WIRTSCHAFTSUNIVERSITÄT WIEN
DIPLOMARBEIT

Titel der Diplomarbeit:
„LaTeX Literaturmanagement“

Englischer Titel der Diplomarbeit:
„LaTeX Bibliography Management“

Verfasser:	Alexander Fröhlich
Matrikel-Nr.:	8751967
Studienrichtung:	Betriebswirtschaft J-151
Textsprache:	Deutsch
Beurteiler:	Univ. Prof. Dipl.-Ing. Mag. Dr. Wolfgang Panny
Betreuer:	Dipl.-Ing. Mag. Dr. Albert Weichselbraun

Ich versichere,

dass ich die Diplomarbeit selbstständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfe bedient habe,

dass ich die Ausarbeitung zu dem obigen Thema bisher weder im In- noch im Ausland (einer Beurteilerin/ einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe,

dass diese Arbeit mit der vom Betreuer beurteilten Arbeit übereinstimmt.

_____ _____

Datum Unterschrift

Abstract

Diese Arbeit befasst sich mit Literaturmanagement im Umfeld von LaTeX. Im ersten theoretischen Teil wird auf den Zusammenhang zwischen LaTeX, der bibliographischen Umgebung von LaTeX „thebibliography“ und BibTeX eingegangen. Produkte, die sich mit der Unterstützung, Erweiterung oder Neuentwicklung von BibTeX auseinandersetzen, werden vorgestellt. Im Besonderen werden Neuentwicklungen von BibTeX beschrieben, die sich mit Erweiterungen hinsichtlich Multilingualität beschäftigen. Ein anderer Kernpunkt ist, Alternativen aufzuzeigen, die versuchen, fehlende Modularität und komplexe Wartbarkeit von BibTeX-*bst* zu beseitigen. Im zweiten Teil dieser Arbeit wird eine komplette Übersicht über die Arbeitsweise von BibTeX gegeben. Neben der Syntax der Sprache *bst* werden interne BibTeX-Funktionen dargestellt. Anhand von Standard-Style-Implementierungen wird der komplette Aufbau einer *bst*-Datei erklärt. Der dritte Teil dieser Arbeit beschreibt BibSQL, eine Neuimplementierung von BibTeX in SQL und Python. Dabei wird von einer Analyse der funktionalen Anforderungen ausgegangen. In einer Machbarkeitsstudie werden Anforderungen, die sich aus funktionalen Vorgaben von BibTeX und den neuen Anforderungen ergeben, untersucht. Abschließend wird die technische Umsetzung im Detail beschrieben.

Key Words
LaTeX, BibTeX, MlBibTeX, BibTeX++, BibSQL

Abkürzungen

3NF	Dritte Normalform
.bib	BibTeX Datenbank-File
.bst	BibTeX Style-File
API	Application Programming Interface
CSA	Cambridge Scientific Abstracts
CSL	Citation Style Language
DDL	Data Denition Language
DML	Data Manipulation Language
DOM	Document Object Model
ER	Entity Relation
GUI	Graphical User Interface
IEEE	The Institute of Electrical and Electronics Engineers
ISI	Institute for Scientific Information
MARC	Machine-Readable Cataloging
MODS	Metadata Object Description Schema
MS	Microsoft
RDBMS	Relational Database Management System
RIS	Research Information Systems
RTF	Rich Text Format
SAX	Simple API for XML (XML-parser)
SQL	Structured Query Language
SRU	Search/Retrieval via URL
SSAX	S-exp based XML-parsing (http://www196.pair.com/lisovsky/xml/ssax/)
SXML	S-exp based XML (http://okmij.org/ftp/Scheme/SXML.html)
SXSLT	Manipulation Language for XML (http://www.omegahat.org/Sxslt/)

UPN	Umgekehrte Polnische Notation
W3C	The World Wide Web Consortium
XML	Extensible Markup Language
XPath	XML Path Language
XSLT	Extensible Stylesheet Language for Transformation

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einführung

LaTeX (sprich „Lah-tech“ oder „Lej-tech“, kann auch „LaTeX“ geschrieben werden) ist ein auf TeX aufbauendes Computerprogramm, das von Leslie Lamport [Lam95] geschrieben wurde. Es vereinfacht den Umgang mit TeX, indem es entsprechend der logischen Struktur des Dokuments auf vorgefertigte Layout-Elemente zurückgreift.
BibTeX ist ein Programm, das bibliographische Daten, die in einem bestimmten Datenformat gespeichert sind, ausliest und mit dem Text eines Dokuments (einer Hausarbeit, einer Dissertation, eines Buches ...) verknüpft. Zur Verknüpfung dient ein Zitierschlüssel, der durch den Befehl \cite {Zitierschluessel} in das Dokument eingebunden wird. Die eigentlichen bibliographischen Daten werden von BibTeX bei einem Kompilationsdurchlauf aus der *bib*-Datei ausgelesen und formatiert. Zur Formatierung steht eine Reihe von Zitierstilen bereit, die beliebig eingebunden werden können.

1.1 Motivation

BibTeX ist eine Open Source Software für die Verwaltung von Literaturquellen und unterstützt das Einbinden von Literaturdatenbanken in ein LaTeX-Dokument. BibTeX in der Version 1.0 hat sich als Quasistandard etabliert, wenn es darum geht, Literaturdaten mit LaTeX zu verknüpfen.
Die BibTeX Datenbank basiert auf einem einfach strukturierten Dateiformat. Wegen der einfachen Bedienung wird das BibTeX-Dateiformat auch in Zukunft eine wesentliche Rolle spielen. Eine große Anzahl an bibliographischen Daten ist immer noch im BibTeX-Format verfügbar. BibTeX wird aber selten als Exportformat für Literaturdaten verwendet. Viele Kataloge unterstützen das EndNote-Format oder einfach unstrukturierte Textformate. Bei Verwendung von BibTeX mit nicht englischsprachigen Textelementen ist die Bedienung durch die fehlende UTF8-Unterstützung sehr umständlich. Bei mehrsprachigen Dokumenten ist die Unterstützung unterschiedlicher Encodings stark eingeschränkt.
Ein weiterer Nachteil von BibTeX ist die umständliche Handhabung mit dem Stilformat, was besonders bei mehrsprachigen Projekten ein Problem darstellt. Viele alternative Produkte und Projekte beschäftigen sich damit, diese Benutzermängel zu beseitigen und bieten eine Fülle an Möglichkeiten, bibliographische Daten und Format-Stile zu verwalten,

sind aber beim Einbinden in das LaTeX-System nicht flexibel genug. Strukturiertere Formate wie XML spielen beim Speichern und Austauschen von Literaturdaten, besonders in der professionellen Literaturverwaltung, eine große Rolle. Viele Projekte befassen sich mit der Konvertierung von BibTeX zu moderneren Dateiformaten. Projekte wie Bibutils verwenden eigene Dateiformate, wie z.B. MODS, das als Schnittstellen-Dateiformat zwischen BibTeX und Endnote dient, jedoch keine Konvertierungsmöglichkeit von XML nach BibTeX beinhaltet. Diese Arbeit soll einen Überblick über die unterschiedlichen Ansätze verschaffen. Aus diesen Erkenntnissen soll ein Anforderungskatalog für eine BibTeX-Neuentwicklung entstehen.

1.2 Vor- und Nachteile von BibTeX

Der Vorteil von BibTeX ist die Integration im LaTeX „thebibliography environment“. BibTeX ist im Prinzip sehr offen aufgebaut, die Datenbank ist generisch, d.h. Feldtypen sind frei definierbar. Durch Stildateien, die eigentlich Programmcode sind, ist eine große Flexibilität in der Gestaltung der Ausgabe gegeben. Diese namenlose Programmiersprache, in Folge *bst* genannt, bietet alle Möglichkeiten und Gestaltungsspielräume. Auch Erweiterungen in Richtung Mehrsprachigkeit, wie das Paket babelbib zeigt, sind möglich. BibTeX arbeitet als Präprozessor für LaTeX. Dahinter liegt ein Interpreter, der eine Stack-basierende Programmiersprache unterstützt. Der Programmcode wird in sogenannte Style-Dateien mit dem Suffix *bst* abgelegt und vom BibTeX Präprozessor interpretiert. Anwender müssen die gewöhnungsbedürftige UPN-Sprache lernen. Die Standard-*bst*-Dateien bieten eine Vielzahl von Funktionen an, die verstanden werden müssen. Erst dann ist der Anwender in der Lage stabile Änderungen an Style-Dateien vorzunehmen. Ein weiterer Nachteil ist die nicht sehr ausführliche Dokumentation des Präprozessors. Die Programmiersprache *bst* besteht aus mehr als 50 Schlüsselwörtern. Die Datenbasis ist unstrukturiert, lässt Redundanzen zu und ist nur schwer vernetzbar. Verteilte Serverlösungen sind nur eingeschränkt möglich.

1.3 Problemstellung

BibTeX besteht im Wesentlichen aus zwei Komponenten. Die erste Komponente ist die Literaturdatenbank, die zweite Komponente beschäftigt sich mit der Formatierung von Literatureinträgen. Beide Komponenten werden in einfachen Dateien als Text abgespeichert. Obwohl beide Themen getrennt betrachtet werden müssen, stehen sie in einem gewissen Zusammenhang, da es sich bei BibTeX um eine generische Datenbank handelt.

Die generische Datenhaltung zeichnet sich dadurch aus, dass die Eintrags- und Feldtypen erst durch die formatierenden Styles definiert werden.
Multibyte Encoding (z.B. UTF8) wird vom Präprozessor nur eingeschränkt unterstützt. Bibliographische Daten werden problemlos von einem UTF8-Datenfile in ein UTF8-*bbl*-File übertragen. Intern kann BibTeX mit UTF8 nicht umgehen, was sich durch fehlerhafte Sortierung von Literatureinträgen äußert.
File-basierende Datenbanken bringen Probleme mit sich. Die Vorgaben sind zwar strukturiert, während der Eingabe werden aber keine Prüfungen vorgenommen. Das bedeutet, dass bei der Eingabe keine Prüfung auf fehlende Felder, Datentypen oder Querverweise stattfindet. Dieses Problem kann auch eine relationale Datenbank nicht lösen. Wenn BibTeX-Datenbankfiles in der Struktur einschränkt werden, geht der große Vorteil der generischen Datenhaltung verloren. Eine neue Lösung sollte kompatibel zu Standard-*bst*-Files sein und möglichst viele Formate anderer Literaturdatenbanken unterstützen.
BibTeX ist ein verbreiteter Standard zum Speichern von Literatur-Informationen, eignet sich aber nicht uneingeschränkt für den Datenaustausch. Somit ist BibTeX für Mehrbenutzerumgebungen nicht geeignet.
Viele Literaturdatenbanken unterstützen den Export von *bib*-Dateien, können aber nicht direkt eingebunden werden. Pakete wie MlBibTeX sind zwar multilingual ausgerichtet, bieten aber derzeit keine Unicode Unterstützung und sind nur in C-Quellcode verfügbar. Weiterentwicklungen vom LaTeX-\cite Kommando sind kompatibel zum *bst*-Standard, unterstützen aber nicht UTF8.

1.4 Ziele

Ziel dieser Arbeit ist es, Kriterien und Anforderungen an ein modernes Bibliographiemanagement-System und eine komplette Beschreibung der Arbeitsweise von BibTeX, insbesondere der Stilbeschreibungssprache *bst*, zu liefern. Eine Neuimplementierung von BibTeX soll eine flexiblere Style-Gestaltung ermöglichen, ohne auf die grundlegende Funktionalität von BibTeX zu verzichten. Dabei sollen bestehende Standard-Files, unter Benutzung einer Standardprogrammiersprache, wiederverwendet werden können.
Hauptziel ist es, eine modulare Architektur zu schaffen, um Weiterentwicklungen zu erleichtern. Die Anbindung an andere Systeme und Formate soll möglichst offen sein. Dabei müssen Schnittstellen in eigenen Modulen erweiterbar sein. Die volle Funktionalität LaTeX/BibTeX muss erhalten bleiben.

1.5 Methodik

Auf Basis von Recherchen über Literaturmanagement-Systeme sollen Klassifizierungen und Kriterien festgelegt werden. Dabei werden Formate und Schnittstellenbeschreibungen erstellt und Anforderungen an ein neues System definiert. Um die Standard BibTeX-Funktionalität zu erhalten, müssen der LaTeX/BibTeX Arbeitsablauf, der Aufbau, die Arbeitsweise und der Funktionsumfang von Styles beschrieben werden. Auf Basis einer Machbarkeitsstudie und funktionaler Anforderungen wird ein technisches Konzept erstellt.

Arbeitsschritte:

- Analyse von BibTeX-Arbeitsablauf
- Beschreibung der Syntax von BibTeX
- Beschreibung aller eingebauten Funktionen
- Beschreibung von *bst*-Standardfunktionen
- Beschreibung von Formatierungsfunktionen
- Analyse bestehender Lösungen
- Schreiben eines Übersetzers (Auswahl der geeigneten Programmiersprache)
- Auswahl des Zielformats
- Funktionierende dokumentierte Lösung

2 Literaturverwaltungssysteme

Literaturverwaltungssysteme dienen hauptsächlich dem erleichterten Umgang mit Literaturquellen. Rechnergestützte Systeme (Literaturverwaltungssoftware) bieten die Möglichkeit, einen Datenpool bibliographischer Angaben oder Literaturquellen anzulegen, vgl. [KS09]. Diese Programme unterstützen auch die Strukturierung literaturgestützter Arbeiten. Dabei ist zwischen webbasierten und lokalen Systemen zu unterscheiden. Webbasierte Systeme unterstützen in den meisten Fällen den Zugang zu Literaturdatenbanken. Auch lokal installierte Arbeitsplatzsysteme bieten in manchen Fällen direkten Zugriff auf bibliographische Datenbanken.
Neben Literaturrecherche und -verwaltung von bibliographischen Daten bieten die meisten Systeme die Möglichkeit, Literaturquellen direkt in wissenschaftliche Arbeiten einzubinden. Dabei werden Elemente wie Zitierschlüssel und Literaturverzeichnisse erstellt, sortiert und formatiert.
Ein wichtiges Kriterium von Literaturverwaltungssystemen ist die Unterstützung von Standards für:

- Speicherformat bibliographischer Daten
- Import und Export fremder Datenformate
- Schnittstellen zu externen Datenquellen
- Speicherformat der Zitierstile
- Import und Export fremder Stilformate

2.1 LaTeX Literatur-Management-Systeme

BibTeX ist der gebräuchlichste bibliographische Prozessor in Verbindung mit LaTeX. Aus diesem Grund beschäftigen sich mehrere Versuche mit unterschiedlichen Ansätzen, BibTeX weiter oder neu zu entwickeln. Eine vergleichende Studie verschiedener Ansätze [Huf08] soll Auswirkungen auf das Design von MlBibTeX untersuchen. Dabei werden Softwarequalitätsanforderungen der unterschiedlichen Lösungsansätze BibTeX

gegenübergestellt. Untersucht werden BibTeX ++, cl-bibtex, MlBibTeX sowie die Pakete natbib, jurabib und Tib. Dabei werden die verwendeten Sprachen XML, XSLT, nbst, Perl, DSSSL auf folgende Kriterien untersucht:

- Fehlerfreiheit: Software entspricht den angeforderten Spezifikationen
- Robustheit: Programme arbeiten stabil, auch bei abnormen Bedingungen
- Erweiterbarkeit: Software ist bei ändernden Anforderungen einfach zu erweitern
- Wiederverwendbarkeit: Die Möglichkeit, Programme in Kombination mit anderen zu verwenden

Andere untersuchte Faktoren sind Effizienz, Portabilität, Integrität, Nachvollziehbarkeit, Verwendbarkeit und Lesbarkeit. In Bezug auf Modularität wird der Grad der Zerlegbarkeit und die Möglichkeit der Komponentenbildung betrachtet. J. M. Hufflen kommt in seiner Studie [Huf08] zu folgendem Ergebnis:

	BibTeX	XSLT	DSSSL	Bibulus
Correctness	ja			
Robustness	ja	ja	ja	
Extendability	nein	ja	nein	ja/nein
Reusability	ja/nein			
Modularity	schlecht	ja	ja	ja
Continuity	Durchschnitt	ja	ja	ja
Efficiency	ja		ja	
Ease of use	ja		nein	nein

Tabelle 2.1: Sprachen für bibliographische Styles[Huf08]

Wenn LaTeX als Textsatzprogramm zum Einsatz kommt, nimmt BibTeX einen besonderen Stellenwert ein. Aufgrund der Bedeutung von LaTeX und der bereits angeführten Probleme beim Einsatz von BibTeX haben sich mehrere Projekte mit der Weiter- und Neuentwicklung im Umfeld von LaTeX Thebibliograpy Environment beschäftigt.
Diese Projekte unterscheiden sich wesentlich durch ihren Ansatz:

- Erweiterte LaTeX-Pakete, die unabhängig von BibTeX arbeiten
- Erweiterte BibTeX-*bst*-Styles oder Weiterentwicklungen bestehender *bst*-Files
- Neuimplementierung von BibTeX und Ablöse von BibTeX

Als Standardliteratur für BibTeX gilt [Pat88a]. In [Pat88b] beschreibt der Autor von BibTeX in kurzer Form die wesentlichen Bestandteile der Style-Sprache bst. Eine wesentlich ausführlichere Beschreibung von BibTeX , im Besonderen der Sprache bst, bietet [Mar05]. Dabei wird neben den Besonderheiten der UPN-Sprache[1] bst auch auf interne *bst*-Funktionen eingegangen.
Vor- und Nachteile von BibTeX gibt Jürgen Fenn in [Fen07]. Eine Beschreibung der bst- Programmierung mit vielen Beispielen und einer kompletten Übersicht der Sprachmerkmale bekommt man in [Rai02]. Dabei werden anhand von einfachen Beispielen der Aufbau der *bst*-Datei sowie Datentypen, Variablen und vordefinierte Funktionen erklärt. Im Besonderen wird auf die Postfix-Notation von BibTeX eingegangen. Einen Überblick über die Abläufe in BibTeX gibt Uwe Siart in [Sia08]. Darin werden auch die wichtigsten Hilfspakete wie z.B. babelbib vorgestellt.

2.1.1 Erweiterte BibTeX-*bst*-Styles

Bei erweiterten BibTeX *bst*-Styles geht es darum, durch Neuimplementierung von *bst*-Dateien erweiterte Funktionalität bzw. multilinguale Fähigkeiten mittels BibTeX zu erreichen. Nachteile, die sich durch die *bst*-Sprache ergeben, bleiben bestehen. Durch die fehlende Modularität müssen bestehende bibliographische Stile neu entwickelt werden.
Das Paket babelbib [Har02] unterstützt im Zusammenspiel mit dem multilingualen Sprach-Paket babel das Erzeugen von mehrsprachigen Literaturangaben. Dabei wird ein zusätzliches Feld language eingeführt. Für babelbib müssen Style-Files neu geschrieben werden. Der Programmcode muss um die Spalte language erweitert werden. Das Paket babelbib unterstützt nur den referenzabhängigen Ansatz, d.h. jedem Eintrag aus der Datenbank wird eine Sprache zugeordnet. Spracheigenschaften eines Dokumentes werden nicht berücksichtigt.
Ein anderes Projekt, das auf die *bst*-Sprache aufsetzt, ist biblet [Mil05]. Ziel von biblet ist es, anstatt *bbl*-Dateien HTML-Dateien zu erzeugen. Dabei werden eigene *bst*-Styles verwendet. Ein eigenes Programm übersetzt BibTeX-*bst*-Dateien in biblet-Dateien.
Die Umsetzung erweiterter Funktionalität direkt mit BibTeX ist aufgrund der Flexibilität der *bst*-Programmiersprache möglich. Ein Problem bei der Verwendung von BibTeX ist, dass es im eigentlichen Sinne keine Formatierungsstandards gibt und somit Styles mit erweiterter Funktionalität angepasst werden müssen. Änderungen an Styles müssen vom Benutzer durch Programmierung durchgeführt werden.
Eine Lösung für dieses Problem bietet das Programm makebst, beschrieben von Patrick W. Daly [Dal03]. Die Idee dahinter ist ein generischer, bibliographische Master-Style. Diese Dateien mit der Endung *msb*, werden von docstrip[2] verarbeitet. Dafür ist ein

[1] UPN - umgekehrte polnische Notation
[2] Docstrip, geschrieben von Frank Mittelbach ist ein Bestandteil von LaTeX2; Der ursprüngliche Zweck war es, unnötige Kommentarzeilen aus TeX-Dateien zu entfernen.

docstrip-Batchjob notwendig. Die Erstellung eines Batchjobs wird durch das menügeführte Programm makebst erleichtert. Beim Aufruf von makebst, wird mittels Benutzerdialog durch einzelne Formatierungsmerkmale geführt.
Das Standardmasterfile merlin.bst wurde original aus den Standard Styles plain.bst, unsrt.bst und dem File apalike.bst erstellt. Zusätzlich wurden Erweiterungen für Mehrsprachenfähigkeit [Dal07] durchgeführt. Da merlin.bst ausschließlich die Sprache Englisch und babel unterstützt, gibt es weitere Masterfiles, wie z.B. french.mbs oder german.mbs.

2.1.2 Zusätzliche LaTeX-Pakete

Einen anderen Lösungsansatz bietet bibLaTeX. Dieses LaTeX-Paket arbeitet im Gegensatz zu BibTeX mit LaTeX Macros. BibTeX-Style-Dateien können nicht verwendet werden. Das Format der bibliographischen Datenquellen ist syntaktisch ähnlich, aber durch zusätzliche Spalten problematisch anzuwenden.
Auch KBibTeX arbeitet nicht mit BibTeX, sondern mit LaTeX-Macros. KBibTeX benötigt eigene Style-Files. Der Vorteil von KBibTeX liegt im Umgang mit UTF8-Dateien.
Natbib (Natural Sciences Citations and References) [Dal09] ist eine Neuentwicklung des LaTeX \cite Kommandos. Natbib ermöglicht die Zitierform „author-year“ zusammen mit numerischen Formaten.
Das Paket amsrefs von Michael Downes und David M. Jones ist eine BibTeX ähnliche Implementierung in LaTeX und kann BibTeX ersetzen oder mit BibTeX gemeinsam eingesetzt werden[3].

2.1.3 Neuimplementationen von BibTeX

Andere Projekte beschäftigen sich mit der kompletten Neuentwicklung von BibTeX. Bibulus von T. M. Widman [Eur03] ist ein in Perl geschriebenes Projekt, das das BibTeX-Format durch eine XML-basierte Datenquelle ersetzt. Bibulus beschäftigt sich mit der Lösung klassischer BibTeX Probleme wie Sortierung nicht englischsprachiger Dokumente oder der gemeinsamen Benutzung von Bibliographien in unterschiedlichen Sprachen. Ein anderer Punkt ist die verbesserte Interaktion mit anderen Programmen, da in LaTeX und BibTeX die meisten diesbezüglichen Features hartcodiert sind. Bibulus-Datenquellen müssen im XML-Format vorliegen. Eigene Tools unterstützen die Konvertierung von *bib*-Dateien nach XML. Die Mehrsprachenfähigkeit von Bibulus wird durch die volle Unterstützung von Unicode ergänzt. Der wesentliche Nachteil von Bibulus ist, dass bestehende *bst*-Styles nicht wiederverwendet werden können. Es gibt keine Möglichkeit, *bst*-Dateien zu importieren oder zu übersetzen.

[3] http://tug.ctan.org/tex-archive/macros/latex/required/amslatex/

Das Projekt MlBibTeX wurde im Oktober 2000 gestartet und hatte eine Neuentwicklung von BibTeX mit multilingualen Features zum Ziel. Anforderung war es, eine ergonomische benutzerfreundliche Applikation zu schaffen. Da die ersten Versionen von MlBibTeX nicht 100% kompatibel mit *bib*-Dateien waren, wurde ein Tool geschrieben, welches *bib*-Dateien in ein anderes Format übersetzen kann [Huf02a]. 2003 wurde die Version 1.3 vorgestellt [Huf03b]. In dieser Version wurde die Ablöse von *bst* durch nbst implementiert. Die Sprache nbst beschreibt bibliographische Formatierung in einer XSLT-ähnlichen Syntax. MlBibTeX war ursprünglich in C programmiert. Um dem Anwender die Möglichkeit zu geben, eigene prozedurale Elemente zu entwickeln, wurde der bibliographische Präprozessor auf die Programmiersprache Scheme umgestellt [Huf05b]. Im Gegensatz zu Bibulus hat MlBibTeX BibTeX komplett abgelöst [Huf07].
MlBibTeX baut auf babelbib auf. Im Unterschied zu babelbib unterstützt MlBibTeX auch den dokumentenabhängigen Ansatz [Huf02b]. MlBibTeX löst sowohl bestehende *bib*-Dateien, als auch *bst* Styles ab (nbst). Dabei wird versucht, *bst* Dateien in Scheme Code zu übersetzen. Nachteil ist der fehlende Lösungsansatz für die Unicode-Unterstützung. Problematisch bleibt die Sortierung bei mehrsprachigen Dokumenten und bei nationalen Sonderzeichen. Die Modularität der Style Dateien ist weiterhin nicht gegeben. Es gibt keine Übersetzung von *bst*-Dateien in *nbst*-Dateien. Im Gegensatz zu BibTeX ist MlBibTeX nicht für alle gängigen Plattformen verfügbar.
Eine Chronologie der Entwicklungsstufen und Erläuterungen zu den jeweiligen Architekturüberlegungen wird in [Huf07] angeführt.
BibTeX++ [KD03], ein Projekt der Universität ENST Bretagne, ist eine Neuimplementierung von BibTeX. BibTeX++ behält im Wesentlichen den BibTeX Workflow bei. Der Kern wurde aber in Java komplett neu entwickelt. Dabei bleibt die Frage offen, ob für einen Style Designer Java einfacher in der Bedienung ist als *bst*.
Biblatex von [Leh09] ist eine Neuentwicklung, die das Einbinden von Bibliographien unter LaTeX unterstützt. BibTeX wird für die Sortierung und Erzeugung der Labels verwendet. Die Formatierung wird durch LaTeX Macros unterstützt.Cl-BibTeX[4] von Mattias Koeppe ist eine Neuimplementierung von BibTeX in Common Lisp. Durch die Möglichkeit, Formatierungsfunktionen in Lisp anstelle *bst* zu schreiben, soll die Bearbeitung von bibliographischen Stilen erleichtert werden.

