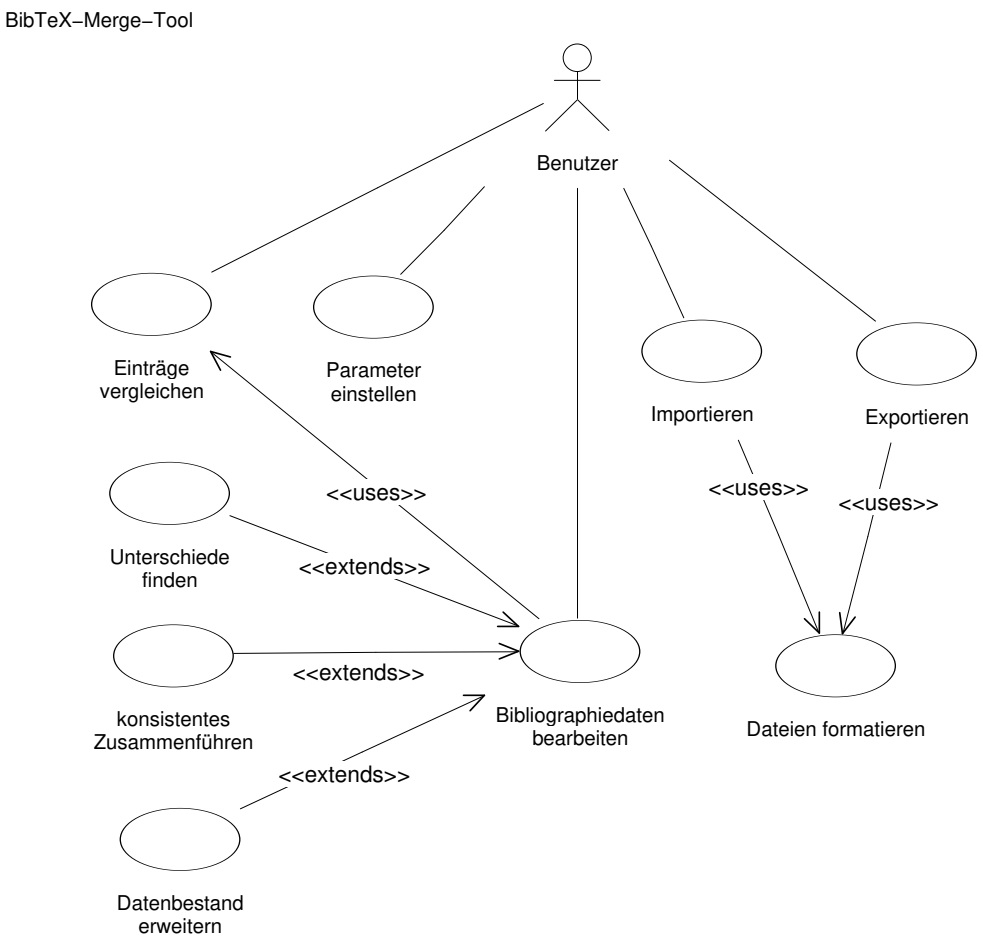

Abbildung C.1: Use Case Modell

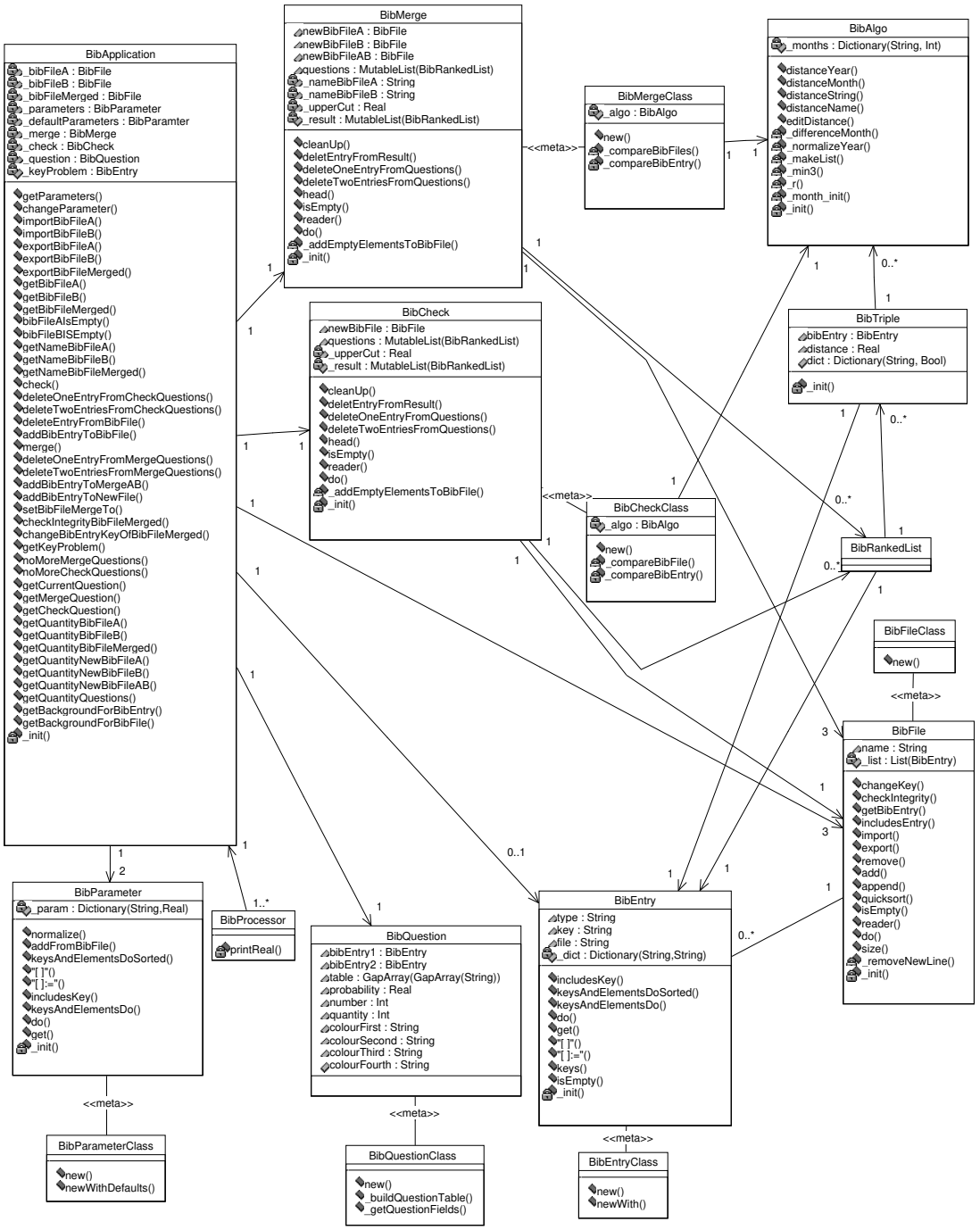

Abbildung C.2: Klassendiagramm

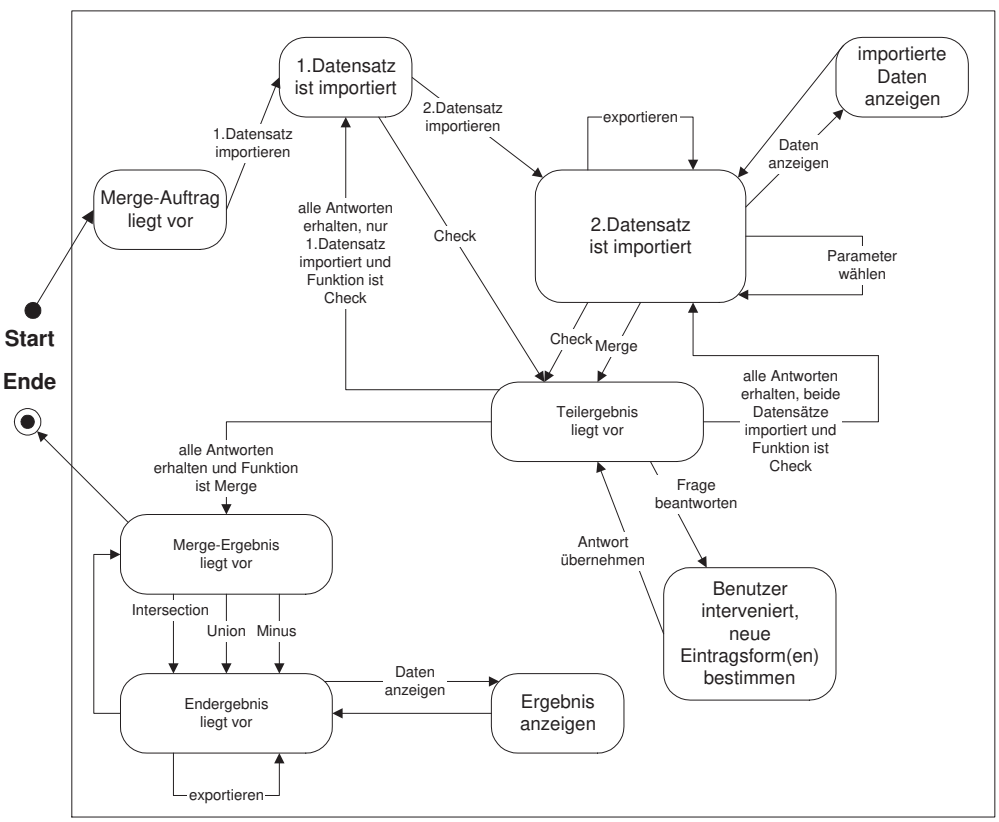

Abbildung C.3: Ablaufdiagramm des Systementwurfs

Anhang D

Systembeschreibung

D.1 BibAlgo

– siehe auch Kapitel 5.3.2 –

Slots

- *_months :Dictionary(String,Int)*
  Durch die Methode *_months_init* wird dieses Dictionary mit Werten belegt. Es enthält anschließend, für die verschiedenen Möglichkeiten einen Monat abzukürzen, die entsprechende Zahl. Ebenso sind für die englische Sprache alle Monate enthalten, um auch dies zu berücksichtigen. Für den Monat Oktober sind zum Beispiel folgende Einträge enthalten:

  oktober, october, oct, okt, 10

Methoden

- *distanceYear(:String, :String)*:
  Um zwei Jahresangaben zu vergleichen, werden diese zunächst durch die Methode *_normalizeYear* in zwei Integers übersetzt. Anschließend werden die beiden Zahlen miteinander verglichen. Sollten die beiden Zahlen gleich sein, so wird die Distanz 0.0 zurückgegeben. Ist die Differenz 1, so wird 0.5 und bei einer Differenz von 2 wird 0.8 zurückgegeben. Ansonsten ist die Distanz gleich 1.

- *distanceMonth(:String, :String) :Real*
  Diese Methode nutzt den privaten Slot *_months*, um einen Integer für die Monatsangabe zu erhalten. Allerdings werden beide Strings zuvor in Kleinbuchstaben überführt. Die von der Methode *_differenceMonth* zurückgegebene Distanz wird genauso wie bei *distanceYear* interpretiert.

- *distanceString(:String, :String) :Real*
  Mit Hilfe der Methode *editDistance* wird zunächst berechnet, an wie vielen Stellen sich die beiden Strings unterscheiden. Das Ergebnis wird durch die Länge des längeren Strings dividiert und als Ergebnis zurückgegeben.

- *distanceName(:String, :String) :Real*
  In Kapitel 4.3 wurden Ansätze beschrieben, um aus einem String möglichst viele Informationen zu erhalten. An dieser Stelle wird ein Teil dieser Ideen verwendet. Die beiden Strings werden mit der Methode *_makeList* in Teilstrings zerlegt. Die zurückgegebene Liste wird nun sequentiell abgearbeitet und die einzelnen Teilstrings mit der Methode *editDistance* verglichen. Die Summe der Unterschiede wird wiederum durch die Gesamtlänge dividiert.

- *editDistance(:String, :String) :Int*
  Die Funktionsweise dieser Methode ist bereits in Kapitel 4.1.1 beschrieben worden. Die Hilfsfunktion *_r* entspricht der dort beschriebenen Funktion *r*.

- *_differenceMonth(:String, :String) :Int*