2.1.4 Konvertierungstools für BibTeX

Das sind im Wesentlichen Projekte, die sich mit der Konvertierung von BibTeX in andere Formate beschäftigen. Bibutils von Chris Putnam übersetzt BibTeX-Daten in das MODS-Format[Gar03]. Applikationen, die BibTeX nach HTML, XML exportieren oder

[4] http://www.nongnu.org/cl-bibtex/

importieren, sind zum Beispiel BibTeXML, BibTEX, bib2xhtml, bibtex2html, BibTex-XML-HTML oder bib2html.

2.2 Neuentwicklungen im Umfeld LaTeX Thebibliography Environment

In diesem Kapitel wird auf die Erweiterungen bibliographischer Neuimplementierungen eingegangen. Dabei werden folgende Eigenschaften vorrangig betrachtet:

- Mehrsprachenfähigkeit
- Benutzergerechte Bearbeitungsmöglichkeiten von Styles
- Erweiterte Schnittstellen

Vor und Nachteile sollen gegenübergestellt werden und Methoden für die Weiterverwendung bzw. Weiterentwicklung sollen betrachtet werden.

2.2.1 BibLaTeX

BibLaTeX ist ein LaTeX Paket, das erweiterte bibliographische Funktionen bietet. Dieses Paket ist eine komplette Neuimplementierung der bibliographischen Funktionalität von LaTeX. BibTeX wird nur zum Sortieren der Literatureinträge und zum Erzeugen der Labels verwendet. [Leh09]
BibLaTeX ist praktisch gesehen eine Erweiterung von BibTeX. BibLaTeX wird als Paket in ein LaTeX Dokument eingebunden. Anstelle von bst- Dateien wird die Formatierung der bibliographischen Einträge mit LaTeX-Macros gesteuert. Dadurch ist es möglich, ohne Kenntnisse der BibTeX postfix Stack Sprache Zitierstile anzupassen. Im Gegensatz dazu sind sehr gute LaTeX Kenntnisse notwendig.
Damit das bibLaTeX Paket geladen wird, muss in der LaTeX Datei folgender Eintrag eingefügt werden:

```
\usepackage[style=authortitle-icomp]{bibLaTeX}
```

Der verwendete Zitierstil wird mit dem Parameter Style angegeben. BibLaTeX unterstützt eine Schnittstelle zum babel- Paket, wodurch länderspezifische Formatierungen ermöglicht werden.
Beispiel Laden des babel-Pakets:

```
\usepackage[babel,german=parameter]{qsquotes}
```

Empfohlen wird das Paket qsquotes. Wenn dieses verfügbar ist, arbeitet bibLaTeX mit sprachabhängiger Formatierung der Anführungszeichen. Anderenfalls wird das englisch amerikanische Format gewählt. Das babelbib-Paket, das mehrsprachige Literaturverzeichnisse unterstützt, ist zu bibLaTeX nicht kompatibel. Multilinguale Bibliographien sind ein Standardfeature von bibLaTeX .

Folgende LaTeX-Pakete sind zu bibLaTeX inkompatibler:

- backref - erzeugt Rückreferenzierungen
- bibtopic - unterstützt nach Themen unterteilte Bibliographien
- chapterbib - unterstützt die Aufteilung des Literaturverzeichnisses auf bestimmte Bereiche
- jurabib - Formatierung für juristische Dokumente
- natbib - unterstützt die Kombination Autor- Jahr und numerische Zitierungen

Äquivalent zu BibTeX werden die bibliographischen Dateien eingebunden:

```
\bibliography{ Liste Dateinamen}
```

BibLaTeX verwendet im Vergleich zum BibTeX-Standard, ein erweitertes Format für bibliographische Dateien. Der Aufbau und die Syntax sind gleich wie in einer *bib*-Datei. Das Format wurde um neue Feldtypen, wie z.B. sortname, shorttitel, subtitel, erweitert. Eintragstypen sind ähnlich wie bei Standard-BibTeX bibliography Styles, vgl. [Leh09] S. 9-11. Neben den regulären Eintragstypen wie article, book, inproceedings, ect., gibt es noch eine Vielzahl von Eintragstypen, die eine Erweiterung zu Standard-BibTeX bibliography Styles darstellen. Diese werden ähnlich wie bei BibTeX von den Standardstilen nicht unterstützt.
Eintragsfelder werden in optionale und Pflichtfelder unterteilt. Vergleicht man die Liste [Leh09] aller Eintragsfelder, erkennt man, dass bibLaTeX wesentlich erweiterte Funktionen im Vergleich zu BibTeX anbietet.

BibLaTeX unterstützt unterschiedliche Arten von Datentypen:

- **Name lists** - werden geparst und in einzelne Teile zerlegt. Typisch dafür sind die Felder author und editor. Die Besonderheit ist, dass einzelne Namen in die Bestandteile Suffix, Vorname, Nachname und Präfix zerlegt werden (ähnlich wie BibTeX Funktion format.names$)

- **Literal lists** - wWerden geparst und zerlegt, als Separator dienen die Wörter „and“ oder „others“.
- **Key lists** - Dieser Datentyp beinhaltet eine Liste von druckbarem Text, abhängig von einem Schlüsselwert. Ein typisches Beispiel ist das Feld language.
- **Fields** - Der gesamte Feldinhalt wird ausgegeben. Dabei wird zwischen Range fields, Integer fields, Literal fields, Verbatim fields, Key fields und Special fields unterschieden.

Der Datentyp language ist ein Key list Feld und spezifiziert die Sprache der zitierten Bibliographie. BibLaTeX kann sprachtypische Merkmale wie Silbentrennung mittels babel-Paket bewerkstelligen.

Unterstützte Sprachen:

Sprache	Region/Dialekt	babel identifier
Danish	Denmark	danish
English	USA american	USenglish, english
	United Kingdom british	UKenglish
	Canada	canadian
	Australia	australian
	New Zealand	newzealand
French	France, Canada	french, francais, canadien
German	Germany	german, ngerman
	Austria	austrian, naustrian
Italian	Italy	italian
Norwegian	Norway	norsk, nynorsk
Portuguese	Brazil	brazil
	Portugal	portuges
Spanish	Spain	spanish
Swedish	Sweden	swedish

Tabelle 2.2: BibLaTeX Sprachen

Bibliographische Zitierstile

BibLaTeX wird mit eigenen Zitierstilen ausgeliefert. Diese sind speziell für bibLaTeX implementiert. BibTeX Bst- Dateien können nicht verwendet oder konvertiert werden. Zitierstile können abgeändert oder neu erstellt werden. Der BibLaTeX Stil numeric entspricht der Standard Formatierung von LaTeX. Alphanumerische Zitierung wird mit dem Stil alphabetic durchgeführt, dieser entspricht dem klasischen BibTeX Style alpha.bst.
Im Unterschied zu BibTeX unterscheidet bibLaTeX zwischen bibliographischem Stil (Literaturverzeichnis) und Zitierstiel (Zitierschlüssel).

Beispiel Verwendung von bibLaTeX:
`bibstyle=<file>`
`citestyle=hfilei`

Der Zitierstil wird über eine Option beim Laden des Packages angegeben. Dabei handelt es sich um eine sogenannte „load time option“. Der Dateiname wird mit der Endung *.bbx* angegeben.
Im Gegensatz zu BibTeX wird das Sortierverhalten nicht ausschließlich über die Stil Datei angegeben, sondern ist eine eigene Option, eine sogenannte „Preamble Option“. Diese Optionen können beim Laden des Pakets angegeben werden:

`sorting=nty, nyt, nyvt, anyt, anyvt, ynt, ydnt, debug, none`

Die Sortieroption kann zur Laufzeit, z.B. beim Laden des Zitierstiles, geändert werden.

Sortieroptionen:
nty - Sort by name, title, year.
nyt - Sort by name, year, title.
nyvt - Sort by name, year, volume, title.
anyt - Sort by alphabetic label, name, year, title.
anyvt - Sort by alphabetic label, name, year, volume, title.
ynt - Sort by year, name, title.
ydnt - Sort by year (descending), name, title.
debug - Sort by entry key. This is intended for debugging only.
none - Do not sort at all. All entries are processed in citation order.

Ein bibliographischer Stil ist eine Zusammenstellung von Macros und ist in einer Datei mit der Endung *.bbx* gespeichert.

Eine *bbx*-Datei hat folgende Struktur:

```
\ProvidesFile{example.bbx}[2006/03/15 v1.0 bib\LaTeX\ bibliography
  style]
\renewenvironment*{thebibliography}{...}{...}
\renewenvironment*{theshorthands}{...}{...}
\renewcommand*{\thebibitem}{...}
\renewcommand*{\thelositem}{...}
\InitializeBibliographyStyle{...}
\DeclareBibliographyDriver{article}{...}
\DeclareBibliographyDriver{book}{...}
\DeclareBibliographyDriver{inbook}{...}
...
\DeclareBibliographyDriver{shorthands}{...}
\endinput
```

Über die Anweisung:

```
\DeclareBibliographyDriver{<type>}{<code>}
```

können die unterschiedlichen Eintragstypen definiert werden. Der Typ entspricht dem jeweiligen Eintragstyp aus der .bib Datei. Der <code> Eintrag ist ein beliebiger LaTeX Macro Code.

2.2.2 KBibTeX

KBibTeX ist ein BibTeX Editor für KDE. KBibTeX arbeitet flexibler als BibTeX im Umgang mit Unicode. Wenn KBibTeX das Encoding nicht feststellen kann, geht das Programm von UTF8 aus. Wenn UTF8 encoded BibTeX Files verwendet werden, muss das *tex*-File auch in UTF8 encoded sein. Bibliographische Styles werden als Set von Macros in Style-Files *.bbx* abgelegt. Die Formatierung erfolgt über LaTeX-Macros und nicht mehr über BibTeX .

2.2.3 MlBibTeX

MlBibTeX ist eine Neuentwicklung von BibTeX. MlBibTeX steht für 'MultiLingual BibTeX'. Um multilinguale Inhalte und Funktionen zu ermöglichen wurden die Entitäten um das Attribut [LANGUAGE] erweitert.
Das Ziel von MlBibTeX ist die Vereinfachung der Bearbeitung und Verwaltung bibliographischer Stile. Es werden Methoden dargelegt und untersucht, um bibliographische

Stile in der Sprache nbst zu schreiben. Da nbst mit XSLT nahe verwandt ist, kann diese Anleitung auch für die Programmierung der Styles in XSLT helfen.[5]

Erweiterungen von MlBibTeX

MlBibTeX verfolgt zwei Ansätze für mehrsprachige Bibliographien.

Der referenzabhängige Ansatz

Dieser Ansatz bezieht sich auf die verwendete Information. Jeder Eintrag einer bibliographischen Datenbank wird mit einer Sprache deklariert. Damit wird die Sprache mit dem zitierten Dokument in Verbindung gebracht und nicht mit dem Dokument, in dem es zitiert wird. Dabei wird versucht, Standardfelder in die richtige Sprache zu übersetzen. So wird zum Beispiel ein Monatseintrag „july“ mit „Juli“ ausgegeben, wenn der Eintrag als 'german' deklariert ist.

Der dokumentabhängige Ansatz

Ein anderer Ansatz bezieht sich auf die geschriebene Arbeit. Diesem Ansatz zufolge müssten alle Literaturangaben in einem deutschsprachigen Dokument in deutschsprachiger Formatierung ausgegeben werden. Unter Verwendung von BibTeX würde das bedeuten, einen entsprechenden bibliographischen Stil zu verwenden bzw. zu adaptieren und Einträge in den bibliographischen Datenbanken an die jeweilige Sprache anzupassen. Das würde z.B. bedeuten, alle Einträge im Feld note (Anmerkung) in die jeweilige Sprache zu übersetzen, folglich für ein und denselben Eintrag einen zweiten Eintrag zu erzeugen.
Die Wahl eines dieser Ansätze sollte nicht vom bibliographischen Programm vorgegeben sein, sondern vom Autor. Für jeden dieser Ansätze existiert eine Vielzahl an LaTeX Paketen. Den referenzabhängigen Ansatz unterstützen zum Beispiel das mlbib und babelbib Paket, unter Verwendung eines Feldes mit der Bezeichnung „language“.
Der dokumentenabhängige Ansatz wird u.a. vom Oxford Paket unterstützt, das dem Anwender die Sprachauswahl für die gesamte Bibliographie ermöglicht.
MlBibTeX unterscheidet zwischen den Methoden „language changes“ und „language switches“ [Huf01].
Für dendokumentenabhängigen Ansatz werden Sprachschalter („Language switches“) verwendet. Dabei wird zwischen zwei verschiedenen Arten von Schaltern unterschieden: Zwischen denen, die eine Standardsprache verwenden und jenen ohne Standardsprache.

[5] XSLT, ist eine Programmiersprache zur Transformation von XML-Dokumenten.

Durch syntaktische Regeln wird festgelegt, wie einem Feld eine bestimmte Sprache zugeordnet wird.

Im folgenden Beispiel werden die Einträge für das Feld note auf Englisch, Französisch und Deutsch angegeben. Wenn ein Dokument in einer anderen der drei angeführten Sprachen kompiliert wird, wird der Eintrag der Sprache Englisch verwendet, da sie über die Definition - LANGUAGE = english - dieser Bibliographie als Standardsprache zugeordnet wurde.

Beispiel für einen Sprachschalter mit einer Standardsprache:

```
@BOOK{king1982f,
AUTHOR = {Stephen~Edwin King},
TITLE = {The Running Man},
NOTE = {[Written as] * english
[Sous le pseudonyme de] * french
[Unter der Pseudonym] * german
Richard Bachman},
PUBLISHER = {New American Library},
YEAR = 1982,
MONTH = may,
LANGUAGE = english}
```

Beispiel für einen Sprachschalter ohne einer Standardsprache:

```
@BOOK{gibson1986,
AUTHOR = {William Gibson},
TITLE = {Burning Chrome and Other Stories},
NOTE = {[Titre de la traduction fran\c{c}aise :
        \emph{Grav\'{e} sur chrome}] ! french
        [Titel der deutschen \'{U}bersetzung: \emph{Cyberspace}] !
          german},
PUBLISHER = {Victor Gollancz, Ltd.},
YEAR = 1986,
LANGUAGE = english}
```

In diesem Fall wird, wenn die referenzierende Sprache nicht übereinstimmt, keine Ausgabe für das Feld note generiert.
Für den Ansatz „Language change“ werden Textelemente für den sprachspezifischen Austausch durch eine spezielle Syntax gekennzeichnet:
`[string] : sprache`

Implementierung

MlBibTeX ist in der Sprache C geschrieben. Dabei wurden speziell die Entwicklungswerkzeuge von GNU eingesetzt. Der Scanner und der Parser wurden mit den GNU-Werkzeugen flex und bison entwickelt.
Strategie und Entscheidungen, wie die Neuimplementierung von MlBibTeX erfolgen soll: Nachdem eine komplette Neuentwicklung beschlossen wurde, fiel die Auswahl der Programmiersprache auf C. Ein wichtiger Entscheidungsgrund waren neben Effizienz die Lesbarkeit und die Wartbarkeit des Programms. Die Überlegungen wurden auch hinsichtlich der Neuimplementierung von TeX selbst bestimmt. Neuimplementierungen von TeX wie NTS (New Typesetting System) verwenden Java, eine objektorientierte Technologie. Berichten zufolge ist das neue Programm über 100 mal langsamer als TeX [Huf02a]. Ein Prototyp wurde in C programmiert, um ihn mit BibTeX vergleichbar zu machen. Die Entscheidung fiel auf C, da C sehr effizient arbeitet und auf fast allen Plattformen verfügbar ist. Nachteil der Programmiersprache C ist, dass Entwickler von bibliographischen Styles mit der Sprache umgehen müssen. Da man von Style Designern kaum Erfahrung im Umgang mit der Sprache C erwarten kann, ist dieser Ansatz unpraktikabel.
Für die erste veröffentlichte Version (1.3) wurde mlBibTeX in Scheme entwickelt. Diese Vorgangsweise ermöglicht die Darstellung von SXML[6] als Scheme Implementierung des XML-Baumes.

Ein multilingualer Eintrag einer Bibliographie unter MlBibTeX 1.2 hatte folgendes Aussehen:

```
@BOOK{howard1969,
    AUTHOR = {Robert Ervin Howard, abbr => R. with
              first => Lyon Sprague, von => de, last => Camp, abbr =>
                L. Sprague with
              Lin Carter}
    TITLE = {Conan of {Cimmeria}},
    PUBLISHER = {Ace Books},
    ADDRESS = {New York, New York},
    NOTE = {[Titre de la traduction fran\c{c}aise : 'Conan le Cimm\'{e
      }rien'] ! french
             [Titel der deutschen \'{U}bersetzung: 'Conan von Cimmerien
               '] ! german}
   YEAR = 1969,
  LANGUAGE = english}
```

[6]Eine Sprache wie SXML zur Definition anderer Sprachen nennt man Metasprache. SXML ist eine Möglichkeit, XML-Datei zu schreiben und zu verarbeiten.

Eine Besonderheit der Version war die explizite Schreibweise des Namensfeldes, wobei die BibTeX typische Zerlegung in first, last, von und junior bereits in den bibliographischen Daten über benannte Parameter dargestellt wurde.

```
first => Lyon Sprague, von => de, last => Camp
```

Außerdem wurden formatierte Abkürzungen direkt in der Bibliographie abgebildet.

```
abbr => L. Sprague with Lin Carter
```

Die Transformation in einen XML-Baum:

Listing 2.1: Funktion book in XML

```
<book id="howard1969" language="english">
  <author>
    <name><personname><first abbr="R.">Robert Ervin</first><last>Howard</last></
      personname></name>
    <with/>
    <name>
      <personname>
        <first abbr="L. Sprague">Lyon Sprague</first><von>de</von><last>Camp</last>
     </personname>
    </name>
    <with/>
    <name><personname><first>Lin</first><last>Carter</last></personname></name>
  <title>
    Conan of <asitis>Cimmeria</asitis>
    <!-- asitis is for a group of words that should not be case-converted. -->
  </title>
  <publisher>Ace Books</publisher>
  <year>1969</year>
  <address>New York, New York</address>
  <note>
  <group language="french">
    Titre de la traduction française : <emph emf="yes" quotedbf="yes">Conan le
      Cimmérien</emph>
  </group>
  <group language="german">
    Titel der deutschen Übersetzung: <emph emf="no" quotedbf="yes">Conan von Cimmerien
      </emph>
  </group>
  </note>
</book>
```

Darstellung in Scheme SXML:

```
(book (@ (id 'howard1969'
             language='English'))
          (author ....)
          (title  ....)
          ....)
```

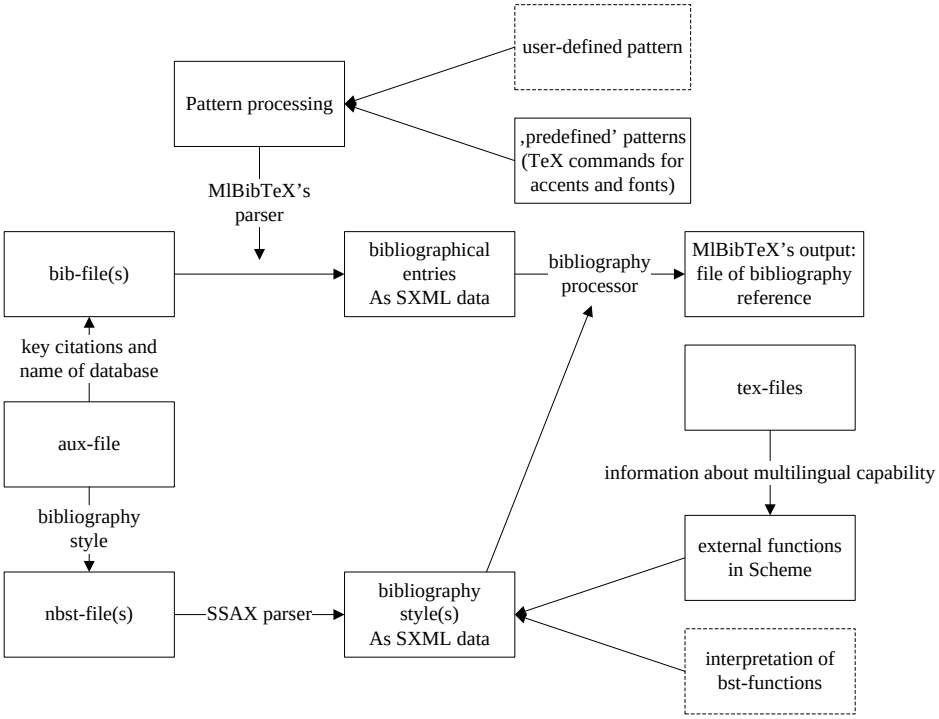

Abbildung 2.1: Arbeitsablauf von MlBibTeX[Huf05b]

Architektur

Ausgegangen wird wie beim herkömmlichen BibTeX von einer *aux*-Datei, die von LaTeX erstellt wird, eine Liste der zitierten Einträge enthält sowie eine oder mehrere *bib*-Dateien, die die bibliographischen Einträgen enthalten. Der MlBibTeX Parser erzeugt daraus eine Liste bibliographischer Einträge, abgebildet in SXML. Das Ergebnis ist ein XML-Baum, der dem DOM-Modell[7] entspricht. Der Parser arbeitet mit Mustern. Neben den vordefinierten Mustern können auch benutzerdefinierte Muster verwendet werden.
Die *aux*-Datei beinhaltet auch die Information über den verwendeten Zitierstil. Wenn der *bst*-Kompatibilitätsmodus nicht verwendet wird, wird die entsprechende Formatierung mittels *nbst*-Datei durchgeführt. *Nbst*-Dateien enthalten analog zu den *bst*-Formatierungsfunktionen Templates, die (ähnlich XSLT) Formatierungstags enthalten [Huf04a].
Diese Dateien werden SSAX[8] geparst, gruppiert und vorkompiliert. Das Ergebnis ist der bibliographische Stil in einer SXML-Datei abgebildet [Huf04b].
Abbildung 2.1 zeigt den Arbeitsablauf von MlBibTeX.
Jedes Template der *nbst*-Datei resultiert in einer Scheme-Funktion[9]. Bei dieser Übersetzung wird auf externe Quellen zugegriffen, die Scheme-Interpretationen von *bst*-Funktionen beinhalten. *Nbst*-Dateien können auch externe Aufrufe von Scheme-Funktionen beinhalten. Diese bilden multilinguale Methoden von *TeX*-Dateien ab, können aber auch benutzerdefiniert sein. Externe Prozeduren in Scheme werden für String-Operationen verwendet, sprachabhängiges Sortieren und um *TeX*-Dateien auf mehrsprachenfähige Elemente zu durchsuchen.

[7] DOM (Document Object Model) ist eine vom World Wide Web Consortium (W3C) standardisierte Schnittstelle, die den Zugriff auf einzelne Elemente einer XML-Datei ermöglicht. Die DOM-Schnittstelle ermöglicht in Form einer baumartigen Struktur den Zugriff auf verschiedene unabhängige Teile des Datensatzes. Dieser wahlfreie Zugriff steht im Gegensatz zum sequenziellen Einlesen einer XML-Datei. Da die Datei immer komplett in den Speicher eingelesen wird, ist DOM eine sehr speicherintensive Methode. Im Gegensatz dazu liest SAX immer nur kleine Teile der XML-Datei ein. Ein wahlfreier Zugriff ist aber nicht möglich. Technisch gesehen ist DOM ein Parser, der eine XML-Datei in einen sogenannten Baum aufarbeitet. Ein Baum besteht aus verschiedenen Knoten, jeder dieser Knoten enthält verschiedene Referenzen auf benachbarte Knoten.