  Durch Zugriff auf den Slot *_dict* werden die beiden Strings verglichen und die Differenz als Integer zurückgegeben.

- *_normalizeYear(:String) :Int*
  Für Jahreszahlen existieren im allgemeinen zwei Konventionen. Zum einen "1998" und zum anderen "98". Diese Methode gibt für beide Darstellungen die *Zahl* "98" zurück.

- *_makeList(:String) :MutableList(String)*
  Der String, den diese Methode erhält, wird auf Leerzeichen, Kommata und Semikolons durchsucht. Jedesmal wenn eines dieser Zeichen auftritt, wird der vordere Teil des String abgeschnitten und zu der Ergebnisliste hinzugefügt.

- *_months_init() :Void*
  Beim erzeugen eines Objektes dieser Klasse wird automatisch diese Methode ausgeführt und der Slot mit den beschriebenen Werten belegt.

D.2 BibApplication

Diese Klasse enthält alle Methoden, die die Applikation benötigt, um die STML-Seiten mit den nötigen Informationen zu versorgen, um daraus den HTML-Output zu generieren. Diese Methoden werden somit direkt aus den einzelnen STML-Seiten aufgerufen.

Slots

- *_bibFileA :BibFile*, *_bibFileB :BibFile* und *_bibFileMerged :BibFile*
  Die Slots *_bibFileA* und *_bibFileB* nehmen die importierten Daten aus den beiden BibTeX-Dateien auf. Der Slot *_bibFileMerged* soll das Endergebnis beinhalten.