[8] SSAX ist ein XML-Parser-framework für Scheme Programme, basierend auf s-expressions.

[9] Scheme ist einer der beiden wichtigsten Dialekte der Programmiersprache Lisp. Im Gegensatz zu Common Lisp, dem anderen großen Dialekt, folgt Scheme einer minimalistischen Design-Philosophie. Der Begriff S-Expression (Symbolische Ausdrücke) bezieht sich auf eine Konvention zur Darstellung semi-strukturierter Daten in lesbarer Textform. Symbolische Ausdrücke werden meist von Symbolen und Listen gemacht.

Nbst bibliographische Styles

Für die bibliographischen Styles benutzt MlBibTeX seit der Version 1.3 die Sprache *nbst*, die mit XSLT nahe verwandt ist [Huf03a]. Mit dieser Version wird *bst* durch *nbst* (new bibliography style language) abgelöst. Der Hauptunterschied zwischen nbst und XSLT ist die Implementierung von mehrsprachigen Eigenschaften. MlBibTeX liest *bib*-Dateien wie XML-Bäume. Bibliographische Styles können auf die jeweiligen Feldwerte mittels Path Expressions, ähnlich der XPath Sprache, zugreifen.
»*We can be asked for a question: why two languages: nbst and Scheme? Why have we not used Scheme for the whole of a bibliography style? Such a conventions would have made MlBibTeX close to the stylesheets written in DSSSL, associated with SGML texts. But it was told that programming with DSSSL was difficult for style designers that are not experienced programmers. In fact, DSSSL is not declarative enough, if we compare it to XSLT or nbst. Besides, nbst allows refinements to be put into action without modifying an existing style directly.*«[Huf05b] Im Gegensatz zu anderen Neuimplementierungen, wie BibTeXML oder BIB2XML, verwendet mlBibTeX nur eine XSLT ähnliche Sprache (nbst). Die Ähnlichkeit mit XSLT orientiert sich an einem leicht zu erlernenden Standard, ist aber nach außen hin offen und flexibel erweiterbar. Dadurch wird auch das Einbinden von externen, in Scheme geschriebenen Modulen für die komplexen multilingualen Textformatierungen erleichtert.
Ein Beispiel einer *nbst*-Datei ist im Listing 2.2 abgebildet. Die *bst*-Funktion {format.date} wird als *nbst*-Template <nbst:template name='format.date'> dargestellt.

Listing 2.2: Beispiel: nbst.xslt

```
<nbst:template name="format.date">
  <nbst:choose>
    <nbst:when test="year">
      <nbst:choose>
        <nbst:when test="month">
          <nbst:apply-templates select="month"/>
          <nbst:text> </nbst:text>
          <nbst:value-of select="year"/>
        </nbst:when>
        <nbst:otherwise>
          <nbst:value-of select="year"/>
        </nbst:otherwise>
      </nbst:choose>
    </nbst:when>
    <nbst:otherwise>
      <nbst:if test="month">
        <nbst:warning>
            There's a month but no year in
          <nbst:value-of select="@id"/>
        </nbst:warning>
```

```
          <nbst:apply-templates select="month"/>
        </nbst:if>
      </nbst:otherwise>
    </nbst:choose>
</nbst:template>
```

Die Felder year und month entsprechen den selektierten Werten aus der *bib*-Datei. Wenn year und month leer sind, dann wird ' ' (NULL) auf den Stack (entspricht einem Funktionsrückgabewert) geschrieben. Wenn nur year leer ist, wird die Warnung (Kommando warning$) 'there's a month but no year in ' mit dem entsprechenden Zitierschlüssel cite$ zusammengehängt. Das *bst*-Program greift durch das interne Kommando ITERATE auf die Werte der *bib*-Datenbank zu. Dabei werden die bibliographischen Einträge hintereinander abgearbeitet. Einstiegspunkt ist immer eine Funktion, deren Name dem bibliographischen Typ entspricht (z.B. FUNCTION {book}). Auf die einzelnen Feldwerte wird direkt mit dem Namen (z.B year) wie mit einer gewöhnlichen Variable zugegriffen.
In *nbst* sind bibliographische Einträge entries und deren Feldwerte Knoten eines Baumes. Im klassischen XSLT kann auf solche Knoten rekursiv mittels xsl: apply-templates Tags zugegriffen werden: `<nbst:apply-templates select='month'/>`
Um *nbst*-Programme mehrsprachenfähig zu machen werden Templates implementiert, die die verwendete Sprache ermitteln;
`<nbst:apply-templates use-languge=\$document-language/>`

Zum Vergleich die Funktion {format.date} aus dem Style plain.bst:

Listing 2.3: plain.bst (265-281)

```
265 FUNCTION {format.date}
266 { year empty$
267     { month empty$
268   { "" }
269   { "there's a month but no year in " cite$ * warning$
270     month
271   }
272       if$
273     }
274     { month empty$
275   'year
276   { month " " * year * }
277       if$
278     }
279   if$
280 }
```

Wie in XSLT können xsl:template und xsl:apply-templates optional mode-Attribute verwenden. Mode erlaubt es, Elemente mehrmals mit unterschiedlichen Ergebnissen durch-

zuführen. Wird das mode Attribut angegeben, werden nur Templates mit genau diesem mode-Attribut verwendet. Der Wert kann für bestimmte Sprachen definiert werden. So kann genau angegeben werden, wann welcher Knoten verarbeitet werden soll.

Verwendung von MlBibTeX

Aus der Sicht des Benutzers verhält sich MlBibTeX gleich wie BibTeX. MlBibTeX wird mit dem Namen der TeX-Datei als Parameter gestartet. Die *TeX*-Datei beinhaltet die Definition des bibliographischen Stils:

```
{\bibliographystyle{S}}
```

Wenn die angeführte Stil-Datei nicht in der Version nbst verfügbar ist, wird auf eine namensgleiche *bst*-Datei zurückgegriffen [Huf05c]. MlBibTeX benötigt eine aktuelle Version von LaTeX2E inklusive einer Version des multilingualen babel-Paketes.

Kompatibilität

Ziel von MlBibTeX war es, eine benutzerfreundliche ergonomische Anwendung zu schaffen. Der Endbenutzer soll keine komplexen Änderungen an den Dateien vornehmen müssen. Da eine Viehlzahl von *bib*-Dateien bereits existieren, ist es eine Notwendigkeit, dass MLBibTeX diese auch unterstützt. MlBibTeX unterstützt die Wiederverwendung der „alten“ *bib*-Files mit Ausnahme der syntaktischen Bedeutung der eckigen Klammern. Anstelle von *nbst*-Styles können auch mit dem sogenannten „Kompatibilitätsmodus“ BibTeX *bst*-Dateien verwendet werden. Dieser Modus wird jedoch nicht empfohlen, da er die Funktionalität von MlBibTeX auf BibTeX einschränkt. Es gibt Möglichkeiten, *bst*-Dateien nach *nbst* zu konvertieren. Es wird aber empfohlen, *nbst*-Dateien komplett neu zu implementieren. [Huf05a].

Übersetzung von *bst* nach *nbst*

Eine kleine Änderung in einer *bst*-Datei durchzuführen ist eine leichte Aufgabe. Einen kompletten Style zu interpretieren und neu zu schreiben ist deutlich schwieriger. Dabei muss schrittweise analysiert werden, welche Werte im Stapelspeicher liegen. Die Sprache *bst* ist nicht modular, sie ist zwar durch Funktionen strukturiert, Parameter und Rückgabewerte werden über den Stapelspeicher abgehandelt. Das bedingt, dass fast

ausschließlich globale Variablen verwendet werden. Im Gegensatz zu bst, einer stapelorientierten Sprache, ist nbst eine regelbasierte Sprache.

» Some statements of bst are not really translatable into nbst: for example, the assignment (':='), because nbst is like a purely functional language, in the sense that a variable or a parameter cannot be changed, once it has been given a value« [Huf05a]

Im Gegensatz dazu erlaubt *nbst* rekursive Templates für die iterative Programmierung, die die while$ Funktion von *bst* übersetzt. Ein anderes Problem ist die Vielzahl an speziellen *bst*-Funktionen, die mit der Stapelbearbeitung oder dem direkten Zugriff auf bibliographische Einträge zu tun haben. Diese Funktionen können nicht immer ein Gegenstück in XSLT haben,z.B.: call.type$, format.name$, while$, skip$, duplicate$, swap$, top$.
Hufflen wählt den Ansatz des Reverse Engeneering. Im Gegensatz zu Re-engeneering (Transformation einer Sprache in eine andere Sprache), wird die Software analysiert und deren Design spezifiziert. Tabelle 2.3 [Huf05a] zeigt einige Übersetzungen von *bst*-Funktionen nach *nbst*.

bst-Ausdruck	Übersetzung nach *nbst*
I1 I2 >	I1 > I2
I1 I2 =	I1 = I2
I1 I2 +	I1 + I2
S ttchange.case$	concat(substring(S,1,1),lowercase(substring(S,2)))
S llchange.case$	lowercase(S)
S chr.to.int$	(char->integer S)
cite$	@id
L empty$	not(string(L))
L missing$	not(L)
newline$	<nbst:text>&eol;</nbst:text> or <nbst:value-of select="&eol;">
S num.names$	count(name) if name(S) 2 author, editor
preamble$	@preamble
S purify$	call(bst-purify,S)
quote$	<nbst:text>"</nbst:text> or <nbst:value-of select=""">
S I1 I2 substring$	substring(S,string-length(S) + I1 - I2 + 2,I2) if I1 < 0
S text.length$	string-length(S)
S I text.prefix$	substring(S,1,I)
type$	name()
S width$	(tex-width S)
S write$	<nbst:value-of select="S">

Tabelle 2.3: Übersetzungen von *bst*-Funktionen nach *nbst*

Die direkte Übersetzung einiger *bst*-Funktionen benötigt den Aufruf von externen Scheme-Funktionen. Die Schnittstelle zu Scheme erfolgt über path expressions, die die Aufrufe der externen Prozeduren beinhalten:
Call(function-name, arg1, ..., argn) Zum Beispiel die Funktion purify$: call(bst-purify,\)
Es gibt mehrere Gründe warum externe Funktionen notwendig sind, z.B. Operationen, die nur schwer oder sehr umständlich mit Funktionen der XPath-Bibliothek umzusetzen sind. Ein Beispiel dafür ist die Funktion purify$. Diese interne *bst*-Funktion bearbeitet Textketten auf eine BibTeX spezifische Art. Sie entfernt alle nicht-Dezimalzeichen, Ausnahme sind Zeichen des Alphabets, wenn sie nacheinander gereiht sind, oder nach einer Dezimalzahl kommen.

Listing 2.4: Beispiel einer externen Funktion in Scheme implementiert

```
(define (bst-purify string-0)
  (let thru ((index (- (string-length string-0) 1))
             ;; Current index, we are going backward.
             ;;The second argument allows us to accumulate retained
             ;; characters in a list, we begin with an empty list:
           (accumulator '()))
       (if (negative? index)
           ;; The string has been processed,
           ;;we convert the list of accumulated characters into a string:
           (list->string accumulator)
           (thru (- index 1) (let ((current-char (string-ref string-0 index)))
                               ;; Discarding it if it is not alphanumeric:
                               (if (or (char-alphabetic? current-char) (char-numeric?
                                 current-char))
                                   (cons current-char accumulator)
                                    accumulator))))))
```

Purify Funktion in Python implementiert (Listing: 2.5)

Listing 2.5: Python-Funktion bst_purify in render.py

```
1  def bst_purify(inStr):
2
3      outStr = ''
4      printOut = 'N'
5      for c in inStr:
6          if re.match(r"\d", c):
7              outStr =outStr+c
8              printOut='Y'
9          elif re.match(r"\w", c):
10             if printOut == 'Y':
11                 outStr =outStr+c
12         else:
13             printOut = 'N'
```

Ein anderer Grund für die Implementierung von Scheme sind spezifische Sortierroutinen, wie zum Beispiel das Sortieren von Monatsnamen. Spezielle TeX-Features, wie Bestimmen der Länge einer Textkette, oder Öffnen und Durchsuchen von externen Dateien, sind in Scheme einfacher umzusetzen.

2.2.4 Biblet

Biblet ist ein Tool, das mit Hilfe von bibliography Styles (*bst*) XHTML Dateien aus BibTeX Datenfiles (*bib*) generiert. Biblet ist in der „BibTeX Stack language“ geschrieben. Dadurch ist Biblet zu jedem System, das mit BibTeX läuft, portabel.
Biblet ist ein eigener, spezieller bibliographischer Style, dessen Output (*.bbl*-File) eine

HTML-Datei ist. Biblet kann dafür verwendet werden, bibliographische Datenbanken, gespeichert im BibTeX-Format *bib* in einem HTML Browser anzuzeigen. Biblet ist somit keine Neuimplementierung von BibTeX, sondern verwendet BibTeX in einer abgewandelten Weise. Einerseits ist biblet ein gutes Beispiel dafür, wie offen die BibTeX *bst*-Sprache ist, andererseits welchen Einschränkungen BibTeX unterliegt.
Ein interessanter Teil von biblet ist das Modul „XML to bst“, das LaTeX Symbole nach Unicode konvertiert.
Abbildung 2.2 zeigt den Arbeitsablauf von bilet. Die Tatsache, dass BibTeX ausschließlich in ein File schreibt, welches die Endung *.bbl* hat, zeigt auf, wie stark BibTeX an LaTeX gebunden ist.
Da die *bst*-Sprache keine Arrays unterstützt, ist es unmöglich, statische Übersetzungen zu implementieren (Mapping-Tabellen). Biblet benötigt aber die Funktionalität, LaTeX Symbole in HTML Entitäten zu übersetzen. Die Möglichkeit Such- und Ersetzfunktionen zu schreiben, scheitert daran, dass BibTeX Funktionen auf hundert Token limitiert sind. Da jede Ersetzung drei Token benötigt (Suchen, Ersetzen und Aufruf der Funktion) können 33[10] Ersetzungen pro Funktion umgesetzt werden. Bei über 500 Symbolen ergibt das über 70 Funktionen. Um diese nicht per Hand zu programmieren, wurde ein Generator (trans.bst) entwickelt, der ein *bst*-Programm (*trans.bst*) generiert.

Listing 2.6: Beispiel-Output von Biblet

```
<!DOCTYPE html PUBLIC '-//W3C//DTD XHTML 1.0 Strict//EN'
  'http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd'>
<html xmlns='http://www.w3.org/1999/xhtml' xml:lang='en' lang='en'>
  <head><title>My publications</title></head>
  <body>
    <h1>My publications</h1>
    <div class='bib-bibliography'>
      <h2 class='bib-year' id='year-2005'>2005</h2>
        <ul>
          <li class='bib-bibitem' id='cite-m05toc'>
            <div class='bib-article'>
              <p>
                <span class='bib-author'>Tristan Miller.</span>
                <span class='bib-title'>The tyranny of copyright.</
                  span>
                <em>Imagine</em>, 4(1):1,8&ndash;11, May 2005.
                ISSN 1710-5994.
              </p>
        ...
</html>
```

[10] 33 ist ein technisches Limit

Modul „XML to bst“

Als Datenquelle dient die XML-Datei[11], *ent.xml* („Vidar Bronken Gundersen and Rune Mathisen's comprehensive database“) [MG00], die LaTeX zu HTML Mappings beinhaltet. Abbildung 2.3 zeigt die Arbeitsschritte des Moduls „XML to bst“.

Für die Transformation wurde XSLT verwendet. Mapping-Einträge aus der *ent.xml* Datenbank werden in Sequenzen übertragen:

```
'\textparagraph' '&para;' find.replace
'\textpilcrow' '&para;' find.replace
'\checkmark' '&#x2713;' find.replace
...
```

Listing 2.7: Beispiel für einen Eintrag aus ent.xml

```
<char pos="127">
  <entity name="para" set="iso-8879-num">
    <desc>=pilcrow (paragraph sign)</desc>
  </entity>
  <entity name="para" set="html4-lat1">
    <desc>pilcrow sign = paragraph sign</desc>
  </entity>
  <unicode value="00B6">
    <desc>PILCROW SIGN</desc>
  </unicode>
  <latex>
    <seq>\P</seq>
    <seq req="textcomp">\textparagraph</seq>
    <seq req="textcomp">\textpilcrow</seq>
  </latex>
  <plain value="B6" set="iso-8859-1" glyph="¶"/>
</char>
```

Aus mehreren Gründen ist die *trans.bst* Datei nicht direkt benutzbar. Einerseits müssen alle 33 Zeilen in Funktionen unterteilt werden, andererseits müssen Übersetzungen aus der Datei *ent.xml* (Beispiel in Listing 2.7 nachträglich korrigiert werden. Diese Korrekturen müssen per Hand durchgeführt werden. Anschließend kann das Ergebnis ins Biblet *bst*-Style file übertragen werden.

[11] Die Extensible Markup Language (engl. für „erweiterbare Auszeichnungs-Sprache“), abgekürzt XML, ist ein Standard zur Modellierung von semi-strukturierten Daten in Form einer Baumstruktur. Er wird vom World Wide Web Consortium (W3C) definiert. XML definiert also Regeln für den Aufbau von Dokumenten, die Daten enthalten, die zum Teil einer fest vorgegebenen Struktur entsprechen, teilweise aber auch Elemente beinhalten, die nicht diesem statischen Schema entsprechen.

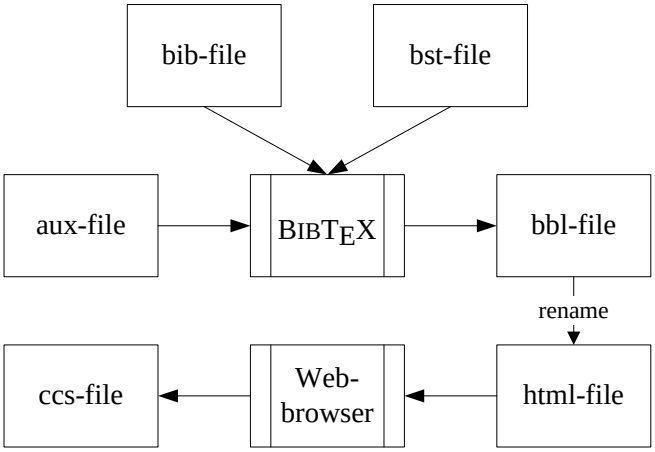

Abbildung 2.2: Biblet Workflow

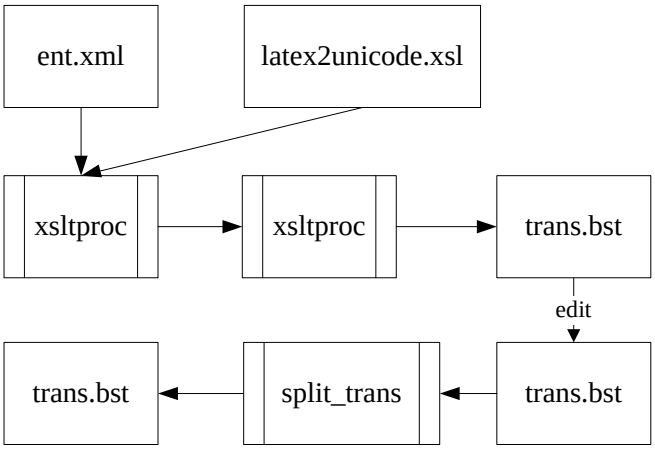

Abbildung 2.3: Transformation XML - bst

Aus dieser umständlichen Art, Textersetzungen durchzuführen ergibt sich ein massives Performance-Problem.

2.2.5 babelbib

Standard-Bibliographiestile enthalten eine Reihe von hartcodierten Textelementen, die für nicht englische Dokumente ungeeignet sind. Solche Elemente sind Monatsbezeichnungen oder die Wörter 'PhD thesis', 'pages', 'edition' u.a.
Im Fall von nicht Englisch sprachlichen Dokumenten muss eine spezielle *bst*-Datei eingesetzt werden, was jedoch im Fall von mehrsprachigen Dokumenten nicht funktioniert, da nur ein Stil definiert werden kann. Eine mögliche Lösung des Problems ist das Paket babelbib von Harald Haders[Har03]. Dieses Paket setzt auf babel auf und erlaubt neben einer flexiblen Sprachbehandlung auch die Anpassung verschiedener typographischer Feinheiten [Lin07]. Dafür ist in der *bib*-Datei ein zusätzliches Attribut language notwendig.

```
lstinline[language=TeX]{language = {german}}
```

Notwendig wird auch die Verwendung eines zu babelbib kompatiblen bibliographischen Stils. Babebib stellt unter anderem die Styles baabbrv, babalpha, babplain und babunsrt äquivalent zu den Standard-Stilen abbrv.bst, alpha.bst, plain.bst und unsrt.bst zur Verfügung. Eine weitere Voraussetzung ist, dass die jeweilig vorkommende Sprache mit babel geladen wird, wie im folgenden Beispiel.

```
{\usepackage[english, german,frenchb]{babel}}
```

Die in der *bib*-Datei vorkommende Sprache muss mit dieser Angabe übereinstimmen. Wenn beispielweise das Paket ngerman verwendet wird[12], muss dies auch in der *bib*-Datei angegeben sein. Dabei ist zu achten, dass auf fremdsprachige Inhalte verzichtet wird.
So kann bspw. der folgende Eintrag nicht von babelbib übersetzt werden:

```
{month = {März}}\\
{edition = {Zweite}}
```

Das Literaturverzeichnis kann jederzeit auf einsprachig umgestellt werden:

[12] entspricht der deutschen Rechtschreibreform

```
{\usepackage[fixlanguage]{babelbib}}\\
{\selectbiblanguage{english}}
```

2.2.6 BibTeXML

BibTeXML ist ein Projekt, das sich mit der Definition einer XML-Syntax für BibTeX beschäftigt. Daneben umfasst BibTeXML eine Reihe von Werkzeugen, die das Arbeiten mit diesen Dateien unterstützt. Das Datenmodell ist eng an BibTeX angelehnt. Für den XML-Benutzer ist es jedoch möglich, Bibliographien mit benutzerdefinierten Daten zu erweitern. Die Import- und Exportmechanismen sind über Stylesheets (XSLT) frei definier- und erweiterbar.
Das Ziel von BibTeXML ist es, eine XML-Umgebung zu schaffen, um BibTeX Daten anzuzeigen und zu strukturieren. BibTeXML soll das Arbeiten mit Bibliographien erleichtern und in Online-Datenbanken das Herunterladen ermöglichen.
Die Grundidee ist, die gesamte Bibliographie mit XML zu verwalten [BPLW01]. Die *bib*-Dateien werden nicht mehr direkt bearbeitet, sondern für die Verwendung von BibTeX generiert. Der Gedanke dahinter ist, eine effiziente Möglichkeit für die Verwaltung strukturierter Daten zu nutzen. XSLT soll die Konvertierung der XML Daten in unterschiedliche Formate unterstützen.

Die Implementierung

» BibTeX *does not have a formally defined grammar, which makes the implementation of a parser difficult.* « [Bee93]

Der BibTeXML Parser baut auf das „bibpars tool“[13], entwickelt von [Bee93] auf.

BibTeX-Einträge können spezielle Zeichenkommandos beinhalten, die in XML nicht sehr aussagekräftig sind. Um dieses Problem zu lösen wurde ein Konverter in Perl geschrieben, der Unicode Zeichen in spezielle XML Elemente übersetzt. Diese Lösung schafft eine volle Rückwärtskompatibilität zu LaTeX.

[13] ein lex/yacc generierter lexical analyzer

Das BibTeXML-Format:
BibTeXML unterstützt zwei unterschiedliche XML Schemata. Eines für die BibTeX-Standard Felder wieauthor, editor, year usw..). BibTeX Standard-Felder werden im Wesentlichen durch die Standard-Style-Files (*bst*) definiert. BibTeX ignoriert unbekannte Felder und ist somit offen für spätere Erweiterungen. Diese Nicht-Standard-Felder werden in einem zweiten Schema definiert. Dies gewährleistet eine möglichst hohe Kompatibilität zum Original.

Beispiel einer BibTeXML-Übersetzung

BibTeX Eintrag vor der Übersetzung:

Listing 2.8: Original-*bib*-Eintrag

```
@Book{lamport:86,
  author =      "Leslie Lamport",
  title  =      "{\LaTeX}:A Document
                Preperation System",
  publisher =   "Addison-Wesley",
  year =        "1986"
  }
```

Nach der Übersetzung:

Listing 2.9: BibTeX-BibTeXML-Übersetzung

```
<bibliography>
  <bibitem type = "book">
  <lable>lampoort:86</label>
    <author>
      <firstname>Leslie</firstname>
      <lastname>Lamport</lastname>
    </author>
    <title><tex code="\LaTeX{}">LaTeX</tex>
     A Document Preperation System>
    </title>
    <publisher>Addison-Wesley"</publisher>
    <year>1986</year>
  </bibitem>
</bibliography>
```

BibTeXML-Dateien (XML), können nicht direkt von BibTeX verarbeitet werden. Daher müssen sie vor der Verwendung in eine gewöhnliche *bib*-Datei umgewandelt werden. Das wird mit einem Stylesheet „BibTeXML2BibTeX“ bewerkstelligt.

2.2.7 CL-BibTeX

CL-BibTeX ist in Common Lisp geschrieben und ersetzt das BibTeX-Programm komplett. Ziel ist es, dem Benutzer die Möglichkeit zu geben, die Formatierung der bibliographischen Angaben in Common Lisp Programmen zu schreiben.
CL-BibTeX liegt in einer stabilen Version vor, die zu BibTeX kompatibel ist. Das Programm teilt sich in mehrere Komponenten auf. Es gibt ein Modul, das BibTeX-Style-Files (.bst) lesen kann, ein Modul das bibliographische Datenfiles (.bib) verarbeitet und ein Modul, das LaTeX (.aux)-Dateien lesen kann. Eingebaute BibTeX Funktionen sind direkt in Lisp implementiert. Mit CL-BibTeX ist es möglich, BibTeX-Style-Dateien weiter zu verwenden, oder Styles in Common Lisp zu schreiben. Zusätzlich gibt es einen Interpreter für BibTeX Style Files. Ein Übersetzer transformiert BibTeX-Style-Files direkt nach Lisp. Diese übersetzten Formatierungsprogramme können vom Benutzer beliebig erweitert werden.

2.2.8 BibTeX++

BibTeX++ [KD03] ist eine in Java geschriebene Neuentwicklung von BibTeX. Java wurde gewählt, um die Portabilität zu gewährleisten und um einen modernen objektorientierten Ansatz zu verfolgen. Um die Kompatibilität zu BibTeX zu gewährleisten, wurde ein Compiler geschrieben, der BibTeX Style-Files nach Java übersetzt. Die Architektur ist erweiterbar ausgelegt, um mit anderen bibliographischen Stilen und Quellenangaben kompatibel zu sein.

Die Besonderheit von BibTeX++ ist die Möglichkeit, „alte“ *bst*-Style-Files in Java-Code zu übersetzen. BibTeX-Styles werden direkt in Java-Klassen abgebildet. Die Annahme ist, dass Erweiterungen und Anpassungen an bibliographischen Styles einfacher in Java realisiert werden können.

»Style programming in BibTeX++*: If we have to define a new language for* BibTeX *++ it should be a clearer language than bst«* [KD03]

»For most users and style designers, since the second block of code has been optimized for human comprehension, it should be easier to modify an existing style as a development basis of a new native BibTeX++ *style. «* [KD03]

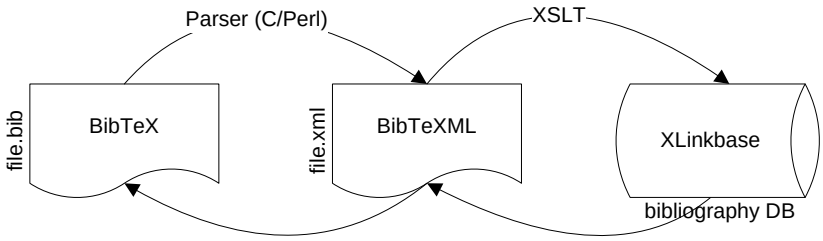

Abbildung 2.4: BibTeXML Architektur [BPLW01]

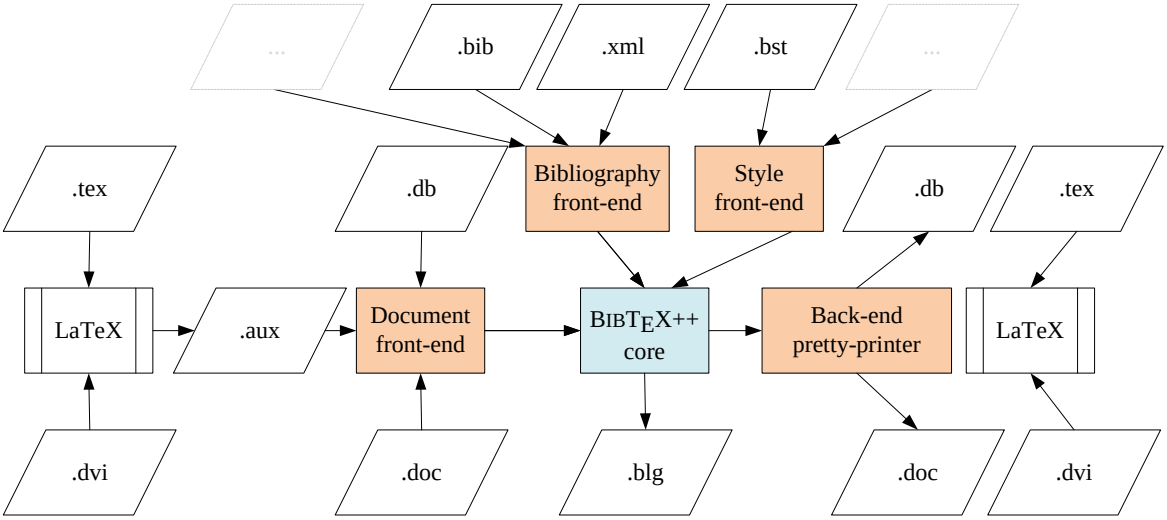

Abbildung 2.5: BibTeX++ Workflow [KD03]

Der BibTeX++ Anwender benötigt keine Kenntnisse in Java oder in objektorientierter Programmierung. Style Designer müssen auf jeden Fall Java Programmierkenntnisse besitzen.
Ein Teil des Konzeptes von BibTeX++ ist es, alle alten Styles von BibTeX anzubieten (Kompatibilität) und BibTeX erweiterbar zu machen (Modularität).
Klassische Themen sind Verwendung von unterschiedlichen Encodings und Multilingualität. Andere Themen sind vernetzte Zugriffe, auch über das Internet. Durch Plug-In-Konzepte kann eine generellere Art entwickelt werden, „alte“ Styles modular zu erweitern.

Architektur von BibTeX++

Die Architektur folgt einem generischen, skalierbaren Ansatz. Im Prinzip ist die Architektur eng an jene von BibTeX angelehnt. Als Input dient das *aux*- und das *bib*-File. Der Output wird in das *bbl*-File geschrieben. Der grundsätzliche Workflow ist durch das Front- und Backend bestimmt, die für bestimmte Ein- und Ausgaben-Muster bereitstehen.

Compiler *bst* zu Java

Die *bst*-Sprache ist Stack-basierend und verwendet die Postfix-Notation. Daten werden über einen Stack ausgetauscht und manipuliert. Auch Funktionen übergeben Daten über diesen Mechanismus. Es gibt einige Stack-basierende Sprachen (z.B. Forth, PostScript, RPL), und auch die zugehörigen Compiler, die diese Sprachen in C oder Java übersetzen. Für einen Programmierer stellen vor allem die Postfix-Notation, implizite In- und Output-Variablen und die umgekehrte Schreibweise der Kontrollstrukturen eine Herausforderung dar. Für den Bau eines Compilers ist die Übergabe von Daten innerhalb von Funktionen und zwischen Funktionen das Hauptproblem.
Es gibt mehrere Ansätze und Vorschläge, eine Stack-basierende Sprache in eine Standardprogrammiersprache zu übersetzen. Ein möglicher Ansatz ist der „Source to source Translator“. Der bekannteste dieser Art ist „f2c Forth to C“ [Ert92], der die Programmiersprache Forth nach C übersetzt. Eine andere Art von Compiler, die Stack-basierende Sprachen übersetzen, sind Decompiler, die byte-Code zum Beispiel nach Java rückübersetzen. Eine ausführliche Darstellung der Möglichkeiten *bst* nach Java zu übersetzen ist in [KD03] angeführt.

2.3 Zusammenfassung BibTeX-Neuentwicklungen

BibTeX ist der gebräuchlichste bibliographische Prozessor im Zusammenhang mit LaTeX. Ein Hauptargument für die Verwendung von BibTeXist die Code-Stabilität.
»One more point about BibTEX is code stability. The current version of BibTEX 0.99c has not been modified since 1988 . So, both BibTEX the program and the file format are quite stable. This is why a large number of free-software tools for processing files in BibTEX format could be developed« [Fen07]
Durch diese Stabilität des Dateiformates (*bst*-File) wurde eine Vielzahl an nützlichen Tools entwickelt.
Die Tatsache, dass BibTeX-*bst* eine Programmiersprache zum Definieren von bibliographischen Styles ist, ermöglicht einen großen Gestaltungsraum. Der größte Nachteil der BibTeX-Programmiersprache bst ist die fehlende Modularität, dadurch sind flexible Lösungen nur bedingt möglich. Das Zusammenspiel der umständlichen Programmierung mit dem eingeschränkten Funktionsumfang von BibTeX macht die Weiterentwicklung, zum Beispiel von multilingualen Features, sehr aufwendig. Ein weiteres Problem liegt direkt im Sprachkern von BibTeX, alphanumerische Zitierschlüssel werden bei Verwendung von Multibyte-Encoding nicht korrekt sortiert.
Es gibt einige Ansätze diese Defizite zu beseitigen. Dabei gilt es, die Modularität und Erweiterbarkeit unter Bewahrung der Fehlerfreiheit, Robustheit, Effizienz und der einfachen Bedienbarkeit von BibTeX zu verbessern.
Einige Ansätze bauen direkt auf den Sprachkern von BibTeX auf, um multilinguale Features zu ermöglichen.
Babelbib ist ein Zusatzpaket für LaTeX und stellt ein Set für eigene bst-Dateien zur Verfügung, die um das Eintragsfeld language erweitert wurden. Der Vorteil ist, dass BibTeX wie gewohnt verwendet werden kann. Der Nachteile ist, dass *bib*-Dateien um das Feld language erweitert werden müssen.
»The AMS BibTEX styles are different from the standard styles in one aspect: They print the language of the citation for some document types. Thus, they already have the BibTEX field language.« [Har02]
Das babelbib-Paket umfasst nur einen eingeschränkten Umfang an Style-Dateien. Das heißt, die große Anzahl an bestehenden, in BibTeX-*bst* geschriebenen Styles, kann nicht wiederverwendet werden. Ein weiterer Nachteil von babelbib ist, dass sich Sprachmerkmale immer auf die Referenz beziehen, nicht aber auf das Dokument.
MlBibTeX dagegen verfolgt beide Ansätze, den referenz- und den dokumentenabhängigen Ansatz.
»This new tools meets the new requirements

- *Of users who wish the language used for the bibliographical references of a printed work to be work's language, this approach being so called document-dependent*

- *As well as users who prefere the language of each reference to be the reference's, that is, the information related to work written in English is displayed in English, this second approach being so-called reference-dependet.« [Huf02b]*

Im Gegensatz zu babelbib ist MlBibTeX eine komplette Neuentwicklung von BibTeX. MlBibTeX benötigt eine aktuelle Version von LaTeX2E, inklusive einer Version des multilingualen babel-Paketes. MlBibTeX ersetzt *bst* durch durch die Style-Definition nbst, eine XSLT ähnliche Sprache. Nbst-Files ersetzen bestehende *bst*-Files. Im Gegensatz zu *bst*, erlaubt *nbst* rekursive Templates für die iterative Programmierung.
Die Wiederverwendbarkeit alter *bst*-Files ist nur bedingt möglich. Einzelne *bst*-Funktionen wurden von *bst* nach *nbst* übersetzt. Komplexe Funktionen und interne BibTeX-Funktionen wurden in Scheme implementiert. Somit besteht dasselbe Problem wie bei babelbib, die Vielzahl an *bst*-Files kann nicht direkt wiederverwendet werden. Für MlBibTeX spricht die Möglichkeit der modulareren Entwicklung von Styles und die erweiterten multilingualen Eigenschaften, für babelbib der etablierte *bst*-Standard.
MLbibtex ist nicht für alle gängigen Plattformen in einer stabilen Version verfügbar. Dem Benutzer steht ausschließlich der Source-Code zu Verfügung. Praktisch gesehen ist die Benutzergruppe im Wesentlichen auf Unix-Benutzer beschränkt.
Das **BibLaTeX**-Paket ist eine komplette Neuimplementierung der bibliographischen Funktionalität von LaTeX.
» *This package provides advanced bibliographic facilities for use with LaTeX in conjunction with BibTeX. The package is a complete reimplementation of the bibliographic facilities provided by LaTeX.* « [Leh09]
BibLaTeX wird ausschließlich zum Sortieren der Literatureinträge und zum Erzeugen der Labels verwendet. Anstelle von *bst*-Dateien wird die Formatierung der bibliographischen Einträge mit LaTeX-Macros gesteuert. Dadurch ist es möglich, ohne Kenntnisse der BibTeX postfix-Stack-Sprache, Zitierstile anzupassen. BibLaTeX wird mit eigenen Zitierstilen ausgeliefert. Diese sind speziell für BibLaTeX implementiert. BibTeX-bst-Dateien können nicht verwendet oder konvertiert werden. Zitierstile können abgeändert oder neu erstellt werden.
BibTeX++ ist eine in Java geschriebene Neuentwicklung von BibTeX.
» BibTeX++ *is an extendable tool dealing with the bibliographical area of electronic documents. It aims at extending the well known* BibTeX*in the LaTeX world by adding modern features such as Unicode document encoding, Internet capabilities, scalability to future usages, and future tools through plugin mechanisms and at the same time to remain compatible with plain old* BibTeX*.*
BibTeX++ *is a free software program written in Java, a clean portable object oriented language that natively handles Unicode. Since it is written in Java,* BibTeX++ *is ready to run on every Java-enabled computer, although the installation phase is still to be streamlined.* «[KD03]
BibTeX++ ermöglicht es, einen *bst*-Style in Java-Klassen zu übersetzen. BibTeX++ ver-

wendet erweiterte Übersetzertechniken des Compilers (BISTRO), um *bst*- und *bib*-Files in in Java geschriebene Styles zu übersetzen. BibTeX++ -Styles können als erweiterbare Java-Klassen plattformunabhängig eingesetzt werden. Sie sind unter Linux und MS-Windows getestet und in einigen Linux-Distributionen und MiKTeX enthalten.
Ähnlich wie BibTeX++, ist Bibulus eine Neuentwicklung von BibTeX. Anstelle von Java wurde Perl als Programmiersprache verwendet. Genauso wie Java, kann Perl mit Unicode -Encoding umgehen. Im Unterschied zu BibTeX++ liegt kein Übersetzer vor, der *bst*-Styles nach Perl übersetzt.
Die Gemeinsamkeit von **Bibulus** und BibTeX++ liegt darin, modulare plugin-Techniken zu schaffen und erweiterbaren Code zu erzeugen. Der Anwender benötigt keine Kenntnisse in objektorientierter Programmierung. Sehr wohl sind aber Java- beziehungsweise Perl-Kenntnisse für die Entwicklung von Styles notwendig. Im Unterschied zu BibTeX++ ist Bibulus multilingual ausgerichtet, unterstützt jedoch weder direkt noch indirekt BibTeX-*bst*-Files .
Cl-bibtex ist in Common Lisp geschrieben und liegt in einer stabilen Form vor.
»Cl-bibtex is based on ANSI Common Lisp. It includes an interpreter for the bst language, and can also compile a BibTEX style file into a Common Lisp program, as a starting point for customising such a style, by refining the corresponding Common Lisp program.« [Huf08]
Cl-bibtex ermöglicht es, sowohl BibTeX-Style-Files als auch Styles, die in Common Lisp geschrieben sind, zu verwenden. Darüber hinaus steht ein Programm zur Verfügung, das BibTeX-*bst*-Dateien nach Cl-bibtex transformiert. Der LaTeX–BibTeX-Workflow wurde im Wesentlichen beibehalten.
Ziel von **BibTeXML** ist es, eine XML-Umgebung zu schaffen, die BibTeX-Daten anzeigt und strukturiert.
»BibTeXML is an XML representation of BibTeX data. It can be used to represent bibliographic data in XML. The atvantage of BibTeXML over BibTeX's nativ syntax is that it can be easily managed using standard XML tools (in particular, XSLT style sheets), while native BibTeX data can only be manipulated using specialized tools.« [BPLW01]
BibTeXML soll das Arbeiten mit Bibliographien erleichtern und die direkte Anbindung an Online-Datenbanken ermöglichen. BibTeXML bezieht sich ausschließlich auf bibliographische Inhalte (*bib*-Files) und nicht auf Formatierungen (*bst*-Files).
Die Gemeinsamkeit dieser Neuentwicklung ist, dass bestehende *bst*-Files nicht uneingeschränkt wiederverwendet werden können. BibTeX++ unterstützt die Übersetzung von Styles direkt in Javaklassen. MlBibTeX bietet einen Kompatibilitätsmodus an, in dem alte *bst*-Styles verwendet werden können. Neuentwicklungen wie MlBibTeX, BibTeX++ und Bibulus setzen auf strukturiertere bibliographische Datenhaltung, zumeist in XML.
Alle Programme haben gemeinsam, dass ein Style-Programmierer die jeweilige Programmiersprache beherrschen muss. Ob das in jedem Fall die Entwicklung und Wartung von bibliographischen Styles erleichtert, kann nur subjektiv bewertet werden. Auf jeden

Fall besteht der Anspruch, eine modernere, modularere, objektorientierte Umgebung zu schaffen.
Bibulus und MlBibTeX sind stark multilingual ausgerichtet. BibTeX++ und Cl-bibtex unterstützen die Übersetzung von Styles in die jeweilige Zielsprache.
Bibulus und BibTeX++ unterstützen nativ Unicode-Encoding.
MlBibTeXhat durch die Verwendung einer XSLT-ähnlichen Sprache am ehesten eine Applikation geschaffen, die es einem großen Anwenderbereich ermöglicht, Styles zu entwickeln und zu erweitern oder entsprechende Frameworks dafür zu schaffen.
Babelbib ist die einfachste Möglichkeit, multilinguale Features zu nutzen und in der vertrauten BibTeX-Umgebung zu bleiben, setzt aber auf die alte *bst*-Technologie auf.

2.4 Standard-Schnittstellen

Die meisten Bibliographie-Programme können Datenbanken anderer Programme nicht direkt einlesen. Um Datenbestände von einem Programm in ein anderes zu übertragen, muss im Regelfall ein Standard-Austauschformat zwischengeschaltet werden.

2.4.1 Standard- und Austauschformate

Der bekannteste offene Standard für bibliographische Metadaten ist BibTeX. Ein weiterer verbreiteter Standard ist RIS. Der Standard „Refer“ unterstützt nur drei Publikationstypen. Programme, die u.a. BibTeX und/oder RIS unterstützen, sind:

- Bibliographix (liest und schreibt BibTeX und RIS)
- BibTeX (ist der Standard selbst)
- Citavi (liest und schreibt BibTeX, liest RIS)
- Endnote (liest und schreibt RIS)
- Synapsen (liest und schreibt BibTeX)

Copac
Copac ist ein Zusammenschluss mehrerer Kataloge bibliographischer Informationen britischer und irischer Hochschulen mit freiem Zugang. Es gibt vier Schnittstellen zu Copac. Die Web-Schnittstelle (Web/Z39.50 Gateway) ist die am meisten verwendete. Es kann jedoch auch der Zugriff über die Copac Z39.50-Schnittstelle, die OpenURL-Schnittstelle oder die SRW/SRU-Schnittstelle erfolgen.

Machine-Readable Cataloging (MARC)
MARC definiert ein bibliographisches Datenformat, welches hauptsächlich für den Datenaustausch konzipiert wurde. MARC liegt in mehreren nationalen und internationalen Varianten vor.

MODS XML
Metadata Object Description Schema (MODS), ist ein XML-basierter Standard zur Beschreibung bibliographischer Daten [Gar03].
MODS wurde als Kompromiss zwischen den komplexen MARC Formaten und dem einfachen „Dublin Core Metadaten“ Format entworfen[14].

RIS
RIS ist ein standardisiertes Tag-Format, das von „Research Information Systems Incorporated“ entwickelt wurde. Es ermöglicht Zitatverwaltungsprogrammen den Datenaustausch bibliographischer Informationen. Es beschreibt Tags (zwei Zeichen), die die Struktur einer Bibliographie beschreiben[15].

2.4.2 Zitierstil Sprachen

Neben den produktspezifischen Zitierstilen wie Endnote Styles, oder Bibtex bst, wird mit CLS versucht, einen offenen Standard für den Austausch von Zitierstilen zu schaffen.

CSL (Citation Style Language)
CSL ist ein offener XML-basierender Standard, um Zitat-Formatierungen zu konfigurieren. Damit soll eine gewisse Unabhängigkeit von einzelnen Applikationen geschaffen werden. CSL ist eine einfache Möglichkeit, unter Verwendung von XML Zitierstile zu beschreiben. Zum Entwickeln neuer Zitierstile wird ein einfacher Texteditor verwendet. Derzeit liegen über 1000 verschiedene Stile vor. CSL eignet sich auch für den Austausch von Zitierstilen.
CSL basiert auf der Klasse Zitat, die alle Validierungen um sie herum strukturiert. Wenn ein neuer Stil erzeugt wird, z.B. „author-year“, dann ist eine Schemadefinition für Autor und Jahr notwendig. Darüber hinaus sind Referenz-Typen (book, article, journal usw.) in der Referenzklasse gruppiert. Die Klassen-basierte Struktur wird verwendet, um ein stabiles modulares System zu schaffen. Die Struktur ist hierarchisch aufgebaut, mit Rendering-Informationen in den entsprechenden Metadaten. Das Ergebnis der Zeichensetzung wird nur dann angewandt, wenn Metadaten im Datensatz vorhanden sind.

[14] http://www.loc.gov/standards/mods/
[15] http://www.refman.com/support/risformat_intro.asp

Das Markup ist so konzipiert, dass es leicht in eine GUI-basierte Anwendung, wie z.B. einem Stil-Editor, integriert werden kann.

2.4.3 Online Literaturdatenbanken

CiteSeer CiteSeer ist eine Suchmaschine für frei zugängliche wissenschaftliche Literatur. Sie wird für Literaturrecherche und Zitationsanalyse verwendet. Schwerpunkt ist das Fachgebiet Informatik.

IEEE Xplore
IEEE Xplore ist eine Volltextdatenbank, vorwiegend im Bereich technischer Literatur; Fachgebiete: Elektrotechnik, Informatik und Elektronik.

PubMed
PubMed ist eine Meta-Datenbank für biomedizinische Zwecke. Sie dient hauptsächlich zur Recherche von medizinischen Artikeln. Es handelt sich um eine bibliographische Referenzdatenbank.

Web of Science
Web of Science (ISI Web of Knowledge) ist ein kommerzielles Angebot mit mehreren Online-Zitationsdatenbanken, erstellt vom Institute for Scientific Information (ISI). Ein Vergleich mit anderen sozialwissenschaftlichen Datenbanken wird in [NO07] angestellt.

Z39.50
Z39.50 wurde entwickelt, um die Kommunikation zwischen Computersystemen, welche Literaturdaten verwalten, zu unterstützen [Mil]. Die Kommunikation kann zwischen zwei Personal Computern, oder einem öffentlichen OPAC Terminal erfolgen[16]. Z39.50 ist ein Netzwerkprotokoll, das im Bibliothekswesen als Standard zur Abfrage von bibliographischen Informationssystemen verwendet wird. Das Protokoll befindet sich auf der Anwendungsschicht 7 des OSI-Schichtenmodells [AP97]. Verwaltung und Entwicklung des Protokolls erfolgt über die Library of Congress.

Entrez
Entrez ist eine Suchmaschine, die vom National Center of Biotechnology Information (NCBI) für biomedizinische Zwecke betrieben wird. Entrez ermöglicht den gleichzeitigen

[16]OPAC (Online Public Access Catalogue) ist eine gebräuchliche Katalogform des Bibliothekswesens

Zugriff auf mehrere Datenbanken und vernetztes Suchen.

Medline
Medline (Medical Literature Analysis and Retrieval System Online) ist eine bibliographische Datenbank, die biomedizinische Informationen anbietet. Sie enthält bibliographische Informationen und Artikel aus Fachzeitschriften für Medizin, Pflege, Apotheke, Zahnmedizin, Veterinärmedizin und Gesundheitswesen.

2.5 Software für Literaturverwaltung

In diesem Kapitel werden unterschiedliche Literaturverwaltungssysteme gegenübergestellt. Dabei soll der Leistungsumfang hinsichtlich Kriterien einer BibTeX Neuimplementierung verglichen werden. Besondere Beachtung soll den Themen Mehrsprachigkeit, Unicode, Schnittstellen zu anderen Systemen und Einbindungsmöglichkeiten in LaTeX geschenkt werden. Dabei ist die Verwendbarkeit hinsichtlich bibliographischer Stile ein hervorzuhebendes Kriterium.

2.5.1 Systeme zur Verwaltung von Bibliographien

Aigaion

Aigaion[17] ist ein Mehrbenutzersystem zum Verwalten von Bibliographien, mit BibTeX- und RIS-Import- und Exportfunktionen. Es transformiert formatierte Bibliographien ins HTML- und RTF-Format. Aigaion ist eine Web-basierte Anwendung zur Verwaltung von Bibliographien. Unterstützte Zitierstile sind u.a. APA, Chicago/Turabian, MLA, Harvard, BMJ und IEEE. Zitierstile werden mit dem OSBib-Format definiert. Aigaion ist in PHP und MySql entwickelt und ist plattformunabhängig.