- *_defaultParameters :BibParameter* und *_parameters :BibParameter*
  Der Slot *_defaultParameters* wird bereits beim Initialisieren mit den Werten aus der Datei "parameter.txt" gespeist. Bei jedem Importieren von Daten werden die entsprechenden Parameter aus diesem Slot ausgelesen. Der Slot *_parameters* kann vom Benutzer angepaßt werden. Er enthält nur die Parameter, die tatsächlich für die beiden BibTeX-Dateien benötigt werden.

- *_merge :BibMerge* und *_check :BibCheck*
  Um die Zwischenergebnisse und die einzelnen *RankedLists* zwischenzuspeichern, werden beim Aufruf der Methoden *merge* beziehungsweise *check* diese Slots als Puffer benötigt. Aus ihnen werden ebenfalls die einzelnen Fragen bei Bedarf herausgezogen.

- *_question :BibQuestion*
  Der Slot *_question* speichert die aktuelle Frage, die der Benutzer gerade beantworten soll.

- *_keyProblem :BibEntry*
  Die Methode *checkIntegrity*, die ein Objekt der Klasse *BibFile* bekommt, gibt als Ergebnis ein Objekt der Klasse *BibEntry* zurück. Dieses Objekt muß kurzfristig zwischengespeichert werden, um nach Intervention des Benutzers den Schlüssel dieses Objektes zu ändern.

Methoden

Da diese Klasse eine Fülle von Methoden besitzt, werden die Methoden mit geringer Funktionalität nicht aufgeführt. Hierzu zählt zum Beispiel die Methode *bibFileAISEmpty*, die die Liste des Objektes auf *isEmpty* abfragt. Eine vollständige Auflistung der Methoden enthält das Klassendiagramm in Anhang C.
Die große Anzahl von Methoden kommt durch die Kapselung der Anwendung zustande, da innerhalb der STML-Seiten möglichst geringe Funktionalität vorhanden sein soll. Des weiteren wurden sämtliche zu speichernde Objekte in privaten Slots aufgehoben, damit nur die vordefinierten Methoden auf sie zugreifen können.

- *getParameters() :BibParameter* und *changeParameter(:String, :Real) :Void*
  Zum Ändern und Anzeigen der Parameter greifen diese Methoden auf den privaten Slot *_parameters* zu. Die Methode *ChangeParameter* ändert einen Eintrag im Dictionary.

- *importBibFileA(:String) :Bool* und *importBibFileB(:String) :Bool*
  Diese Methoden schicken an den jeweiligen Slot (*_bibFileA* oder *_bibFileB*) die Nachricht *new* und anschließend *import*.

- *exportBibFileA(:String) :Bool*, *exportBibFileB(:String) :Bool* und *exportBibFileMerged(:String) :Bool*
  Genauso wie die vorigen Methoden leiten auch diese Methoden ihre Nachricht an die entsprechenden Slots weiter.

- *getBibFileA() :Reader(BibEntry)*, *getBibFileB() :Reader(BibEntry)* und *getBibFileMerged() :Reader(BibEntry)*
  Zum Anzeigen der Daten in den Slots wird auch hier nur die entsprechende Nachricht an die einzelnen Slots weitergeleitet.

- *check (:String) :Void* und *merge() :Void*
  Da *check* auf den beiden Slots *_bibFileA* und *_bibFileB* getrennt ausgeführt wird, benötigt *check* noch den Dateinamen, über den das Objekt identifiziert wird, als Parameter. Ansonsten enthalten auch diese Methoden keine eigene Funktionalität, sondern rufen nur mit den entsprechenden Objekten die nötigen Methoden auf.

- *deleteOneEntryFromCheckQuestions(:BibEntry) :Void* und *deleteOneEntryFromMergeQuestions(:BibEntry) :Void*
  Sollte der Benutzer entschieden haben, daß das System eine Gleichheit erkannt hat, die gar keine darstellt, so muß dieser Eintrag aus dem Fragenkatalog gelöscht werden. Dies ist für die beiden Vorgänge *Merge* und *Check* äquivalent und unterscheidet sich nur in dem betroffenen Slot.

- *deleteTwoEntriesFromCheckQuestions(:BibEntry, :BibEntry) :Void* und *deleteTwoEntriesFromMergeQuestions(:BibEntry, :BibEntry) :Void*
  Im Gegensatz zum vorherigen Fall wird diese Methode benötigt, wenn der Benutzer zwei Einträge als gleich erkannt hat. Nachdem der neue Eintrag erstellt wurde, müssen die beiden alten gelöscht werden.