BibDesk

BibDesk ist ein Programm mit graphischer Benutzeroberfläche und läuft ausschließlich auf Mac OS X. Es unterstützt die Verwaltung von BibTeX Dateien. BibDesk unterstützt Zitierstile über BibDesk Exportvorlagen.
Exportformate: BibTeX Medline, MODS XML, RIS.

[17] http://www.aigaion.nl/

Importformate: BibTeX Copac, Endnote/Refer/BibIX, ISI, Medline, MODS XML, PubMed, RIS, SciFinder, MARC, JSTOR und Reference Miner.
Referenzlisten können durch die Formate HTML, LaTeX RTF, Plain text, RSS, Atom, DOC, PDF und XML dargestellt werden. BibDesk erlaubt den direkten Zugriff auf online Datenbanken ArXiv, CiteSeer, IEEE Xplore, PubMed Web of Science und viele Bibliotheken, die den Z39.50-Standard unterstützen.

Biblio (Drupal module)

Das Drupal Scholar Modul erlaubt es Benutzern, Listen wissenschaftlicher Publikationen zu verwalten und anzuzeigen. Importformate sind BibTeX, RIS, MARC, EndNote und XML.
Exportformate sind BibTeX, EndNote und XML.
Das Drupal Scholar Modul stellt AMA, APA, Chicago, CSE, IEEE, MLA und Vancouver als Output Styles zur Verfügung.

Biblioscape

Biblioscape ist eine kommerzielle Software und läuft ausschließlich unter MS-Windows. BibTeX, Medline, RIS werden als Exportformat unterstützt. Importformate sind BibTeX, Copac, Endnote/Refer/BibIX, ISI, Medline, MODS XML, PubMed, RIS und SciFinder. Biblioscape unterstützt die Zitierstile APA, Chicago/Turabian, MLA, Harvard und weitere (mehr als 2000) mitgelieferte Stile. Zitierstile werden in der graphischen Benutzeroberfläche (GUI style Editor) verwaltet. Als Referenzlisten werden HTML, Plain Text und RTF unterstützt. Unterstützte Textverarbeitungsprogramme sind MS-Word und WordPerfect.

BibSynonym

BibSynonym ist ein System zum Austausch von Bookmarks und Literaturlisten. Es dient dazu Lesezeichen zu organisieren und zu veröffentlichen. Bibliographien sind im BibTeX-Format gespeichert, sie können aber auch ins EndNote-Format oder nach HTML exportiert werden.

Bibus

Bibus verwaltet Literaturquellen, organisiert Zitate und hilft beim Wissens- und Aufgabenmanagement. Internetrecherche für Literatur ist in fast 300 Katalogen inklusive Web-Suche über die ISB-Nummer möglich. Die Anzahl der speicherbaren Titel ist auf 100 pro Projekt beschränkt

EndNote

EndNote ist eine kommerzielle Literaturverwaltung für wissenschaftliches Arbeiten, erstellt automatisiert Literaturverzeichnisse und bietet Recherchemöglichkeiten in Online-Datenbanken. EndNote ist ein Programm zum Speichern und Verwalten von Literaturreferenzen und Abbildungen in einer privaten Datenbank. EndNote läuft unter MS-Windows und MacOSX und kann mit Microsoft Word verknüpft werden. Ähnlich wie BibTeX, wird das Erstellen von wissenschaftlichen Arbeiten unterstützt, indem Zitierschlüssel verwaltet und Literaturverzeichnisse erstellt werden. Im Unterschied zu BibTeX handelt es sich um ein Programm mit graphischer Benutzerumgebung. Zitier-Stile können nur menügeführt definiert und angepasst werden. Sie werden in einem eigenen EndNote-Format gespeichert. Standard Formate wie APA, Chicago/Turabian, MLA und Harvard werden unterstützt. Die Verwaltung der Stile erfolgt über den ebenfalls menügeführten „Style Manager“.
Exportformate: BibTeX, Medline, RIS.
Importformate: Endnote/Refer/BibIX, ISI, Medline, PubMed, RIS, SciFinder, MARC, JSTOR, Reference Miner.
Referenzlisten unterstützen die Formate HTML, RTF, Plain Text und XML.
EndNote erlaubt den direkten Zugriff auf Online-Datenbanken, u.a. PubMed, CINAHL, DIMDI und Bibliotheken, die den Z39.50 Standard unterstützen. EndNote verfügt über mehr als 3000 bibliographische Zitier-Stile, die jeweils in einer Datei gespeichert sind[18].

RefBase

Das RefDB Literatur- und Bibliographieverwaltungsprogramm erzeugt formatierte Bibliographien für Markupsprachen wie DocBook (XML+SGML), TEI XML oder LaTeX. RefDB ist von der Kommandozeile aus bedienbar, direkt aus Texteditoren (Emacs, vim), über eine PHP-basierte Web-Oberfläche oder über eine SRU-Schnittstelle. Es importiert übliche Formate wie BibTeX und exportiert nach MODS, ODS, Endnote, BibTeX und in das RIS-Format.

[18] Angabe des Herstellers EndNote x2.0.1 Help

zNote (Python)

ZNote ist ein Web-basierendes Bibliography-Management Tool. Es verwendet ein hierarchisches XML-Dateiformat.

Zotero

Zotero ist ein Plugin für den Browser Firefox, das nur zusammen mit Firefox Version 2 oder höher funktioniert. Aus bestimmten Webseiten lassen sich bibliographische Daten verwalten und u.a. nach BibTeX exportieren. Zotero bietet Funktionalität für das Sammeln, Organisieren und Durchsuchen von wissenschaftlichen Arbeiten. Zusätzlich gibt es grundlegende Import/Export-Funktionen und Werkzeuge zum Formatieren von Bibliographien. Über (automatische) Updates der Software werden laufend weitere Zitationsstile geliefert, die eine automatische Erkennung neuer Online-Ressourcen ermöglichen. Alle mitgelierten Zitationsstile sind in CSL (Citation Style Language) geschrieben.
Zotero unterstützt alle gängigen Betriebssysteme, sowie die Textverarbeitungsprogramme MS-Word, Open Office und LyX. Eine Vielzahl von In- und Exportformaten von Bibliographien wird unterstützt: BibTeX, Copac, Endnote/Refer/BibIX, ISI, Medline, MODS XML, PubMed, RIS. Als Referenzlisten können HTML- und RTF-Dateien, jedoch keine LaTeX-Dateien eingesetzt werden.
Der Hauptvorteil von Zotero ist, dass mit vielen gängigen Datenbank Schnittstellen (wie z.B. ArXiv, CiteSeer, IEEE Xplore, PubMed) als auch mit wissenschaftlichen Webseiten und digitalen Bibliotheken wie PubMed, Google Scholar, Google Books, Amazon.com oder Wikipedia gearbeitet werden kann.

2.5.2 Systeme mit Zugriff auf bibliographische Datenbanken

2collab

2collab[19] ist eine Web-basierende Applikation, die die Kommunikation und Zusammenarbeit bei der Suche wissenschaftlicher Texte und den Austausch von Referenzen, Lesezeichen und sonstiger vernetzter Information unterstützt.
Als Exportformate können BibTeX und RIS gewählt werden. Als Importformat dient RIS. Referenzlisten können in HTML als Textdateien oder im RSS Format verarbeitet werden.

[19] http://www.2collab.com/

Basilic

Basilic ist ein Bibliographie-Server für Forschungseinrichtungen, der die Ausbreitung der Veröffentlichungen automatisiert und erleichtert. Dabei werden die über Internet automatisch generierten Webseiten auf einer Datenbank veröffentlicht. Jede Veröffentlichung hat eine verbundene Webseite, die Downloads und weitere Dokumente enthält (Bilder, BibTeX ...). Darunter steht eine Suchmaschine mit mehreren Optionen für die Anzeige von Ergebnissen, die indizierte Seiten verwaltet.

BibConverter

BibConverter[20] ist eine Web-Anwendung, die es erlaubt Daten aus Online-Literaturdatenbanken ins BibTeX-Format zu übersetzen. BibConverter unterstützt die Übersetzung von Daten aus IEEE Xplore, Engineering Village 2 und ISI Web of Science.

Bibwiki (MediaWiki)

Bibwiki[21] ist eine Erweiterung von MediaWiki, welche die Verwaltung von wissenschaftlichen Bibliographien unterstützt. Bibliographien werden im BibTeX-Format gespeichert.

Bibwiki ermöglicht das Importieren von bibliographischen Einträgen direkt von der entsprechenden Web-Site und formatiert die Bibliographie als BibTeX-Eintrag. Für solche Einträge können auch Memos, Artikel und Auszüge verfasst werden. Für erfasste Artikel können Literaturreferenzlisten erstellt und abgespeichert werden. Bibwiki ermöglicht es mittels BibTeX-Style-Files Literaturlisten ins HTML-Format zu exportieren.
Verfügbare Quellen für den Import sind öffentliche Datenbanken und Buchhandlungen:
ArXiv, Amazon, CSA/Proquest Databases
US-amerikanische Bibliotheke: Library of Congress
Britische Bibliotheken: British Library London, Library of the University of London
Deutsche Bibliotheken: Deutsche Nationalbibliothek, FU Berlin, HU Berlin, TU Berlin
Österreichische Bibliotheken: Österreichische Nationalbibliothek, TU Graz, UB Graz, UB Innsbruck, UB Klagenfurt, UB Leoben, JKU Linz, UB Salzburg, Boku Wien, TU Wien, UB Wien, WU Wien.

[20] http://www.bibconverter.net/
[21] http://www.mediawiki.org/wiki/MediaWiki

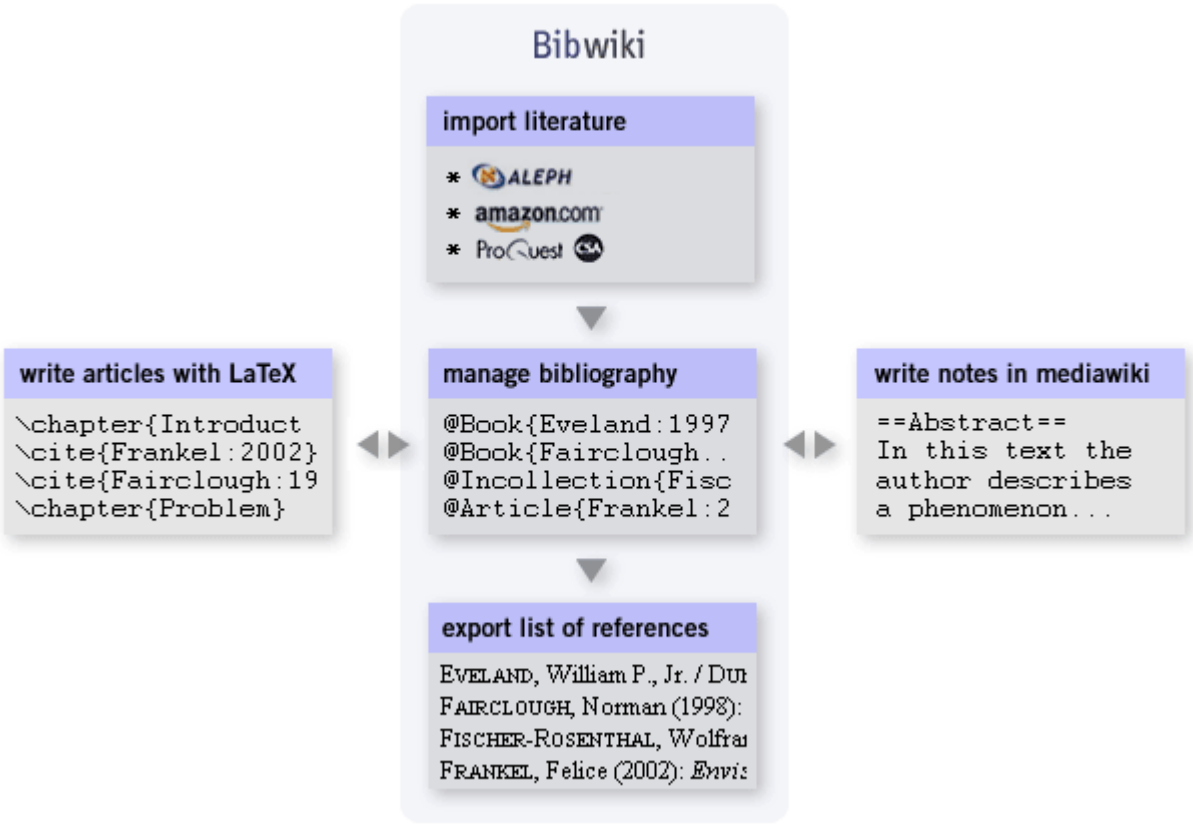

Abbildung 2.6: Bibwiki Workflow (http://www.plaschg.net/bibwiki/Main/KeyFeatures)

Sente

Sente ist ein Literaturverwaltungsprogramm für akademische Zwecke, das u.a. eine automatische Suche in einer Vielzahl von Datenbanken bietet. Suchergebnisse werden täglich aktualisiert und die Funde anhand mehrerer Kriterien automatisch eingefügt

Synapsen

Synapsen basiert auf einer SQL-Datenbank und dient dem Verwalten von Literaturquellen. Anhand von eingegebenen Schlagworten vernetzt das Programm einzelne Quellen automatisch. Synapsen bietet eine direkte Verbindung zu Textverarbeitungsprogrammen und Datenimport aus OPAC sowie GBV oder der Library of Congress via Z39.50-Protokoll.

2.5.3 Werkzeuge für die Verwaltung von BibTeX-Datenbanken

allbib

Allbib[22] ist ein in Perl geschriebenes Programm mit GTK bindings, welches das Bearbeiten von BibTeX Literaturdatenbanken unterstützt (Erzeugen und Verändern von bibliographischen Einträgen). Allbib versteht die BibTeX-File-Syntax und ist ein nützliches Werkzeug beim Suchen und Überprüfen solcher Dateien.

Bib-it

Bib-it ist eine graphische Benutzeroberfläche, die die Verwaltung von BibTeX-Dateien unterstützt und auf allen Plattformen mit Java Runtime 1.5 läuft.
Zur Unterstützung der verschiedenen BibTeX-Styles (bst), die unterschiedliche Arten von Einträgen und Feldtypen verwenden, werden unterschiedlichen Stile in *ini*-Dateien mitgeliefert. Die Standard Stil-Datei ist die Datei *plainStyle.ini*, die auf den Standard-Eintragstypen und Feldtypen der ursprünglichen vier BibTeX-Style-Files (plain.bst, abbrv.bst, alpha.bst und unsrt.bst) beruht, die mit BibTeX mitgeliefert werden.

BibORB

BibORB[23] ist eine web-basierende Lösung welche verteilte Bibliographien verwaltet. Sie ermöglicht das Importieren und Exportieren von Referenzen. BibORB wurde ursprünglich für die Verwaltung von BibTeX-Dateien entworfen. BibORB erzeugt ein BibTeX File aus einer LaTeX Aux Datei.

Bibtool

BibTool [Neu97] ist ein in der Sprache C geschriebes Programm. Es wird auf der Kommandozeile bedient und dient zur Bearbeitung und Umwandlung von Literaturdatenbanken für BibTeX. Es kann Einträge sortieren, standardisierte Feldbezeichner erstellen und auf benutzerspezifizierten regulären Ausdrücken basierende Datenfeldinhalte verändern. Bibtool wird auf der Kommandozeile aufgerufen. Mit dem jeweiligen vorgesehenen Parameter können unterschiedliche Operationen ausgeführt werden:

[22] http://allbib.sourceforge.net/
[23] http://biborb.glymn.net/doku.php

Sortieren und Zusammenfügen

```
bibtool -s file1.bib file2.bib
```

Die Option -s verbindet die beiden angegebenen Dateien (file1.bib file2.bib) und sortiert die Einträge nach Referenzschlüssel.
Ein Bsp. für Sortieren nach bestimmten Spalten z.B. author erfolgt mit dem Kommando:

```
bibtool -- sort.format="\%N(author)" _le1.bib _le2.bib
```

Generierung von Referenzschlüssel

Mit der Option -K werden die BibTeX-Einträge mit neu erstellten Schlüsselwerten in eine neue Datei geschrieben.

```
bibtool -K file.bib -o file1.bib
```

Die Standard Formatierung von bibtool erzeugt einen Referenzschlüssel, bestehend aus dem Namen des Autors und dem ersten relevanten Wort des Titels, getrennt durch einen Doppelpunkt.
Benutzerdefinierte Formate für den Schlüssel sind mit der Option -f möglich, z.B.:

```
bibtool -f n(author):%2d(year)file.bib -o file1.bib
```

Weitere Funktionen von bibtool sind:

- Syntaxprüfung mit vernünftigen Fehlermeldungen
- Semantische Prüfungen
- Analyse von *aux*-Files und Erstellen von zugehörigen BibTeX-Files
- Mittels regulärer Ausdrücke sind komplexe Abfragen möglich
- Statistik über die BibTeX-Files

Interface zu anderen Programmiersprachen

Bibtool kann für andere Programmiersprachen als Schnittstelle zu BibTeX genutzt werden. Dabei wird die Eigenschaft, dass Bibtool BibTeX-Dateien in normalisierter Form ausgibt, genutzt. Momentan sind Schnittstellen zu CGI Sript (via Perl) und Tcl verfügbar.

btOOL

Ist eine Sammlung von Programm Bibliotheken, die Schnittstellen zu BibTeX-Files unterstützen. Sie liegen in den Programmiersprachen C und Perl vor.

CiteProc

CiteProc wird eine Sammlung von mehreren Programmen genannt, die sich mit der Formatierung von Bibliographien und Zitierungen auseinandersetzen. Alle diese Programme haben das XML-basierende Format CSL (Citation Style Language) gemeinsam. CiteProc wurde in mehreren verschiedenen Programmiersprachen weiterentwickelt (z.B. javascript (in Zotero), Haskell, Python, and Ruby).
CiteProc und CSL bilden ein allgemeines Framework zur Formatierung von Bibliographien unter Verwendung von Markup Sprachen. Unterschiede zwischen den verschiedenen Implementierungen bestehen im bibliographischen Datenbank-Format, die meisten unterstützen MODS XML.

Document Archive (docarc)

Docarc[24] ist eine Web-basierte Datenbank für elektronische Dokumente und BibTeX-Einträge. Diese Software kann über Plugins und Konsolen-Frontends erweitert werden[25]. Document Archive wurde als Alternative zu BibTeX bib Dateien entwickelt. Document Archive ist in Perl geschrieben und verwendet als zugrundeliegende Datenbank MySql.

Document Database

Document Database ist eine Web-basierende,in PHP geschriebene Datenbank-Lösung und dient zum Organisieren von wissenschaftlichen Artikeln und deren BibTeX-Einträgen.

Ebib

Ebib[26] ist eine Unterstützung für BibTeX-Datenbanken unter Emacs. Hauptmerkmale sind die visuelle Darstellung und Unterscheidung der obligatorischen und optionalen Felder. Ebib bietet Kopier-, Ausschneide- und Einfügemechanismen für das schnelle Kopieren von Feldwerten und das automatische Laden von bib Dateien beim Start.

[24] http://docarc.sourceforge.net/
[25] Dzt. nur für Mozilla und Firefox
[26] http://ebib.sourceforge.net/

JabRef

Jabref ist eine grafische Benutzeroberfläche zum Verwalten von Referenzen im BibTeX-Format. Jede Referenz kann mit einer PDF-Datei oder einer Webseite verknüpft werden. JabRef hat Schnittstellen zu Medline, CiteSeer, LyX, WinEdit, EndNote und OpenOffice.

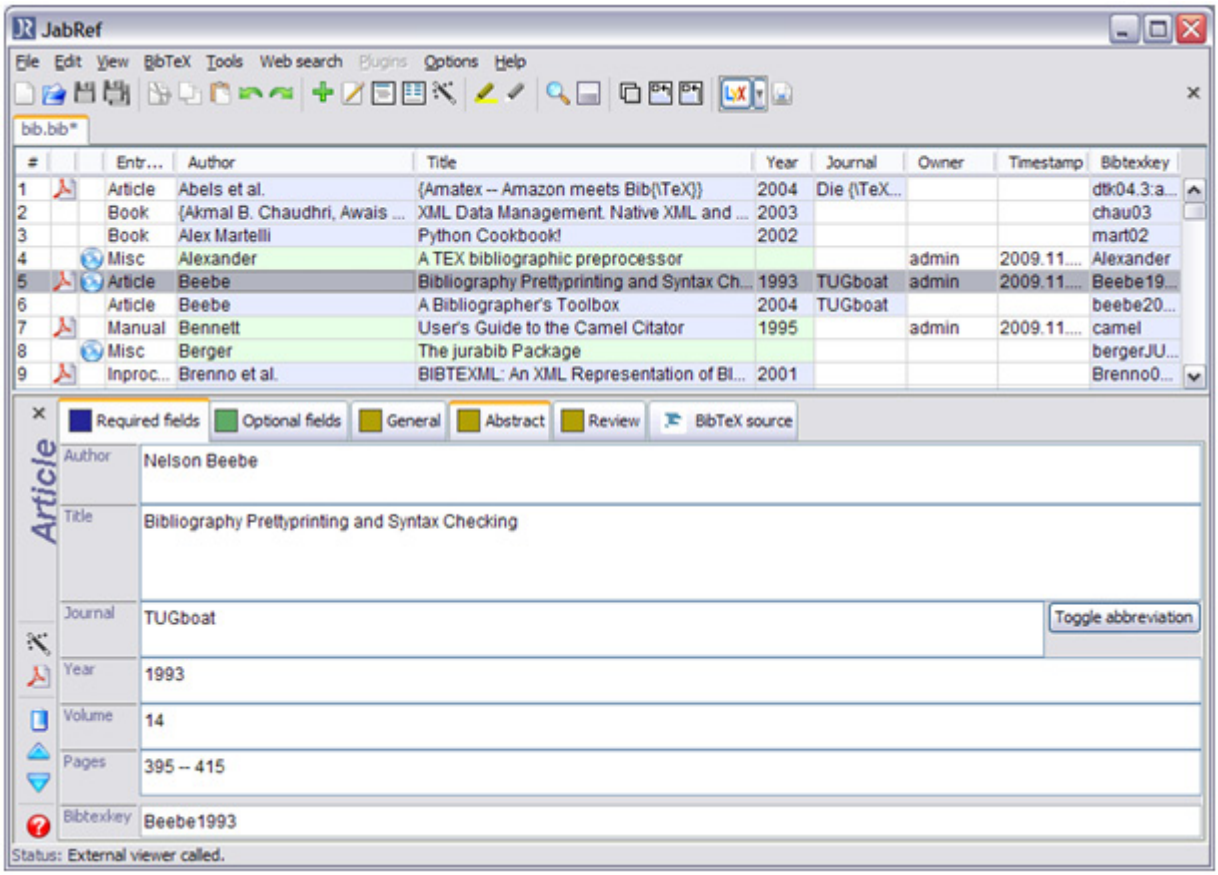

Abbildung 2.7: Ansicht JabRef

Pybliographer

Pybliographer ermöglicht die komplette Übernahme von Medline-Daten nach BibTeX. Pybliographer ist ein in Python geschriebes Kommandozeilentool, das den Import von BibTeX, Ovid, Medline ermöglicht. Das Exportformat ist ausschließlich BibTeX.

Javabib

Javabib[27] ist ein in Java geschriebener BibTeX-Parser. Javabib besteht aus einer Reihe von Java Paketen.

2.5.4 Konverter BibTeX-Datenbanken zu anderen Formaten

Bibutils

Bibutils[28] ist ein Konvertierungsprogramm für bibliographische Daten, mit XML als Zwischenformat. Es konvertiert BibTeX, COPAC, EndNote, EndNote-XML, ISI, MedLine und das RIS-Format zu XML.

bib2xhtml

Bib2xhtml[29] ist ein Programm, das BibTeX-*bib*-Dateien in XHTML-Dateien umwandelt. Die Konvertierung wird mit speziellen BibTeX-Style-Dateien durchgeführt. Dabei werden die Zitierstile empty, plain, alpha, unsort und unsortlist unterstützt. Standard BibTeX-Style-Dateien werden nicht unterstützt.

Bib2x

Bib2x ist ein Konverter, der es ermöglicht, BibTeX-*bib*-Dateien in jedes ASCII/UTF8 Dateiformat zu exportieren. Dabei wird eine flexible Vorlagemethode angewandt. Bib2x ist in PHP geschrieben und ist plattformunabhängig.

bib2xml

Bib2xml[30] konvertiert ein BibTeX-*bib*-File in ein XML-File. Der Output ist eine formatierte XML Datei, die mit BibTeXML kompatibel ist.

[27] http://www.plan.cs.colorado.edu/henkel/stuff/javabib/
[28] http://www.scripps.edu/~cdputnam/software/bibutils/
[29] http://www.spinellis.gr/sw/textproc/bib2xhtml/
[30] http://www.plan.cs.colorado.edu/henkel/stuff/bib2xml/

3 Funktionsweise von BibTeX

Um eine Neuentwicklung von BibTeX vornehmen zu können, muss dessen Funktionsweise analysiert werden. Dabei soll der Zusammenhang mit LaTeX, die internen Funktionen sowie der Aufbau einer *bst*-Datei exakt beschrieben werden. Die Besonderheiten von BibTeX und die daraus resultierenden Schwierigkeiten hinsichtlich einer Neuimplementierung sollen besonders hervorgehoben werden.

3.1 LaTeX-BibTeX Workflow

BibTeX erlaubt es externe bibliographische Datenquellen in LaTeX einzubinden. Dieses Kapitel soll den Arbeitsablauf und Datenfluss zwischen LaTeX-BibTeX erläutern.

3.1.1 LaTeX „thebibliography“ environment

Um Literaturverweise und Literaturverzeichnisse zu erstellen wird unter LaTeX die „thebibliography“ Umgebung eingesetzt. Dafür muss eine Liste von Literatureinträgen erstellt werden. Jeder Eintrag besteht aus einem Label (in eckiger Klammer) und einer Quellenmarke (in geschwungener Klammer), danach folgt der Literatureintrag. Dem Aufruf „thebibliography“ folgt eine Mustermarke, nach deren Länge[1] die Einrückungstiefe der Literaturliste ermittelt wird. Mustermarke, Label, Quellenmarke und Literatureinträge können per Hand editiert und sortiert werden. Formatierungsrichtlinien hinsichtlich der Einträge, Label und Sortierung müssen manuell bearbeitet werden. Ein Literaturverzeichnis kann auch aus einer externen Datenbank extrahiert werden, dabei wird in der Dokumentenpräambel \bibliography als Argument der Name der entsprechenden Datenbank (.bbl- Datei) angegeben.

Beispiel für eine Bibliographie:

[1] sollte der Länge der längsten Quellenmarke entsprechen

Listing 3.1: Beispiel: Bibliographie *bbl*-File

```
\begin{thebibliography}{}
\bibitem[Lamport:, 1995]{lamp:95}
Lamport:, L. (1995).
\newblock Addison-Wesley, 1. edition.
\bibitem[Lingnau, 2007]{ling:07}
Lingnau, A. (2007).
\newblock OREILLY, 1. auflage mai 2007 edition.
\end{thebibliography}
```

3.1.2 BibTeX

BibTeX, entwickelt von Oren Patashnik [Pat88a], ermöglicht es, Einträge der „thebibliography“ Umgebung aus externen Literaturquellen (*bib*-Dateien) zu generieren. Der bibliographische Eintrag wird nicht direkt im LaTeX-Dokument abgelegt, sondern in einer externen Datei (*bbl*). gespeichert. Die Formatierung der Einträge und der Labels sowie die Sortierung werden über Styles definiert (*bst*-Dateien).

3.1.3 Arbeitsweise LaTeX-BibTeX

Der Datenaustausch zwischen LaTeX-BibTeX erfolgt in beide Richtungen ausschließlich über Dateien:

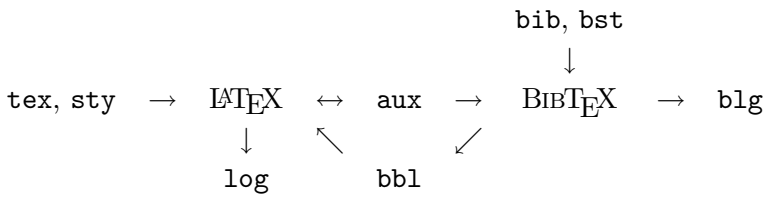

Abbildung 3.1: Datenaustausch der Dateien zwischen LaTeX und BibTeX

Im Zusammenspiel LaTeX-BibTeX sind vier Arbeitsschritte notwendig:

Schritt 1: LaTeX-Compiler (1)
Input ist das *tex*-File. Das *bib*-File muss noch nicht vorhanden sein. Im ersten Schritt hat die \nocite{*} Anweisung aus dem *tex*-File noch keine Auswirkung.
Bezogen auf BibTeX ist der Output das *aux*-File. Gewisse Elemente sind für diesen

Schritt nicht zwingend notwendig, werden aber ins *aux*-File geschrieben.

Zum Beispiel: Output *aux*-File:

```
\relax
\bibstyle{style.bst}  --> Style wird an BibTeX durchgereicht
\citation{citekey1}
\citation{citekey2}
\citation{citekey3}
\bibdata{data.bib}    --> .bib-Files werden an BibTeX durchgereicht
```

Bibliographische Einträge sind zu diesem Zeitpunkt noch unsortiert. Das Encoding entspricht dem des *tex*-Files. Folgende Dateien werden zwar von LaTeX geschrieben, sind jedoch für BibTeX nicht relevant:
teststyl.dvi
teststyl.log

Schritt 2: BibTeX-Compiler
Input für den BibTeX-Compiler sind die *aux*-Dateien[2], die *bib*- und *bst*-Dateien. Das *aux*-File wird in diesem Schritt nicht geändert.
Output ist das *blg*[3]- und das *bbl*-File. Das *bbl*-File bindet die Literatureinträge in der LaTeX „thebibliography“ Umgebung.
Literatureinträge mit thebibliography:

```
\begin{thebibliography}{99} % => longest label
  \bibitem{label1}
      Entry text
  \bibitem{label2}
      Entry text
\end{thebibliography}
```

Die Einträge sind bereits sortiert und mit dem entsprechenden Label versehen.

Schritt 3: LaTeX-Compiler (2)
Output ist ein erweitertes *aux*-File. Bibliographische Einträge werden mit dem Kommando \bibcite{citekey2}{1} sortiert eingefügt.

[2] *Aux*-Dateien beinhalten u.a. die Information, welche *bib*-Dateien geladen werden sollen
[3] Log-File des BibTeX-Compiler

```
\bibstyle{style.bst}
\citation{citekey1}
\citation{citekey2}
\citation{citekey3}
\bibcite{ citekey2}{label2}
\bibcite{ citekey1}{label1}
\bibcite{ citekey3}{label3}
\bibdata{data.bib}
```

Schritt 4: LaTeX-Compiler (3)
In den Ausgabedateien von LaTeX (*dvi*, *pdf*) werden die Zitatreferenzen richtig gesetzt.

3.1.4 Probleme mit MultiByte Encoding

BibTeX hat intern beim Generieren von Referenzschlüsseln einige Probleme mit multibyte Encoding wie UTF8. Bei der bibstyle plain verwendet wird, können UTF8-BibTeX-Files ohne Probleme verwendet werden. Wenn ein Verwendung von alphanumerischen Styles wie *alpha*, können Fehler auftreten, wenn Sonderzeichen im Schlüssel vorkommen.

Ablauf LaTeX und BibTeX mit UTF8
Im ersten Schritt schreibt LaTeX das *aux*-File (Encoding = system default). Dabei hat LaTeX kein Problem beim Lesen einer UTF8 *tex*-Datei. Probleme treten erst beim Sortieren auf, wenn der Zitierschlüssel ein Sonderzeichen beinhaltet. Im zweiten Schritt stellt BibTeX die Literatur zusammen und erstellt dabei die *bbl*-Datei (Encoding ist gleich der *bib*-Datei). Die *bbl*-Dateien werden problemlos in UTF8 geschrieben.

3.2 Aufbau eines BibTeX-Styles

Die Formatierung der Labels, Literatureinträge und deren Sortierung werden mittels BibTeX-Styles definiert. BibTeX-Styles werden in Dateien mit der Endung *.bst* abgespeichert. Der Inhalt von *bst*-Dateien ist ein in einer eigenen Programmiersprache geschriebenes Programm. Diese Programmiersprache hat eigentlich keinen Namen, wird aber häufig als *bst* oder BibTeX-Style bezeichnet. Im Grunde ist BibTeX ein Interpreter der *bst*-Sprache. Dass heißt, das Erzeugen der *bbl*- und der *blg*-Datei ist mittels Programmcode nachvollziehbar und somit transparent. Demgegenüber stehen interne Funktionen, die das Lesen der *bib*-Dateien implizit, auf sehr monolitsche Art und Weise durchführen.
Zuerst liest BibTeX die *aux*-Datei, die den Namen der *bst*- und *bib*-Dateien beinhaltet,

anschließend die *bst*-Datei, die die Anweisung zum Öffnen der *bib*-Datei beinhaltet. Die Anweisung READ öffnet die Dateien, die Anweisung ITERATE {call.type$} iteriert durch die Datenbank. Die Funktion call.type$ ruft entsprechend dem Eintragstyp eine Funktion auf.
BibTeX beinhaltet die Sprache *bst*, die von Oren Patashnik [Patashnik1988] speziell für die Anforderungen der bibliographischen Umgebung von Latex konzipiert wurde. Das Lesen und Schreiben von Dateien ist genau auf die Bedürfnisse des BibTeX-Datenflusses abgestimmt. *Bst* liest ausschließlich Dateien mit der Endung *aux* und *bib* und schreibt in die Dateien *bbl* und *blg*. Die Sprache *bst* besteht aus 10 Anweisungen und 37 internen Funktionen, die auf spezielle Anforderungen von LaTeX abgestimmt sind. Andererseits gibt es keine Funktionen zum Öffnen beliebiger Dateien. Viele Funktionen und Anweisungen, die man von Standardprogrammiersprachen gewohnt ist, fehlen.
Die Funktion format.names$ übernimmt zum Beispiel auf BibTeX spezifische Art die Zerlegung von Namenseinträgen. Neben den internen BibTeX-Funktionen gibt es noch eine Reihe von *bst*-Funktionen, die in den Standard-*bst*-Dateien (plain.bst, alpha.bst, abbrv.bst und unstr.bst) implementiert sind.
Die Funktionen AND und OR sind nicht direkt im BibTeX Sprachumfang enthalten, sondern als Funktion implementiert. Praktisch gesehen besteht BibTeX aus einer zweiten Schicht an Funktionen, nämlich den Funktionen, die sich aus den BibTeX-Standard Style-Dateien ergeben. Intern vordefinierte Funktionen erkennt man daran, dass sie mit einem Dollarzeichen enden. Theoretisch gesehen besteht BibTeX aus diesem internen Sprachumfang. In der Praxis sollten auch die selbstdefinierten Funktionen aus den Standarddateien als eine Art pseudo-Bibliothek (Spracherweiterung) gesehen werden. Die Schwierigkeit besteht darin, diese Funktionen zu erkennen. Auf den ersten Blick ist nicht ersichtlich, welche Funktionen allgemeiner Sprachgebrauch sind und welche Funktionen Style-spezifisch sind.
Standard-*bst*-Dateien halten sich, vor allem bei Funktionsnamen, an genaue Programmierrichtlinien. Dadurch kann eine Selektion empirisch gemacht werden und ist in gewisser Weise Interpretationssache. Im Detail betrachtet unterscheiden sich die Funktionen der Dateien (plain.bst, alpha.bst, abbrv.bst und unstr.bst) nur sehr gering. Da BibTeX nicht modular ist, das heißt es können keine Bibliotheken eingebunden werden, müssen diese Funktionen in jeden neu zu implementierten Style kopiert werden.
Bst ist eine Stack-basierende UPN Sprache[4], Anweisungen werden in Postfix-Notation gegeben. In der Postfix-Notation werden zuerst die Operanden, dann der Operator geschrieben, wie zum Beispiel:

```
1 +2 +

X Y =
```

[4]Umgekehrte Polnische Notation

```
  1
  2
if
```

entspricht der Infix-Notation:

```
1+2

If X=Y
  Then 1
  Else 2
```

UPN arbeitet mit Stack-Operationen, das heißt Argumente und Return-Werte werden über einen Stapelspeicher verarbeitet.
So hat beispielweise die Operation 2 1 + folgende Aktionen zur Folge:
Der Wert 1 wird in den Stapel gestellt, danach der Wert 2 darüber gestellt. Der Operator + erwartet zwei Argumente, die er sich vom Stapel holt, das heißt diese Werte werden aus dem Stapel entfernt. Das Ergebnis der Wert 3 wird in den Stapel gestellt.
Soll der zweite Wert ausgelesen werden, muss der erste und der zweite Wert mit der swap$ Anweisung ausgetauscht werden. Diese Konstruktionen haben zur Folge, dass *bst* nicht einfach zu lesen ist.
Nebenbei gibt es noch Variablen, unterteilt in STRING und INTEGER, mit unterschiedlichem Geltungsbereich (lokal, global) und vordefinierten Variablen. Viele Variablen werden im Programm als Konstanten verwendet. Den Datentyp Konstante gibt es nicht.

3.2.1 Bibtex‘s Hello World

Dieser einfache Versuch, ein möglichst einfaches *bst*-Programm zu schreiben, soll die grundsätzliche Arbeitweise von BibTeX veranschaulichen.

Listing 3.2: BibTeX Hello World

```
1 ENTRY {}{}{}
2 FUNCTION {test}
3 {"Hello World" write$ newline$
4 "Hello World" warning$ newline$}
5 READ
6 EXECUTE{test}
```

In Zeile 1 werden die Deklarationen durchgeführt. Auch wenn keine Deklarationen gemacht werden, muss aus syntaktischen Gründen diese Zeile angegeben werden.
Zeile 2-4 definiert eine Funktion mit dem Namen test. Eine Funktion im Quelltext muss immer vor ihrem Aufruf stehen. Die Funktion test schreibt mit der Funktion write$ in die *bbl*-Datei und in das *blg*-File (Log File) mit der Funktion warning$.
In Zeile 5 wird die READ-Anweisung gesetzt. Diese liest das *bib*-File aus, das in der *aux*-Datei angeführt ist. Auch wenn in diesem Programm das *bib*-File nicht benötigt wird, würde das Weglassen dieser Anweisung einen syntaktischen Fehler verursachen und zu einem Abbruch führen. In Zeile 6 wird die oben angeführte Funktion ausgeführt.

Ausführen von HelloWorld.bst:
Bevor BibTeX ausgeführt werden kann, muss LaTeX ausgeführt werden, um das notwendig *aux*-File zu erzeugen.

LaTeX-Ausgabe auf der Konsole:

```
Erzeuge Projekt: HelloWorld
------------------------------------------------
helloworld.tex...
------------------------------------------------
HelloWorld - 0 Fehler, 0 Warnung(en), 0 Overfull Box(en), 0 Underfull
  Box(en)
```

Im zweiten Schritt wird BibTeX ausgeführt. Die Leerdeklaration der Felder ENTRY führt zu einer Warnung:

```
Warning--I didn't find any fields--line 1 of file helloWorld.bst
```

Interessant in diesem Zusammenhang ist, dass sich sowohl die BibTeX internen Meldungen als auch die selbst definierten Warnungen (Warning–Hello World) in dieser Ausgabe befinden.

BibTeX-Ausgabe (Konsole, Logdatei blg):

```
This is BibTeX, Version 0.99c
The top-level auxiliary file: HelloWorld.aux
The style file: halloWorld.bst
Warning--I didn't find any fields--line 1 of file helloWorld.bst
Database file #1: HelloWorld.bib
Warning--Hello World
```

```
(There were 2 warnings)
```

Schlussendlich das erzeugte *bbl*-File:

```
Hello World
```

Um den eigentlichen Zugriff von BibTeX auf die Datenbank zu veranschaulichen, muss das Beispiel um die Anweisung ITERATE und um die Funktion call.type$ erweitert werden.

Listing 3.3: Einfaches bst-File mit Datenbankzugriff

```
1  ENTRY
2  { author
3  title
4  }
5  {}
6  {}
7  FUNCTION {book}
8  { "Cite =" write$ cite$ write$  newline$
9  " Author = " author * write$ newline$
10 " Title  = " title  * write$ newline$
11 }
12 READ
13 ITERATE {call.type$}
```

Der Aufbau eines *bib*-Files beschreibt drei Schlüsselelemente:

1. den Eintragstyp (@Book)
2. die Quellenmarke (chicago-82)
3. die Eintragsfelder (title, author...)

```
@Book{chicago-82,
  title =         'The {C}hicago Manual of Style',
  author =        ``Chicago'',
  year =          '1982',
  publisher =     'University of Chicago Press',
  edition =       '13th',
}
```

Mit dem Aufruf des ITERATE-Kommandos wird durch das Datenbank-File iteriert. Das heißt alle Einträge werden in einer Schleife durchlaufen. Die Funktion call.typ$ ruft eine Funktion auf, die dem Typ book entspricht und die Werte der Felder in die jeweilig gleichnamigen ENTRY-Variablen speichert. In dem oben angeführten Beispiel sind das author und title:

```
ENTRY{ author  title}
```

Die aufgerufene Funktion entspricht im Namen exakt dem Eintragstyp:

```
FUNCTION {book}
```

Der Zugriff auf die Quellenmarke (Zitierschlüssel) erfolgt über die vordefinierte Variable cite$.
Anhand dieses Beispiels ist zu erkennen, wie generisch BibTeX in Wirklichkeit ist. BibTeX kann beliebige Eintragstypen und Felder deklarieren und in Folge auch verarbeiten. Die in der Literatur angegebenen Entry-Typen (bib types) und Feldtypen (entry fields) werden somit nicht direkt von BibTeX vorgegeben, sondern vom *bst*-File. Technisch gesehen kann ein Style beliebige Eintragstypen mit beliebigen Feldtypen mittels BibTeX verarbeiten. Praktisch gesehen wird von den Standard-Styles ausgegangen (plain, unsrt, alpha, abbrv).
Eine Besonderheit ist, dass jede Funktion, auch Standard-BibTeX-Funktionen wie and und or, als Eintragstyp aufgerufen werden kann. Prinzipiell sind alle gewöhnlichen Zeichen möglich, auch durch Punkt getrennte Funktionen.
Einen besonderen Typ stellt DEFAULT.TYPE dar, eine eingebaute BibTeX-Variable (ohne das Dollarzeichen). Eintragstypen, für die keine spezielle Funktion gefunden wird, werden mit der speziellen Funktion default.type verarbeitet. Dadurch wird ein Fehlerabbruch durch BibTeX vermieden.

3.3 BibTeX-Anweisungen

Anweisungen unterliegen in BibTeX einer strikten Anordnung. Sie teilen das Programm in unterschiedliche Bereiche auf und bilden die Grundstruktur eines Styles. Anweisungen können im Wesentlichen in drei Gruppen eingeteilt werden:

- Deklarationen
- Ausführung von Funktionen
- Ausführung von internen Aktionen

3.3.1 Deklarationen

ENTRY
In diesem Anweisungsblock erfolgt die Deklaration der benutzten Felder aus der Literaturdatenbank, der lokalen Integer- und Stringvariablen. Definierte Felder werden in den

Funktionen wie Variablen angesprochen, die durch das Kommando ITERATE pro Eintrag befüllt werden. Lokale Variablen sind ähnlich zu betrachten. Das Attribut lokal bezieht sich nicht auf den Geltungsbereich einer Funktion, sondern eines Datensatzes (Eintrag) aus der Literaturdatenbank. Somit kann man sie auch als interne Variable oder als von der Literaturdatenbank unabhängige Feldwerte bezeichnen. So verwenden beispielweise Styles, die alphanumerische Labels erzeugen, die Variable label für die Erzeugung eines Labels. Die Besonderheit dieses Kommandos ist, dass es existieren muss, auch wenn keine Deklarationen vorgenommen werden, sonst hätte dies einen Syntaxfehler zur Folge.

STRING/INTEGERS
String/Integers definiert gewöhnliche globale Variablen. Diese Variablen behalten ihren Wert über den gesamten iterativen Ablauf. String-Variablen sind kostenintensiv und auf die Anzahl 20 beschränkt.

MACRO
Makros dienen für die Ersetzung von Text und werden mit zwei Argumenten deklariert, dem zu ersetzenden und dem ersetzenden Wert. Das zweite Argument muss mit einem Doppelhochkomma eingeschlossen sein, zum Beispiel:

```
MACRO {jan} {‘january’}
```

Ersetzungen, die im *bib*-File mit dem @string-Kommando definiert wurden, werden bevorzugt behandelt.

FUNCTION
Dieses Kommando definiert Funktionen, die später aufgerufen werden können. Im Gegensatz zu vordefinierten Funktionen, die mit einem Dollarzeichen enden, kann ein beliebiger Name vergeben werden. Zusammengesetzte Wörter werden mit Punkt getrennt, Unterstrich, Leerzeichen, Zahlen und Sonderzeichen sind nicht erlaubt. Die Trennung der Wörter hat keine syntaktische Bedeutung. In Standard-*bst*-Dateien dient sie aber als Kodierrichtlinie, um die Lesbarkeit zu erhöhen.
Funktionen, deren Namen einem Eintragstyp entsprechen, werden vom Kommando ITERATE{call.type$} direkt aufgerufen.

Beispiel für Entrytype-functions:

```
FUNCTION {article}        article
FUNCTION {book}           book
```

```
FUNCTION {booklet}         booklet
FUNCTION {inbook}          inbook
FUNCTION {incollection}    incollection
FUNCTION {inproceedings}   inproceedings
FUNCTION {manual}          manual
FUNCTION {mastersthesis}   mastersthesis
FUNCTION {phdthesis}       phdthesis
FUNCTION {proceedings}     proceedings
FUNCTION {techreport}      techreport
FUNCTION {unpublished}     unpublished
```

Die Funktion FUNCTION {misc} stellt eine Ausnahme dar. Sie wird durch die eingebaute Funktion default.type aufgerufen, die immer dann zieht, wenn keine entsprechende Funktion für einen Eintragstyp existiert.

3.3.2 Ausführung von Funktionen

EXECUTE
Execute dient zum Ausführen einer Funktion. Die auszuführende Datei wird als Argument übergeben, sie muss zuvor definiert worden sein.

ITERATE
Iterate führt die Funktion aus, die als Argument übergeben wurde. Die Funktion wird für jeden Eintragstyp, der durch die Anweisung READ gelesen wurde, ausgeführt.

REVERSE
Reverse dient zum Aufrufen einer Funktion, die auf alle Literatureinträge in gegebener bzw. umgekehrter Reihenfolge angewandt wird[5].

3.3.3 Ausführung von internen Aktionen

READ
Read dient zum Einlesen der Literaturdatenbank. Ähnlich wie das ENTRY Kommando ist es syntaktisch vorgeschrieben. Es muss genau einmal vorkommen. Die Anweisungen MACRO und ENTRY müssen vor dieser Anweisung stehen. Im Gegensatz dazu machen ITERATE und REVERSE nur anschließend Sinn.

SORT

[5] Wird von alpha.bst verwendet, um extra Label zu vergeben (bei Mehrfachvorkommen eines Labels werden die Zeichen a,b,c ... in umgekehrter Reihenfolge angehängt)

Sortieren der Literatureinträge anhand der String-Variable sort.key$. Das macht nur Sinn, wenn in einem vorangegangenen Schritt die Variable sort.key$ in einer Iterationsschleife für jeden Eintrag ermittelt wurde. Jeder nachfolgende ITERATE Aufruf iteriert in der sortierten Reihenfolge. Analog zu ITERATE kann REVERSE aufgerufen werden, wenn in absteigender Reihenfolge sortiert werden soll. Wenn das SORT-Kommando nicht aufgerufen wird, werden die Einträge in derselben Reihenfolge wie sie in der *bib*-Datei vorkommen, ausgegeben.

3.3.4 Struktur eines BibTeX-Styles

Die Struktur eines *bst*-Files ist im Wesentlichen durch die richtige Reihenfolge (Anordnung der Anweisungen) genau vorgegeben.
Der typische Aufbau einer *bst*-Datei:

```
ENRTRY { Felder }{ locale Integer Variablen }{ locale String Variablen
   }
INTEGERS { globale Integer Variablen}
STRINGS { globale String Variablen}

MACRO {}
FUNCTIONS {}

READ
EXECUTE {begin bib}
ITERATE {presort}
SORT
ITERATE {label}
ITERATE {call type}
EXECUTE {end bib}
```

An erster Stelle steht die Deklaration der Felder (Entry Fields) und Variablen. Danach werden Makros und Funktionen deklariert. Funktionen müssen in korrekter Reihenfolge angeführt werden, das heißt die aufgerufene vor der aufrufenden Funktion. Wenn eine Funktion nach einem bibliographischen Typ benannt ist (Entry Type), kann sie auch als Deklaration derselben angesehen werden.
Anschließend folgt der Ausführungsblock. Die Anweisung READ hat keine Argumente, sie liest das (in der *aux*-Datei definierte) *bib*-File. Jede EXECUTE oder ITERATE-Anweisung ist ein neuer Einstiegspunkt in das Programm, sie können nicht verschachtelt werden. Sie werden in sequentieller Reihenfolge ihres Auftretens ausgeführt. Typisch für einen BibTeX-Style sind zwei EXECUTE-Anweisungen für nicht iterative Aktionen vor und nach dem die bibliographischen Einträge in einer Schleife verarbeitet werden, sowie drei

ITERATE Kommandos, Sortieren, Berechnen der Label und der Aufruf der Eintragstypen.
Der Sortiervorgang geschieht in zwei Schritten, in einem iterativen wird für jeden bibliographischen Eintrag ein Sortierschlüssel ermittelt, anschließend mit dem Kommando SORT die eigentliche Sortierung vorgenommen.