- *deleteEntryFromBibFile(:BibEntry) :Void* und *addBibEntryToFile(:BibEntry) :Void*
  Beim *Check*-Vorgang ist es notwendig, wenn der Benutzer entschieden hat, daß zwei Einträge tatsächlich gleich sind, daß die beiden alten Einträge aus dem Datenbestand gelöscht werden und der neue hinzugefügt wird.

- *addBibEntryToMergeAB(:BibEntry) :Void* und *addBibEntryToNewFile(:BibEntry) :Void*
  Beim *Merge*-Vorgang müssen die Einträge im Gegensatz zum *Check*-Vorgang nie gelöscht werden, da die entsprechenden Einträge noch gar nicht in der Ergebnismenge vorhanden sind. Allerdings muß bei der Entscheidung des Benutzers, daß das System fälschlicherweise eine

Übereinstimmung erkannt hat, dieser Eintrag in die entsprechende Ergebnismenge eingefügt werden. Diese Ergebnismengen sind in dem Objekt der Klasse *BibMerge* als öffentliche Slots enthalten (*newBibFileA*,*newBibFileB* und *newBibFileAB*).

- *setBibFileMergedTo(:String) :Void*
  Je nach ausgewählter Operation (*A Minus B*, *B Minus A*, *Intersection* oder *Union*) müssen die einzelnen Ergebnismengen des Slots *_merge* miteinander verknüpft werden.

- *checkIntegrityBibFileMerged() :BibEntry*
  Um nach doppelten Eintragsschlüsseln zu suchen, wird diese Methode angestoßen. Die Suche erfolgt auf dem Slot *_bibFileMerged*.

- *changeBibEntryKeyOfBibFileMerged(:String) :Void*
  Falls ein doppelter Eintragsschlüssel in den Daten des Slot *_bibFileMerged* gefunden wurde, so wird der BibTeX-Eintrag, der im Slot *_keyProblem* gespeichert wurde, mit dem neuen Eintragsschlüssel versehen.

- *getKeyProblem() :BibEntry*
  Diese Methode gibt den BibTeX-Eintrag, der in dem Slot *_keyProblem* enthalten ist, zurück.

- *noMoreMergeQuestions() :Bool* und *noMoreCheckQuestions() :Bool*
  Wenn der Fragenkatalog des *Merge*- beziehungsweise *Check*-Vorganges leer ist, so gibt diese Methode *"true"* zurück.

- *getCurrentQuestion() :BibQuestion*
  Gibt die aktuelle Frage zurück, die der Benutzer beantworten soll.

- *getMergeQuestion(:Int) :BibQuestion* und *getCheckQuestion(:Int) :BibQuestion*
  Falls das System mehrere Einträge gefunden hat, mit denen ein BibTeX-Eintrag übereinstimmen kann, so wird durch den Parameter festgelegt, welche der Benutzer auf seinen Bildschirm bekommt.

- Die nachfolgenden Methoden geben für die Präsentation die Anzahl der Elemente in den jeweiligen Mengen zurück
  - *getQuantityBibFileA() :Int*
  - *getQuantityBibFileB() :Int*
  - *getQuantityBibFileMerged() :Int*
  - *getQuantityNewBibFileA() :Int*
  - *getQuantityNewBibFileB() :Int*
  - *getQuantityNewBibFileAB() :Int*
  - *getQuantityQuestions() :Int*

- *getBackgroundForBibEntry(:BibEntry) :String* und
  getBackgroundForBibFile(:String) :String
  Diese beiden Methoden geben als RGB-Code die Farbgestaltung eines Objektes der Klasse *BibEntry* oder *BibFile* zurück.

D.3 BibCheckClass

Slots

- *_algo :BibAlgo*
  Um einzelne Objekte der Klasse *BibEntry* zu vergleichen werden Algorithmen benötigt, die in den Methoden der Klasse *BibAlgo* enthalten sind.

Methoden

- *new(:BibFile, :BibParameter) :BibCheck*
  Wie bei jeder *new*-Methode wird das Objekt der entsprechenden Klasse initialisiert. Zusätzlich wird der Slot *_result* mit dem Ergebnis der angestoßenen Methode *_compareBibFile* belegt, und in dem Slot *_upperCut* wird der entsprechende Wert aus dem Objekt der Klasse *BibParameter* gespeichert.

- *_compareBibFile(:BibFile, :BibParameter) :MutableList(BibRankedList)*
  – siehe Kapitel 5.3.2 –

- *_compareBibEntries(:BibEntry, :BibEntry, :BibParameter) :BibTriple*
  – siehe Kapitel 5.3.2 –

D.4 BibCheck

Slots

- *newBibFile :BibFile*
  Nach dem Aufruf der Methode *cleanUp* entsteht in diesem Slot das neue Objekt, das als Ergebnis der Systemfunktion *Check* zurückgegeben wird. Durch Intervention des Benutzers kommen ebenso nach und nach Elemente aus dem Slot *questions* hinzu.

- *questions :BibQuestion*
  Sollten beim Aufruf der Methode *cleanUp* nicht alle BibTeX-Einträge eindeutig abgearbeitet werden können, so werden diese Einträge mit zusätzlichen Informationen als offene Fragen in diesem Slot abgelegt.

- *_result :MutableList(BibRankedList)*
  Dieser Slot wird, wie bereits beschrieben, beim Aufruf der *new*-Methode der Metaklasse belegt. Hierin befindet sich das Zwischenergebnis der *Check*-Funktion.

- *_upperCut :Real*
  Auch dieser Slot wird beim Aufruf der *new*-Methode der Metaklasse belegt. Die Bedeutung dieses Slots wurde bereits bei der Beschreibung der Klasse *BibParameter* geklärt.

Methoden

- *cleanUp() :Void*
  – siehe Kapitel 5.3.2 –

- *deleteEntryFromResult(:BibEntry) :Void*
  Falls der eben erwähnte Fall eintreten sollte und zwei Einträge mit einer Wahrscheinlichkeit übereinstimmen, die größer ist als der *upperCut*, so löscht diese Methode die überflüssigen Einträge.

- *deleteOneEntryFromQuestions(:BibEntry) :Void*
  Sollte der Benutzer eine offene Frage beantwortet haben, so kann dieser Eintrag aus den offenen Fragen, also aus dem Slot *questions*, gelöscht werden. Hierbei müssen allerdings alle Objekte des Slots *questions* bearbeitet werden, da dieser Eintrag aus sämtlichen *RankedLists* ebenso verschwinden muß. Hierdurch könnte eine *RankedList* leer geworden sein, so daß die Methode *_addEmptyElementsToBibFile* ausgeführt wird.

- *deleteTwoEntriesFromQuestions(:BibEntry, :BibEntry) :Void*
  Diese Methode verhält sich äquivalent zu *deleteOneEntryFromQuestions*. Sie besitzt ihre

Berechtigung in der Tatsache, daß am Schluß beider Methoden zusätzlich einmal die Methode *_addEmptyElementsToBibFile* aufgerufen wird. Würde man die Methode *deleteOneEntryFromQuestions* zweimal hintereinander ausführen, so würde womöglich der Eintrag, der eigentlich beim zweiten Aufruf gelöscht werden soll, durch die Methode *_addEmptyElementsToBibFile* in den Slot *newBibFile* geschrieben werden.

- *_addEmptyElementsFromBibFile(:MutableList(BibRankedList)) :MutableList(BibRankedList)*
  – siehe Kapitel 5.3.2 –

D.5 BibEntry

– siehe Kapitel 5.3.1 –

D.6 BibEntryClass

– siehe Kapitel 5.3.1 –

D.7 BibFile

– siehe Kapitel 5.3.1 –

D.8 BibFileClass

– siehe Kapitel 5.3.1 –

D.9 BibMergeClass

Slots

- *_algo :BibAlgo*
  – siehe *BibCheckClass* –

Methoden

- *new(:BibFile, :BibFile, :BibParameter) :BibMerge*
  Die Funktionsweise ist sehr ähnlich zur *new*-Methode der Klasse *BibCheckClass*. Jedoch wird nicht die Methode *_compareBibFile* angestoßen, sondern *_compareBibFiles*, die auf zwei Objekten der Klasse *BibFile* arbeitet. Zusätzlich werden noch die beiden Slots *_nameBibFileA* und *_nameBibFileB* mit den entsprechenden Attributen der beiden mitgelieferten Objekte belegt.

- *_compareBibFiles(:BibFile, :BibFile :BibParameter) :MutableList(BibRankedList)*
  Die Struktur entspricht der Methode *_compareBibFile*, jedoch wird hier nicht ein Objekt der Klasse *BibFile* mit sich selbst verglichen, sondern ein Objekt mit einem anderen.

- *_compareBibEntries(:BibEntry, :BibEntry, :BibParameter) :BibTriple*
  – siehe *BibCheckClass* –

D.10 BibMerge

Diese Klasse und seine Metaklasse *BibMergeClass* unterstützen die Funktionen *Union, Intersection* und *Minus*.

Slots

- *newBibFileA :BibFile*, *newBibFileB :BibFile* und *newBibFileAB :BibFile*
  Im Gegensatz zu *BibCheck* entstehen bei den Funktionen *Union, Intersection* und *Minus* drei Ergebnismengen. Diese drei Ergebnismengen spiegeln sich in diesen Slots wieder. Ein Mengendiagramm zur Erklärung des Zusammenhanges dieser drei Slots folgt in Kapitel 5.3.3, da diese Mengen insbesondere für die Präsentation eine Rolle spielen.