3.4 BibTeX-interne Funktionen

BibTeX-Funktionen beziehen ihre Argumente über den Stapelspeicher. Werte werden über den Stapelspeicher bezogen. Die Rückgabe erfolgt ebenfalls über den Stapel, indem die Werte dort abgelegt werden. Das bedeutet, dass Anzahl und Typ der Argumente nicht syntaktisch geprüft werden. Bei Laufzeit provozieren fehlende Werte oder falsche Typen einen Ablauffehler.
Bei der Reihenfolge der Werte im Stapelspeicher ist zu beachten, dass der Wert, der zuletzt geschrieben wurde (im Code von oben nach unten gelesen), an erster (oberster) Stelle im Stapel ist.

3.4.1 Wertezuweisung

:= weist dem ersten Element aus dem Stapel, das eine Variable sein muss, den Wert aus dem zweiten Literal zu.

3.4.2 Ausgabe in *bbl*/*blg*-Datei

Ausgabefunktionen sind speziell dazu konzipiert, Werte aus dem Stack in Dateien zu schreiben. Dabei handelt es sich ausschließlich um die Dateien des BibTeX-Workflows (*bbl*, *blg*).

newline$ schreibt alles, was sich im Ausgabebuffer angesammelt hat, inklusive einem Zeilenumbruch, in die *bbl*-Datei. Wenn der Buffer leer ist, wird eine Leerzeile geschrieben.

write$ holt den ersten Wert aus dem Stack und schreibt diesen in den Ausgabebuffer für das *bbl*-File.

warning$ schreibt eine numerierte Warnmeldung. Warnmeldungen resultieren im *blg*-File und in der Bildschirmausgabe.

3.4.3 Kontrollfluss

Die Funktion **if$** benötigt drei Werte aus dem Stapel. Die ersten beiden müssen Funktionsliterale sein. Das dritte muss eine ganze positive Zahl sein (meistens das Ergebnis einer Prädikatsprüfung). 0 entspricht dem Wahrheitswert „Falsch“ und führt zur Ausführung der zweiten Funktion, Zahlen, die größer als 0 sind führen zur Ausführung der ersten Funktion, wie im folgenden Beispiel.

```
titel empty$
   'pop$
   'output.nonnull
 if$
```

Die Funktion pop$ wird ausgeführt, wenn die Variable titel leer ist, sonst die Funktion output.nonnull[6].

while$ führt die Funktion, die an erster Stelle im Stack steht, solange in einer Schleife aus, bis die darunterliegende Prädikatsprüfung größer als 0 ist.

3.4.4 Typenkonvertierung

Konvertierungsfunktionen wandeln immer den obersten Wert aus dem Stapelspeicher um.

int.to.str$ wandelt einen numerischen Wert in einen Textwert um.

int.to.chr$ wandelt einen ASCII-Wert zwischen 0 und 127 in das entsprechende Textzeichen um.

chr.to.int$ wandelt ein Textzeichen in den entsprechenden ASCII-Code um. Der Wert muss zwischen 0 und 127 liegen.

[6]Der Datentyp boolean exsistiert in BibTeX nicht.

3.4.5 Stack-Operationen

pop$ nimmt den ersten Wert aus dem Stapel.

swap$ tauscht die beiden ersten Werte des Stapelspeichers aus.

duplicate$ dupliziert den ersten Wert aus dem Stapelspeicher.
Diese Funktion findet Anwendung, wenn ein Wert aus dem Stack einer Funktion übergeben werden soll. Da die meisten Funktionen den verwendeten Wert aus dem Stapel löschen, kann mit dieser Funktion das Verbleiben des Wertes gewährleistet werden. Damit können Funktionen mit Übergabeparameter(n) geschrieben werden, die keinen Einfluss auf den Zustand des Stapelspeichers haben, wie zum Beispiel:

```
FUNCTION {output}
{ duplicate$ empty$
    'pop$
    'output.nonnull
  if$
}
```

Die Funktion output prüft, ob der erste Wert aus dem Stapel leer ist, ohne ihn aus dem Speicher zu entfernen.

stack$ dient hauptsächlich zu Debugging-Zwecken. Der komplette Stack wird im Log ausgegeben und gelöscht.

3.4.6 Globale Konstanten

entry.max$ ist eine interne integer-Konstante mit dem Wert 250.

global.max$ ist eine interne integer-Konstante mit dem Wert 5000.

3.4.7 Integer-Operationen

(Addition) „+“ holt die ersten beiden Werte und schreibt die Summe auf den Stack.
(Subtraktion) „-“ holt die ersten beiden Werte und schreibt die Differenz auf den Stack (den ersten subtrahiert vom zweiten Wert).

3.4.8 Prädikate

Prädikatsprüfungen benötigen ein oder zwei Argumente aus dem Stack. BibTeX kennt den Datentyp boolean nicht, vielmehr wird 1 für „Wahr“ bzw. 0 für „Falsch“ verwendet[7].

„>“ vergleicht die ersten beiden Werte aus dem Stack und schreibt 1, wenn der zweite größer ist, sonst 0.

„<“ vergleicht die ersten beiden Werte aus dem Stack und schreibt 1, wenn der erste größer ist, sonst 0.

„=“ schreibt 1 auf den Stack, wenn die ersten beiden Werte übereinstimmen, sonst 0.

empty$ schreibt 1 in den Stack, wenn der erste Wert leer ist, sonst 0.

**missing$** prüft, ob ein definiertes Feld in einem Eintrag existiert. Im Unterschied zu empty$, bezieht sich diese Prüfung auf Eintragsfelder von Bibliographien.

3.4.9 Spezielle Funktionen

skip$ führt keine Operation durch.

call.type$ führt die namensgleiche Funktion eines Eintragstyps aus. Üblicherweise wird sie als Argument für die Anweisung ITERATE verwendet. Liegt für einen Eintragstyp keine Funktion vor, wird die interne Funktion default.type ausgeführt.

Die Funktion **preamble$** dient dazu, @PREAMBLE-Einträge aus dem *bib*-File zu verarbeiten. Dabei werden alle Vorkommen in ihrer Reihenfolge zusammengesetzt und auf den Stack geschrieben. Diese Einträge werden üblicherweise vor den Literatureinträgen in das *bbl*-File geschrieben und beinhalten gewisse TeX-Steuersequenzen.

3.4.10 Textfunktionen

Textfunktionen dienen in der Regel zum Manipulieren von Texteinträgen. In den meisten Fällen wird ein Wert aus dem Stack geholt, bearbeitet und wieder zurückgelegt. In einigen Fällen werden zusätzliche Werte aus dem Stack geholt, sie dienen als Parameter

[7] Genaugenommen gelten alle Werte größer 0 als „Wahr“.

für die jeweilige Funktion.

width$ ermittelt die tatsächliche Breite eines Strings in einer internen Maßeinheit. Diese Funktion wird zum Vergleichen der Länge von Strings verwendet, um das längste Label zu finden. Rückgabe ist ein relativer Wert, der auch die Breite von Zeichen und Tex-spezifische Merkmale berücksichtigt.

num.names$ ermittelt die Anzahl der Namen in einem String. Dabei werden Textstellen gezählt, die durch das Wort and getrennt sind.

text.length$ gibt die Anzahl der Zeichen eines Textes zurück. Spezielle Zeichen, die aus mehreren Bytes bestehen, werden nur als ein Zeichen gezählt. Klammern werden nicht berücksichtigt.

add.period$ hängt einen Punkt an das Ende eines Textes an, ausgenommen das letzte Zeichen ist ein Ruf- oder Fragezeichen.

Die Funktion **change.case$** benötigt die ersten beiden Werte aus dem Stack. Der erste Wert bestimmt, ob auf Groß- oder Kleinschreibung umgestellt wird. Der zweite Wert wird umgestellt. Diese Funktion hat drei unterschiedliche Argumente: „t“ wandelt das erste Zeichen auf Groß- , alle folgenden auf Kleinbuchstaben um. „u“ wandelt alle Zeichen auf Großbuchstaben, „l“ auf Kleinbuchstaben um.

(Concatenation) „*“ verbindet die ersten beiden Werte aus dem Stapelspeicher und gibt das Resultat zurück.

purify$ entfernt alle nicht alphanumerischen Zeichen aus einem String. Ausgenommen sind Leerräume (whitespace).

Die Funktion **substring$** benötigt drei Werte vom Stapelspeicher. Sie gibt eine Teilmenge vom dritten Wert zurück. Der zweite Wert beschreibt die Position, ab der abgeschnitten wird, der erste Wert die maximale Länge.

quote$ schreibt ein Doppelhochkomma in den Stack.

**text.prefix$** ist vergleichbar mit substring$, berücksichtigt und bereinigt aber Klammern. Der Rückgabewert wird auf eine bestimmte Anzahl von Zeichen abgeschnitten, wobei geschwungene Klammern nicht mitgezählt werden. Bestimmte TeX spezifische Zeichen, die aus mehreren Bytes gebildet werden, werden nur als ein Zeichen gezählt.
Ziel dieser Funktion ist es, den String im kompilierten Dokument abzukürzen und nicht die Anzahl der Zeichen im Quellcode.

3.4.11 BibTeX-Interne Variablen

Neben internen Funktionen gibt es in BibTeX auch interne Eintragsvariablen. Sie werden auch als „automatisch deklarierte Variablen“ bezeichnet.

crossref ist ein weiterer Feldwert, der automatisch deklariert wird. Anders als andere BibTeX-Felder, muss dieser Eintrag nicht mit der Anweisung ENTRY deklariert werden. Dieser Wert wird für Querverweise in bibliographischen Einträgen verwendet.

Die Variable **cite$** übernimmt den Wert des Zitierschlüssels, der durch das LaTeX-\cite Kommando gesetzt wurde.

sort.key$ ist eine spezielle Variable, die durch einen iterativen Prozess für jeden Eintrag befüllt wird. Dafür stellt jeder Style eigens definierte Formatierungsfunktionen zur Verfügung. Nach Ausführung der SORT-Anweisung werden sämtliche bibliographischen Einträge nach diesem Schlüssel aufsteigend sortiert. Wenn die Einträge in absteigender Reihenfolge ausgegeben werden, müssen sie mit der REVERSE Anweisung anstatt ITERATE-aufgerufen werden.

type$ diese Variable beinhaltet den aktuellen Eintragstyp (book, article, etc.) eines bibliographischen Eintrags.

3.4.12 Namensformatierung

Die Funktion **format.name$** ist die komplexeste interne BibTeX-Funktion [markey2005]. Sie erlaubt es auf BibTeX spezifische Art, Teilmengen aus Textketten zu lesen. Sie wird verwendet um Namensfelder (zum Beispiel author, editor) in einzelne Bestandteile zu zerlegen. Die Zerlegung geschieht in zwei Stufen. In der ersten Stufe werden, wenn notwendig, Namenslisten in einzelne Namen zerlegt. Dabei verwendet BibTeX als Trennungsmerkmal das Wort and.
Die Funktion format.name$ verwendet drei Parameter vom Stack. Der erste gibt die Formatmaske an, das heißt wie ein Name formatiert werden soll. Der zweite Parameter ist die gesamte Namensliste, die verarbeitet werden soll. Der dritte Parameter gibt an, welcher Name (von links nach rechts) aus der Liste verarbeitet werden soll. Das heißt um die gesamte Namensliste zu extrahieren, muss über eine while$-Schleife iteriert werden. Die Anzahl der Schleifendurchläufe kann durch die Funktion num.names$ ermittelt werden, wie zum Beispiel:

```
n namelist "{ff~}{vv~}{ll}{, jj}" format.name$ 't :=
```

Eintrag	Vorname	von Teil	Nachname	jr Teil
AA BB	AA		BB	
AA			AA	
AA bb	AA		bb	
aa			aa	
AA bb CC	AA	bb	CC	
AA bb CC dd EE	AA	bb~CC~dd	EE	
AA 1B cc dd	AA~1B	cc	dd	
AA 1b cc dd	AA	1b~cc	dd	
AA bB cc dd	AA~bB	cc	dd	
AA bb cc dd	AA	bb~cc	dd	
AA Bb cc dd	AA	Bb~cc	dd	
AA BB cc dd	AA~BB	cc	dd	
AA \BBb cc dd	AA~\BBb	cc	dd	
AA \bbb cc dd	AA	\bbb~cc	dd	
AA bb cc DD	AA~bb	cc	DD	
AA bb cc DD	AA	bb	cc~DD	
AA bb CC	AA~bb		CC	
bb CC, AA	AA	bb	CC	
bb CC, aa	aa	bb	CC	
bb CC dd EE, AA	AA	bb~CC~dd	EE	
bb, AA	AA		bb	
BB,			BB	
bb CC,XX, AA	AA	bb	CC	XX
bb CC,xx, AA	AA	bb	CC	xx

Tabelle 3.1: Erkennungsmuster für Namensteilung

Das n-te Vorkommen vom Eintrag in namelist wird gemäß Formatmuster „{ff~}{vv~}{ll}{, jj}“ formatiert und über den Stack der Variable t zugewiesen.
In einer zweiten Stufe zerlegt BibTeX gemäß der vorgegebenen Formatmaske den Namen in vier mögliche Ausprägungen:

f = Vorname
l = Nachname
v = von Teil des Namens
j = Jr Teil des Namens

Gemäß der vorgegeben Formatmaske werden diese Elemente wieder zusammengesetzt. Wenn ein Buchstabe angegeben wird, dann wird nur der Anfangsbuchstabe ausgegeben, bei Doppelbuchstaben das gesamte Wort. Der Algorithmus, nach dem BibTeX einen Namen zerlegt, ist relativ komplex. Dabei werden Anzahl, Position und Groß- Kleinschreibung der einzelnen Textteile ausgewertet.
Erkennungsmuster für die Namenszerlegung werden in der Tabelle *3.1* angeführt.

3.5 BibTeX-Sprachmerkmale von benutzerdefinierten Funktionen

Bei benutzerdefinierten Funktionen wird nicht zwischen Klein- und Großschreibung unterschieden[8], ausgenommen sind Textkonstanten, die mit einem doppelten Hochkomma eingeschlossen werden. Das gilt sowohl für vordefinierte Funktionen, benutzerdefinierte Funktionen, als auch für Variablen. Leerräume (Leerzeichen, Tabulatoren, Zeilenumbrüche) sind absolut gleichwertig. Zeilenumbrüche haben keine Bedeutung, mit Ausnahme von Kommentaren[9].

3.5.1 Datentypen

Zahlenkonstanten (Werte) werden mit einer vorangestellten Raute geschrieben. Es können nur ganze Zahlen (Integer) verwendet werden, wie zum Beispiel:

```
#1
#-2
```

Stringkonstanten (Werte) werden mit Doppelhochkomma eingeschlossen. Konstanten sind case-sensitive.

```
"Wert"
""      ->entspricht NULL
```

Wenn einer Konstante das Doppelhochkomma angehängt werden soll, muss die Funktion quote$ verwendet werden

[8] Die *bst*-Sprache ist noncase sensitive.
[9] Ein Zeilenumbruch beendet einen Kommentar.

3.5.2 Variablen

Beim Zugriff auf den Wert einer Variable, wird nur deren Name angegeben. Wenn der Name einer Variable angesprochen wird, muss ein einfaches Hochkomma vorangestellt werden, wie zum Beispiel:

```
var1 ‘var2 :=
```

Der Variable var2 wird der Wert der Variable var1 zugewiesen. Variabelenamen beginnen immer mit einem Buchstaben, gefolgt von Buchstaben, Ziffern und sonstigen darstellbaren Zeichen.
Ausgenommen sind folgende Zeichen:

```
‘\% (),=ß\{\}#
```

Als übliches Trennzeichen wird der Punkt verwendet.

3.5.3 Die Bedeutung von „{}“ und „%“

„%“ leitet einen Kommentar ein.**„{}“** schließt einen Verarbeitungsblock (Literal) ein. Das kann eine benannte Funktion sein (normale Funktionsdeklaration) oder ein anonymer Block innerhalb einer Funktion. Im normalen Programmfluss ist diese Art der Gruppierung nicht relevant:

```
{a1 a2 a3} {a4 a5 } funct$
```

ist äquivalent zu:

```
a1 a2 a3 a4 a5 funct$
```

Literale werden für den Kontrollfluss bei logischen Operationen if$ oder Programmschleifen while$ verwendet.
Ein Beispiel für logische Operation (Wahrheitsprüfung):

```
  literal
  literal
if$
```

Ein einfaches Hochkomma „’“ vor einer Funktion ist äquivalent zu einem Literal.

```
'skip$
```

ist äquivalent zu:

```
{skip$}
%$
```

Wenn einer Variable ein Wert zugewiesen wird, muss sie auch als Literal angeführt werden. „'“ vor einem Variablennamen spricht den Namen der Variable an, wie im folgenden Beispiel:

```
`text' 't :=
```

Der Zuweisungsoperator := weist der Variable `t` den Wert „text“ zu.

3.5.4 Besonderheit der Parameter-Übergabe

Funktionen übergeben Werte hauptsächlich über den Stapelspeicher, Ausnahme ist die Verwendung von globalen Variablen. Es gibt weder In/Out-Parameter noch Rückgabewerte. Alle Aktionen laufen sequentiell ab. Innerhalb einer Funktion gibt es keine definierten Gültigkeitsbereiche, auch nicht, wenn die Funktion in Literale gegliedert ist.

3.5.5 Cross Referenzen

Eine Besonderheit von BibTeX sind die Querverweise (Crossref). Dafür ist ein internes Eintragsfeld deklariert. Diese Feldvariable bildet in BibTeX eine Ausnahme, da sie nicht durch die ENTRY-Anweisung deklariert werden muss.
Standard-Styles ermöglichen fünf Arten von Querverweisen:

1. Article -> Article
2. Book -> Book
3. Inbook -> Book
4. Incollection -> Book
5. Inproceedings -> Proceeding

Cross-Referenzen ermöglichen es, einen bibliographischen Eintrag auf einen anderen zu verweisen. Dafür wird ein eigenes Eintragsfeld verwendet. Das Feld crossref wird in der bibliographischen Datenbank wie jedes andere Eintragsfeld angegeben. Ausnahme ist aber, dass es in der *bst*-Datei nicht im ENTRY-Teil angeführt wird. Der Wert für dieses Feld ist ein cite key eines anderen Eintrags.
Beispiel für ein *bib*-File:

```
@BOOK{book-crossref,
   crossref = 'whole-set',
   title = 'Seminumerical Algorithms',
   volume = 2,
   series = 'The Art of Computer Programming',
   edition = 'Second',
   year = '{\noopsort{1973c}}1981',
   note = 'This is a cross-referencing BOOK entry',
}
@BOOK{whole-set,
   author = 'Donald E. Knuth',
   publisher = 'Addison-Wesley',
   title = 'The Art of Computer Programming',
   series = 'Four volumes',
   year = '{\noopsort{1973a}}{\switchargs{--90}{1968}}',
   note = 'Seven volumes planned (this is a cross-referenced set of
     BOOKs)',
}
```

Der Eintrag @book {book-crossref} hat im Feld crossref einen Verweis auf den Eintrag @book {whole-set}:
Das oben angeführte *bst*-File erzeugt nach Durchführung von BibTeX folgendes *bbl*-File:

```
\bibitem{whole-set}
Donald~E. Knuth.
\newblock {\em The Art of Computer Programming}.
\newblock Four volumes. Addison-Wesley,
  {\noopsort{1973a}}{\switchargs{--90}{1968}}.
\newblock Seven volumes planned (this is a cross-referenced set of
  BOOKs).
\bibitem{book-crossref}
Donald~E. Knuth.
\newblock {\em Seminumerical Algorithms}.
\newblock Volume~2 of {\em The Art of Computer Programming\/} \cite{
  whole-set},
  second edition, {\noopsort{1973c}}1981.
\newblock This is a cross-referencing BOOK entry.
```

Das Ergebnis im *dvi*-File wird in der Abbildung *3.2* (referenzierter Eintrag) und in der Abbildung *3.3* (referenzierender Eintrag) dargestellt.

[2] Donald E. Knuth. *The Art of Computer Programming.* Four volumes. Addison-Wesley, 1973a–901968. Seven volumes planned (this is a cross-referenced set of BOOKs).

Abbildung 3.2: Referenzierter Eintrag

[3] Donald E. Knuth. *Seminumerical Algorithms.* Volume 2 of *The Art of Computer Programming* [2], second edition, 1973c1981. This is a cross-referencing BOOK entry.

Abbildung 3.3: Referenzierender Eintrag

Je nach Typ werden von den Standard-Style-Funktionen folgende Eintragsfelder der referenzierten Einträge verwendet:

	ARTICLE	BOOK	INBOOK	INCOLLECTION	INPROCEEDINGS
volume		x	x		
title				x	x
series		x	x		
key	x	x	x	x	x
booktitle				x	x
journal	x				
editor		x	x	x	x
author		x	x	x	x

Tabelle 3.2: Felder für Querverweise

3.6 BibTeX-definierte Funktionen von Standard-Styles

Die folgenden bst's wurden von Oren Patarshnik [Pat88b] geschrieben und gelten als die klassischen Styles. Sie können als Quasi-Standard angesehen werden. Standard-Styles:

- **plain** - numerisches Label
- **unstr** - wie plain, aber unsortiert
- **alpha** - alphanumerisches Label
- **abbrv** - ähnlich wie plain mit Abkürzungen[10]

Um den Aufbau eines Styles besser zu verstehen, ist es hilfreich, benutzerdefinierte Funktionen in verschiedene Kategorien einzuteilen. Ein anderer Grund für diese Analyse ist es, Standardfunktionen von speziellen Funktionen zu unterscheiden, da generelle Funktionen als Spracherweiterung von BibTeX gesehen werden können:

- Eintragsfunktionen
- Formatierungsfunktionen (Bibliographie, Label, Sortierung)
- Standardfunktionen (Logik, Textfluss, Ausgabe)

Dabei kann nach unterschiedlichen analytischen Methoden vorgegangen werden:

Namensanalyse
Standard-Styles sind durch Einhaltung gewisser Namenskonventionen gut strukturiert. Prinzipiell gilt, Funktionsnamen, die aus einem Wort bestehen, sind Eintragstypen (zum Beispiel techreport, manual, inproceedings, phdthesis, article, proceedings, inbook, booklet, book, conference, incollection, unpublished, misc, mastersthesis). Ausgenommen davon sind generelle Funktionen (or, and, not, presort, output, sortify, emphasize).
Ausgehend von den Eintragsfeldern können spezielle Formatierungsfunktionen gefunden werden. Solche Funktionen setzen sich aus dem Wort format und dem Eintragsfeld[11], durch einen Punkt getrennt zusammen (zum Beispiel format.edition). In manchen Fällen wird der Plural verwendet, um zu zeigen, dass das Eintragsfeld in mehrere Teile zerlegt wird (zum Beispiel format authors).
Einige wenige Funktionen beinhalten auch den Eintragstyp im Namen. Solche Funktionen sind speziell für diesen Typ implementiert und setzen sich meistens aus einem weiteren Wort zusammen. Dabei ist darauf zu achten, dass auch Abkürzungen verwendet werden (zum Beispiel format.incoll.inproc.crossref). Diese Funktion übernimmt die Formatierung der Querverweise von Bibliographien des Typs inproceedings und incollection.

Inhaltliche Analyse
Da die Namensanalyse nicht sicher ist und nur eine Voreinteilung von Funktionen zulässt, müssen Funktionen einer inhaltlichen Analyse unterzogen werden; Dabei werden Funktionen auf Vorkommen von bestimmten Funktionen, Eintragsfeldern, Eintragstypen und

[10] Andere Namensformatierung, MACROS beinhalten mehr Abkürzungen.
[11] Definiert in den Eintragsfeldern durch die Anweisung ENTRY.

Variablen untersucht. Ein weiteres Merkmal ist das Vorkommen von Textkonstanten[12].

Hierarchische Analyse
Dabei werden die Abhängigkeiten der Funktionen untereinander analysiert. Um Funktionen inhaltlich analysieren zu können, muss die Menge aller Funktionen und Variablen bekannt sein. Entscheidend ist die erste und letzte Ebene der Aufrufhierarchie. So unterscheiden sich beispielweise generelle Funktionen von Funktionen für Eintragstypen dadurch, dass sie keine weitere Funktion aufrufen. Generell gilt für Eintragstypfunktionen, dass sie von keiner anderen Funktion aufgerufen werden. Das wird implizit durch die interne Funktion call.type$ durchgeführt. Ausnahme ist das Typenmapping, wobei ein Eintragstyp auf einen anderen vererbt wird (zum Beispiel FUNCTION conference oder inproceedings). In diesem Beispiel ruft die Funktion conference die Funktion inproceedings auf, da sich diese Typen in der Art der Formatierung nicht unterscheiden.