- *questions :BibQuestion*
  – siehe *BibCheck* –

- *_result :MutableList(BibRankedList)*
  – siehe *BibCheck* –

- *_upperCut :Real*
  – siehe *BibCheck* –

- *_nameBibFileA* und *_nameBibFileB*
  Um einzelne Einträge in die jeweiligen Slots richtig zuzuordnen, werden die Namen der beiden BibTeX-Dateien benötigt. Diese beiden Slots werden bereits durch die *new*-Methode der Metaklasse mit ihren Werten belegt.

Methoden

- *cleanUp() :Void*
  Die Funktion dieser Methode entspricht der namensgleichen aus der Klasse *BibCheck*. Jedoch müssen die einzelnen Einträge nicht nur einem einzigen Slot *newBibFile*, sondern je nach Situation den drei Slots *newBibFileA*, *newBibFileB* oder *newBibFileAB* zugeordnet werden.

- *deleteEntryFromResult(:BibEntry) :Void*
  – siehe *BibCheck* –

- *deleteOneEntryFromQuestions(:BibEntry) :Void*
  – siehe *BibCheck* –

- *deleteTwoEntriesFromQuestions(:BibEntry, :BibEntry) :Void*
  – siehe *BibCheck* –

- *_addEmptyElementsFromBibFile(:MutableList(BibRankedList)) :MutableList(BibRankedList*
  Genauso wie die Methode *cleanUp* mußte diese Methode nur an die drei Slots angepaßt werden. Ansonsten funktioniert sie genauso wie die namensgleiche Methode aus *BibCheck*.

D.11 BibParameter

– siehe Kapitel 5.3.2 –

D.12 BibParameterClass

– siehe Kapitel 5.3.2 –

D.13 BibProcessor

– siehe Kapitel 5.3.3 –

D.14 BibQuestionClass

Methoden

- *new(:BibRankedList, :Int, :String, :String) :BibQuestion*
  Die beiden Strings enthalten die jeweiligen Namen der beiden BibTeX-Dateien. Dies ist in diesem Kontext für die farbige Gestaltung wichtig, da die Einträge beider Dateien farblich voneinander zu unterscheiden sein sollen.
  Da das System zu einem Eintrag mehrere mögliche Übereinstimmungen gefunden haben kann, wird in Abhängigkeit von der Zahl vom Typ *Int* der entsprechende BibTeX-Eintrag aufbereitet. Allerdings können nie zwei oder mehr Einträge einer Datei mit einem Eintrag der anderen Datei übereinstimmen. Dies verbietet zum einen die Vorbedingung aus der Anforderungsanalyse und zum anderen sollen diese Elemente bereits durch die *Check*-Funktion herausgefiltert worden sein.
  Das Objekt der Klasse *BibRankedList* ist der Informationsträger. Sie enthält sowohl die beiden BibTeX-Einträge als auch die Wahrscheinlichkeiten, mit denen Sie übereinstimmen.

- *_buildQuestionTable(:Pair(:BibEntry, :BibTriple):GapArray(GapArray(String))*
  Diese Methode wird innerhalb der *new*-Methode aufgerufen. Diese Methode liefert eine Tabelle zurück, die vier Spalten besitzt.
  In der ersten Spalte stehen alle Feldnamen der beiden BibTeX-Einträge. Die Spalten zwei und drei enthalten die beiden BibTeX-Einträge selbst. Sollte ein Feld in einem der beiden BibTeX-Einträge nicht vorhanden sein, so bleibt die entsprechende Zelle leer. In der vierten Spalte werden nur die Zellen ausgefüllt, bei denen die Felder der beiden BibTeX-Einträge exakt gleich waren. Dies soll später dem Benutzer die Arbeit erleichtern, und er soll Unterschiede schneller finden.

- *_getQuestionFields(:BibEntry, :BibEntry):GapArray(String)*
  Die Methode *_buildQuestionTable* benötigt alle Felder der beiden BibTeX-Einträge. Hierzu wird diese Methode aufgerufen, um Funktionalität zu kapseln und Übersichtlichkeit zu gewinnen.

D.15 BibQuestion

Ein Objekt dieser Klasse soll alle Informationen enthalten, die benötigt werden, um den Benutzer über die Gleichheit zweier BibTeX-Einträge entscheiden zu lassen.
Hierzu muß die Information, die in einem Objekt der Klasse *BibRankedList* enthalten ist, so aufgearbeitet werden, daß sie präsentationsfähig ist.

Slots

- *bibEntry1 :BibEntry* und *bibEntry2 :BibEntry*
  Die beiden Slots enthalten die Objekte, die vom Benutzer verglichen werden sollen.

- *table :GapArray(GapArray(String))*
  Diese Array-Struktur enthält die Tabelle, die durch die Methode *_buildQuestionTable* berechnet wurde. Der Benutzer erhält diese Tabelle auf den Bildschirm.

- *probability :Real*
  Die Wahrscheinlichkeit, mit der die beiden BibTeX-Einträge aus den beiden Slots *bibEntry1* und *bibEntry2* übereinstimmen.

- *number :Int*
  Wie bereits erwähnt, kann das System zu einem BibTeX-Eintrag mehrere Übereinstimmungen erkannt haben. Diese Zahl spiegelt die *n-te* Alternative wieder. Diese Zahl kann niemals größer als *quantity* werden.

- *quantity :Int*
  Die Anzahl der möglichen Übereinstimmungen, die ein System erkannt hat.

- *colourFirst :String, colourSecond, colourThird* und *colourFourth*
  Die einzelnen Farben der Spalten für die Tabelle in dem Slot *table*. Allerdings sind *colourFirst* und *colourFourth* unveränderlich. *ColourSecond* und *colourThird* hängen jedoch davon ab, aus welcher BibTeX-Datei der entsprechende Eintrag stammt.

D.16 BibRankedList

RankedLists sind eines der häufigsten Ergebnisse von Information Retrieval Systemen. Hierbei wird eine Liste als Ergebnis zurückgeliefert, in der jedes Element einen Wert besitzt, so daß die einzelnen Elemente durch diesen Wert in eine Relation gesetzt werden können.
Ein Objekt der Klasse *BibRankedList* besitzt ein Objekt vom Typ BibEntry und eine Liste von Objekten der Klasse BibTriple. Diese Liste entspricht einer erweiterten *RankedList*.

D.17 BibTriple

Diese Klasse spiegelte das Ergebnis eines Vergleiches zweier Objekte der Klasse *BibEntry* wieder. Wenn ein Objekt A mit einem Objekt B verglichen wird, so wird das Ergebnis in einem Objekt der Klasse *BibTriple* gespeichert. Die beiden Slots *bibEntry* und *distance* enthalten das Objekt B und die Distanz, die das Ergebnis des Vergleiches ist.
Zusätzlich wird für die spätere Präsentation die Information benötigt, ob zwei Felder mit einer Wahrscheinlichkeit von 1 übereinstimmen. Diese Information wird in dem Slot *dict* gespeichert.

Literaturverzeichnis

[1] Cardelli, Luca ; Wegner, Peter: On Understanding Types, Data Abstraction, and Polymorphism. In: *Computing Surveys* (1985)

[2] Eckel, Bruce: *Thinking in JAVA*. erste. Prentice Hall, 1998

[3] French, James ; Powell, Allison ; Schulmann, Eric ; Pfaltz, John: Automating the Construction of Authority Files in Digital Libraries: A Case Study. In: *Research and Advanced Technology for Digital Libraries*, Springer, 1997. – ISBN 3–540–63554–8

[4] Fuhr, Norbert: Information Retrieval / Universität Dortmund. 1996. – Forschungsbericht

[5] Gawecki, Andreas ; Wienberg, Axel: *STML Developer's Guide*. Higher-Order GmbH, 1997

[6] Goldberg, Adele ; Robson, David: *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1985

[7] Lamport, Leslie: *Das LaTeX Handbuch*. erste. Addison-Wesley, 1995. – ISBN 3–89319–826–1

[8] Lindholm, Tim ; Yellin, Frank: *The Java Virtual Machine Specification*. Addison Wesley, 1996

[9] Matthes, Florian: *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung*. Springer Verlag, 1993

[10] Oestereich, Bernd: *Objektorientierte Softwareentwicklung mit der UML*. erste. Oldenburg, 1997. – ISBN 3–486–24319–5

[11] Patashnik, Oren: BibTeXing / Stanford University. 1988. – Forschungsbericht

[12] Ragett, Dave: HTML 3.2 Reference Specification / W3C. www.w3.org, 1997. – Forschungsbericht

[13] van Rijsbergen, C.J.: Information Retrieval / University of Glasgow. 1996. – Forschungsbericht

[14] Sebesta, R. W.: *Concepts of Programming Languages*. Benjamin/Cummings, 1989

[15] Wahlen, Jens: *Entwurf einer objekt-orientierten Sprache mit statischer Typisierung unter Beachtung kommerzieller Relevanz*, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 1998

[16] Zobel, Justin ; Dart, Philip: Phonetic String Matching: Lessons from Information Retrieval / University of Melbourne. 1996. – Forschungsbericht