Vergleichende Analyse
Eine ausschließende Methode für die Klassifizierung ist der Vergleich auf Übereinstimmung der Funktionen in den vier Standards. Wenn der Vergleich der verschiedenen Styles eine inhaltliche Differenz ergibt, so kann davon ausgegangen werden, dass es sich nicht um eine Standard-Funktion handelt. Dabei werden besonders die Werte von Textkonstanten untersucht. In der Praxis kann die Klassifizierung nur in Kombination von mehreren Methoden erfolgen. Eine Liste aller Funktionen ergibt sich aus der Analyse aller Styles, da nicht jeder Style alle Funktionen beinhaltet. Abbildung *3.4* zeigt das Vorkommen einzelner Funktionen in den vier Standard-Styles.

3.6.1 Standardfunktionen

In diesem Kapitel werden Funktionen vorgestellt, die nicht direkt Sprachumfang von BibTeX sind, aber in den Standard-Styles Verwendung finden. Im Gegensatz zu anderen Funktionen, die speziell für einen bestimmten Style definiert wurden, können diese Funktionen wie eine Erweiterung von BibTeX (Bibliothek) gesehen werden.
Boolesche Funktionen
and (Konjunktion)
or (Disjunktion)
not (Negation)

Boolesche Funktionen können auch kombiniert werden. Bei Kombinationen solcher Funktionen ist darauf zu achten, dass durch die UPN-Notation von rechts nach links gelesen

[12] siehe auch vergleichende Analyse

NAME	PLAIN	ALPHA	ABBRV	UNSRT	NAME	PLAIN	ALPHA	ABBRV	UNSRT
and	x	x	x	x	init.state.consts	x	x	x	x
article	x	x	x	x	initialize.longest.label	x	x	x	x
author.editor.sort	x	x	x		inproceedings	x	x	x	x
author.organization.sort	x	x	x		longest.label.pass	x		x	x
author.sort	x	x	x		manual	x	x	x	x
begin.bib	x	x	x	x	mastersthesis	x	x	x	x
book	x	x	x	x	misc	x	x	x	x
booklet	x	x	x	x	multi.page.check	x	x	x	x
chop.word	x	x	x		n.dashify	x	x	x	x
conference	x	x	x	x	new.block	x	x	x	x
default.type	x	x	x	x	new.block.checka	x	x	x	x
editor.organization.sort	x	x	x		new.block.checkb	x	x	x	x
either.or.check	x	x	x	x	new.sentence	x	x	x	x
emphasize	x	x	x	x	new.sentence.checka	x	x	x	x
empty.misc.check	x	x	x	x	new.sentence.checkb	x	x	x	x
end.bib	x	x	x	x	not	x	x	x	x
field.or.null	x	x	x	x	or	x	x	x	x
fin.entry	x	x	x	x	output	x	x	x	x
format.article.crossref	x	x	x	x	output.bibitem	x	x	x	x
format.authors	x	x	x	x	output.check	x	x	x	x
format.book.crossref	x	x	x	x	output.nonnull	x	x	x	x
format.btitle	x	x	x	x	phdthesis	x	x	x	x
format.bvolume	x	x	x	x	presort	x	x	x	
format.chapter.pages	x	x	x	x	proceedings	x	x	x	x
format.crossref.editor	x	x	x	x	sort.format.names	x	x	x	
format.date	x	x	x	x	sort.format.title	x	x	x	
format.edition	x	x	x	x	sortify	x	x	x	
format.editors	x	x	x	x	techreport	x	x	x	x
format.in.ed.booktitle	x	x	x	x	tie.or.space.connect	x	x	x	x
format.incoll.inproc.crossref	x	x	x	x	unpublished	x	x	x	x
format.names	x	x	x	x	author.editor.key.label		x		
format.number.series	x	x	x	x	author.key.label		x		
format.pages	x	x	x	x	author.key.organization.label		x		
format.thesis.type	x	x	x	x	calc.label		x		
format.title	x	x	x	x	editor.key.organization.label		x		
format.tr.number	x	x	x	x	format.lab.names		x		
format.vol.num.pages	x	x	x	x	forward.pass		x		
inbook	x	x	x	x	initialize.et.al.char.used		x		
incollection	x	x	x	x	reverse.pass		x		

Abbildung 3.4: Vergleich zwischen Funktionen der Standard-Styles

werden muss. Komplexe Kombination kommen in Standard-Syles nicht vor, könnten aber über Literale abgebildet werden. Da diese Funktionen die Wahrheitswerte 0 oder 1 über den Stack an die nächste weitergeben, müssen sie in sequentieller Reihenfolge interpretiert werden.
Beispiel für die Funktion AND:

```
FUNCTION {and}
    { 'skip$
    { pop$ #0 }
  if$
}
```

Zwei boolesche Werte liegen an oberster Stelle am Stack. Die Funktion if$ testet den ersten Wert, ist dieser größer als 0 („Wahr“), bleibt nur der zweite Wert am Stack. Wenn dieser Wert positiv ist, ist die Konjunktion aufgegangen. War der erste Wert 0, schreibt die Funktion 0 („Falsch“) auf den Stack (die Konjunktion ist unabhängig vom zweiten Wert, nicht aufgegangen).

Beispiel für dir Funktion OR:

```
FUNCTION {or}
{   { pop$ #1 }
    'skip$
  if$
}
```

Wenn die Prüfung vom ersten Wert im Stack positiv ist, so steht unabhängig vom zweiten Wert 1 im Stack. Wenn die Prüfung vom ersten Wert negativ ist, verbleibt der zweite Wert. Die Disjunktion ist erfolgreich, wenn einer der beiden Werte wahr ist.

Beispiel für die Funktion NOT:

```
FUNCTION {not}
{   { #0 }
    { #1 }
  if$
}
```

Die Funktion NOT dreht die booleschen Werte einfach um. Aus 1 wird 0, aus 0 wird 1.

Initalisierungsfunktionen
Da bei der Deklaration von Variablen keine Wertezuweisung durchgeführt werden kann, geschieht dies mit eigenen Funktionen.
initialize.et.al.char.used[13]
Integer:
et.al.char.used = 0
Der boolesche Wert „Wahr“ wird gesetzt, wenn das Label einen etalchar beinhaltet.
Wenn der Wert 1 („Wahr“) ist, dann wird vor den bibliographischen Einträgen die Zeile

```
\newcommand{\etalchar}[1]{$^{#1}$}
```

ins *bbl*-File gesetzt.

initialize.longest.label
Die Funktion initialize.longest.label initialisiert Variablen, die zur Ermittlung des längsten Labels benötigt werden.
Strings:
longest.label = ""
last.sort.label = "0"
next.extra =""
Integer:
longest.label.width = 0
last.extra.num = 0

init.state.consts
Für die korrekte Satzsetzung werden Konstanten verwendet, die Auskunft über die aktuelle Position in einem Satz bzw. einem Absatz geben.
Es werden vier Konstanten für den Positionsstatus verwendet:
#0 'before.all :=
#1 'mid.sentence :=
#2 'after.sentence :=
#3 'after.block :=

Formatierungsfunktionen

format.names
Diese Funktion zählt zu den komplexesten Funktionen eines Styles. Sie iteriert mittels while$-Schleife über die Anzahl der vorkommenden Namen und ruft für jeden Namen

[13] nur im Style alpha

die Funktion format.name$ auf. Die Anzahl der Namen aus den Eintragsfeldern wird mit der internen Funktion num.names$ ermittelt. Die Liste der formatierten Namen wird konkatiniert an den Stack zurückgegeben.
Diese Funktion beinhaltet eine wichtige Textkonstante, nämlich die Formatmaske für die Namensformatierung. Sie unterscheidet sich für die Styles alpha, plain, unsrt: '{ff~}{vv~}{ll}{, jj}' von abbrv '{f.~}{vv~}{ll}{, jj}'.

n.dashify
Diese Funktion ersetzt einen einfachen Bindestrich durch einen doppelten. Mehrere aneinander folgende Bindestriche bleiben in ihrer Anzahl erhalten.

multi.page.check
Diese Funktion überprüft in einem Text, ob das Zeichen '-' oder '+' enthalten ist, wenn ja, dann wird 0 zurückgegeben.

field.or.null
Diese Funktion schreibt einen Leerstring, wenn der Stack leer ist.

chop.word
Diese Funktion schneidet einen Text auf die maximale Länge von 5000 Zeichen ab[14].

tie.or.space.connect
Diese Funktion entscheidet, ob eine Tilde oder ein Leerzeichen verwendet werden soll.

sortify
Diese Funktion formatiert einen Wert, bevor er zum Sortieren verwendet wird. Mit der internen Funktion purify$ [15] wird auf Kleinbuchstaben umgestellt.

emphasize
Diese Funktion formatiert einen Textwert kursiv (LaTeX Syntax).

3.6.2 Eintragsfunktionen

Eintragsfunktionen sind Einstiegspunkte für die Formatierung von bibliographischen Einträgen. Für jeden bibliographischen Typ steht eine eigene Funktion zu Verfügung. Der Name der Funktion entspricht dem Eintragstyp aus dem *bib*-File. Eintragsfunktionen werden direkt von der ITERATE-Anweisung mit der Funktion call.type$ aufgerufen.

[14] automatisch deklarierte globale Konstante global.max$
[15] siehe interne Funktionen

Eintragsfunktionen erkennt man daran, dass sie mit der Anweisung output.bibitem beginnen und mit fin.entry enden. Ein anderes Merkmal von Eintragsfunktionen ist, dass sie nicht von anderen Funktionen aufgerufen werden. Das heißt sie stehen immer an erster Stelle in der Aufrufhierarchie (atomar). Eintragsfunktionen unterscheiden sich in den vier Standard-Stytles nicht.

Typischer Aufbau einer Eintragsfunktion:
Betrachtet man den Aufbau von Eintragsfunktionen, so kann man sie in drei Kategorien einteilen:

1. Einfach
2. Komplex
3. Mit Querverweisen

Einfach
Zum Beispiel die Funktion: techreport

Listing 3.4: plain.bst Funktion techreport

```
813 FUNCTION {techreport}
814 { output.bibitem
815   format.authors "author" output.check
816   new.block
817   format.title "title" output.check
818   new.block
819   format.tr.number output.nonnull
820   institution "institution" output.check
821   address output
822   format.date "year" output.check
823   new.block
824   note output
825   fin.entry
826 }
```

In sequentieller Weise werden Eintragsfelder oder das Ergebnis von Formatierungsfunktionen in den Ausgabebuffer geschrieben. Der Eintrag beginnt mit der Funktion output.bibitem, die den Beginn eines Literatureintrages in einem *bbl*-File bildet[16]. Am Ende steht die Funktion fin.entry, die das Absatzende eines Eintrages formatiert. Im Normalfall wird nur ein Punkt gesetzt. Zwischen den beiden Funktionen befinden sich die Feldformatierungen. Entweder wird direkt der Eintrag geschrieben, wie zum Beispiel address, oder mit einer Formatierungsfunktion, zum Beispielformat.date 'year' output.check. Im

[16] schreibt \ bibitem[Label]{Schlüssel}

zweiten Fall wird zusätzlich die Textkonstante 'year' angehängt.
Weitere einfache Eintragsfunktionen:

- misc
- unpublished
- phdthesis
- mastersthesis
- booklet

Komplex
Komplexe Eintragsfunktionen haben eine oder mehrere logische Verzweigungen. Dabei wird in den meisten Fällen die Formatierung vom Vorhandensein bestimmter Eintragsfelder beeinflusst.
So wird zum Beispiel in der Funktion proceedings, das Feld organization ausgegeben, wenn das Feld editor leer ist:

Listing 3.5: plain.bst Funktion proceedings

```
801       editor empty$
802          'skip$
803          { organization output }
804        if$
```

Logische Verzweigungen können auch mehrfach verschachtelt sein, wie zum Beispiel in der Funktion manual.

Listing 3.6: plain.bst Funktion manual

```
700 FUNCTION {manual}
701 { output.bibitem
702   author empty$
703     { organization empty$
704         'skip$
705         { organization output.nonnull
706           address output
707         }
708       if$
709     }
710     { format.authors output.nonnull }
711   if$
```

Die äußere Prüfung (Zeile 702,711) bezieht sich auf das Vorhandensein des Feldes author. Wenn der Wert für dieses Feld nicht vorhanden (empty$) ist, wird die innere Prüfung

(Zeile 703,708) durchgeführt.

Mit Querverweisen
Für folgende Eintragstypen gibt es Verzweigungen für die Formatierungen von Querverweisen:

article
book
inbook
incollection
inproceedings (conference)

Dabei wird das interne Eintragsfeld crossref auf vorhandenen Inhalt überprüft. Je nach Vorhandensein werden zwei unterschiedliche Anweisungsblöcke ausgeführt.
Zum Beispiel Funktion inproceedings:

Listing 3.7: plain.bst Funktion inproceedings

```
664 FUNCTION {inproceedings}
665 { output.bibitem
666   format.authors "author" output.check
667   new.block
668   format.title "title" output.check
669   new.block
670   crossref missing$
671     { format.in.ed.booktitle "booktitle" output.check
672       format.bvolume output
673       format.number.series output
674       format.pages output
675       address empty$
676         { organization publisher new.sentence.checkb
677           organization output
678           publisher output
679           format.date "year" output.check
680         }
681         { address output.nonnull
682           format.date "year" output.check
683           new.sentence
684           organization output
685           publisher output
686         }
687       if$
688     }
689     { format.incoll.inproc.crossref output.nonnull
690       format.pages output
```

```
691       }
692     if$
693     new.block
694     note output
695     fin.entry
696   }
```

Wenn das Feld crossref leer ist, wird der Anweisungsblock (Zeile 671-686) ausgeführt, anderenfalls der Anweisungsblock (Zeile 689-691). Im Fall von Crossreferenzen wird eine spezielle Formatierungsfunktion aufgerufen, im oben beschriebenen Fall die Funktion format.incoll.inproc.crossref.

3.6.3 Formatierungsfunktionen

Formatierung der Literatureinträge
format.editors
format.authors
format.edition
format.title
format.pages
format.bvolume
format.btitle
format.date
format.number.series
format.chapter.pages
format.vol.num.pages
format.thesis.type
format.tr.number
format.in.ed.booktitle

Formatierung von Querverweisen
Dieser Funktionstyp greift direkt auf den Eintrag zu, der referenziert wird. Diese Art von Funktion erkennt man am Verweis zur Referenz[17]:

```
" \cite{" * crossref * "}" *
```

format.book.crossref
format.article.crossref

[17] Dieser Eintrag ist typischerweise in der letzten Zeile der Funktion zu finden.

format.crossref.editor
format.incoll.inproc.crossref

Formatierung der Labels
Diese Funktionen sind speziell für die Formatierung der Labels implementiert.

calc.label
Diese Funktion dient als Einstiegspunkt für die Berechnung des Labels. Das formatierte Ergebnis wird der Variable label zugewiesen. In Folge wird diese Variable wie ein Eintragsfeld verarbeitet. Diese Funktion wird direkt in der Funktion presort aufgerufen, die durch die Anweisung ITERATE ausgeführt wird.
Folgende Funktionen werden von der Funktion calc.label direkt aufgerufen:
author.editor.key.label
author.key.label
author.key.organization.label
editor.key.organization.label

Folgende Funktion wird von den oben genannten Funktionen direkt aufgerufen: format.lab.names

Spezielle Funktionen für alphanumerische Labels (alpha.bst)

forward.pass
Diese Funktion hängt an das Label ein 'a' an, wenn es bereits existiert. Diese Funktion wird mit der Anweisung ITERATE aufgerufen.

reverse.pass
Diese Funktion hängt an das Label ein 'b' an, wenn es bereits existiert. Diese Funktion wird mit der Anweisung REVERSE aufgerufen.

3.6.4 Sortierfunktion

Feldspezifische Sortierfunktionen
Ähnlich den Formatierungsfunktionen für feldspezifische Teile eines bibliographischen Eintrages gibt es Funktionen, die Teile des Sortierschlüssels formatieren:
sort.format.names
sort.format.title

Spezielle Sortierfunktionen
presort

Diese Funktion ist Einstiegspunkt für die Sortierung und Labelberechnung. Neben der Funktion calc.label, führt sie sort.label aus.

Der Sortierschlüssel ergibt sich in der Regel aus der Zusammensetzung von:

- Label
- mehreren Leerzeichen
- dem Sortierlabel (sort.label)
- vom Eintragstyp abhängigem Text

Diese eintragstypischen Formatierungen werden mit speziellen Formatierungsfunktionen durchgeführt (Formatierung des Sortierschlüssels). Das Ergebnis dieser Zusammensetzung ist ein Text, der als Sortierschlüssel verwendet wird. Dieser Text wird in der internen Variable sort.key$ abgespeichert. Die Anweisung SORT nimmt folglich die Sortierung vor.

Formatierung des Sortierschlüssels
Funktionen für die Formatierung des Sortierschlüssels:

editor.organization.sort
author.editor.sort
author.sort
author.organization.sort

3.6.5 Satzstellungsfunktionen

BibTeX schreibt während der Verarbeitung in die globale Variable output.state, die den aktuellen Zustand der Satzstellung wiedergibt. Es gibt vier Konstanten, die den Satzstellungsstatus eines Eintrages während eines Formatierungsdurchlaufs beschreiben:

- 0 = before.all (erste Stelle, genau nach '\bibitem')
- 1 = mid.sentence (der Eintrag steht in der Mitte eines Satzes, ein Komma wird benötigt, wenn mehrere Einträge vorkommen)
- 2 = after.sentence (nach einem Satz, ein Punkt wird benötigt)
- 3 = after.block (nach einem Block oder einem Beistrich, ein Punkt und '\newblock' werden angehängt)

Anmerkung: Standard-Styles verwenden den Ausgabestatus after.sentence nicht!

new.block
Diese Funktion setzt den Status after.block, wenn der aktuelle Status nicht before.all ist.

new.sentence
Diese Funktion setzt den Status after.sentence, wenn der aktuelle Status nicht before.all oder after.block ist.

Übergeordnete Funktionen der (new)-Funktionen sind:
new.block.checka
new.block.checkb
new.sentence.checkb
new.sentence.checka

(a)-Funktionen rufen die jeweilige Unterfunktion auf, wenn das erste Argument am Stack nicht leer ist. (b)-Funktionen beziehen sich auf die letzten beiden Argumente im Stack.

3.6.6 Schreibende Funktionen

Output *bbl-File*
Eintragsfunktionen schreiben nicht direkt in den Ausgabepuffer, dafür werden eigene Output-Funktionen verwendet. Diese Funktionen kapseln das Schreiben in den Ausgabepuffer:
output.nonnull
output
output.check

output.nonnull
ist die Hauptfunktion, die mit der internen Funktion write$ den Inhalt vom Stack in das *bbl*-File schreibt. Die Funktionen output und output.check rufen die Funktion output.nonnull auf. In manchen Fällen wird die Funktion output.nonnull auch direkt verwendet.

3.6.7 output

Die Funktion output überprüft, ob der Stack leer ist, wenn nicht, wird die Funktion output.nonnull aufgerufen. Diese Funktion arbeitet mit dem letzten Wert im Stack als Übergabeparameter.

3.6.8 output.check

Diese Funktion arbeitet mit den zwei obersten Stack-Elementen. Der letzte wird als Parameter t vom Stack gelesen. Dieser beinhaltet einen String, der für mögliche Warnung verwendet wird. Der verbleibende Wert im Stack wird geprüft, ob er leer ist, wenn ja, wird eine Warnung ins Log-File (*bgl*) geschrieben, wenn nicht, wird der Wert vom Stack der Funktion output.nonnull als Parameter übergeben.

3.6.9 Funktionsweise „output.nonnull“

Je nach dem Ausgabestatus werden bestimmte Trennzeichen geschrieben. Der Ausgabestatus wird in BibTeX in der globalen Variable output.state gehalten. Genau genommen ist das eine integer-Variable, die im Standardfall die Werte 0 bis 3 annimmt. Dafür werden vier globale Konstanten initialisiert, die der Ausgabe mitteilen, welcher Separator, Punkt oder Beistrich anzuwenden ist.
Andere Funktionen schreiben direkt in den Output-Puffer. Diese Funktionen werden von den Eintragsfunktionen verwendet, um den Start- und Endteil einer Bibliographie zu schreiben:
output.bibitem
fin.entry

Schließlich gibt es zwei Funktionen, die den Start- und den Endblock des gesamten *bbl*-Files schreiben:
begin.bib
Diese Funktion schreibt die Zeile, wenn ‚et.al’-Zeichen verwendet wurden:

```
'\newcommand{\etalchar}[1]{$^{#1}$}'
```

Im nächsten Schritt wird die interne Funktion preambel$ aufgerufen, die alle Präambeleinträge aus dem textslbib-File aneinandergereiht schreibt. Anschließend wird das Kommando

```
'\begin{thebibliography}{'  longest.label  * '}'
```

geschrieben, wobei die Variable longest.label in einem Schritt davor ermittelt werden muss.

end.bib
Diese Funktion schreibt den Text

```
'\end{thebibliography}'
```

in den Output-Puffer.

Warnings (*blg*-File)
empty.misc.check
Diese Funktion überprüft, ob gewisse Eintragsfelder vorhanden sind. Wenn diese fehlen, wird eine Warnung geschrieben. Dabei wird überprüft, ob alle notwendigen Eintragsfelder vorhanden sind, um den Sortierschlüssel korrekt zu berechnen. Die Liste der Eintragsfelder kann zwischen den unterschiedlichen Styles variieren. Für plain, alpha, abbrv gilt: Wenn der Schlüsselwert (key) nicht leer ist, muss eines der Felder author, title, howpublished, month, year, note vorhanden sein, anderenfalls wird eine Warnung geschrieben. Für unsrt gilt: Es muss eines der Felder author, title, howpublished, month, year, note vorhanden sein.

either.or.check
Diese Funktion überprüft, ob eines der beiden übergeben Felder vorhanden ist, wenn nicht, wird eine Warnung geschrieben.

3.7 Architekturüberlegungen Neuentwicklung BibTeX

Eine der wichtigsten Designentscheidungen von BibTeX ist, dass Publikationstypen und Datenfeldtypen nicht hartcodiert im Programm sind, sondern das Programm nur die Syntax kennt und die Namen im Style-File definiert werden [Bee04]. BibTeX-*bst* ist im eigentlichen Sinn eine Programmiersprache und kann, wenn man den vollen Funktionsumfang erhalten will, nur durch eine Programmiersprache abgelöst werden. Wenn man BibTeX in eine andere Sprache übersetzt, verschiebt sich das Problem nur. Vor allem wenn der Ablauf von bst direkt übersetzt wird, übernimmt man viele komplizierte Operationen, die ja das eigentliche Problem von BibTeX-*bst* sind. Bei Überlegungen zur Reimplementierung von BibTeX muss zwischen der Datenhaltung und der Formatierung der Literatureinträge unterschieden werden.

3.7.1 Datenhaltung

Betrachtet man das BibTeX-Format, erkennt man, dass es für die Anforderungen nicht leistungsfähig genug ist. Es sollte daher nur als Import-Exportformat Verwendung finden. Die Verwaltung unterschiedlicher Dateien ist sehr umständlich, das Zusammenführen von Dateien aus unterschiedlichen Quellen praktisch unmöglich. BibTeX-Files unterstützen vernetztes Arbeiten nur bedingt, da der verteilte Zugriff auf Textdateien praktisch unmöglich ist. Such- und Sortierfunktionen gibt es nur in sehr eingeschränkter Art und Weise.
Eine Eigenschaft von relationalen Datenbanken ist es, Daten so zu halten, dass gewisse Normen (constraints) eingehalten werden müssen. Ein wesentlicher Vorteil von relational modellierten Datenmodellen ist das Vermeiden von Dateninkonsistenzen (Widersprüchlichkeit von Daten). BibTeX prüft in keiner Weise doppelte Einträge, was besonders für die Schlüsselwerte problematisch ist.
Alle oben angeführten Argumente sprechen für die Datenhaltung in einer relationalen Datenbank (RDBMS). Um das neue System offen zu gestalten, kann nur eine Datenbank in Frage kommen, die ANSI-SQL unterstützt. Von den spezifischen Anforderungen des Benutzers wird es abhängen, ob eine Server-Datenbank oder eine einfache File-Datenbank Verwendung finden soll. SQL bietet eine etablierte und genormte Methode, einfach auf strukturierte Daten zuzugreifen. SQL bietet nicht nur Datenbankfunktionalität wie Abfragen, Prüfen, Vergleichen und Sortieren von Datenbeständen, sondern eignet sich auch für die Bearbeitung von Text. Durch die Rolle als Quasi-Standard ist SQL von großer Bedeutung, da eine weitgehende Unabhängigkeit von der benutzten Software erzielt werden kann.
Eine andere Möglichkeit wäre die Transformation der Datenfiles in XML-Dateien. Das würde aber nur die wenigsten der oben angeführten Probleme lösen. XML erhöht den Grad der Strukturiertheit, das Problem konkurrierender Zugriffe wird dabei nicht gelöst. Gegen XML spricht auch dessen schlechte Performance bei großen Datenbeständen.

3.7.2 Formatierung

Um die Anforderung der Wiederverwendbarkeit von bst Styles zu gewährleisten, muss das neue Programm in der Lage sein, *bst*-Dateien zu übersetzen oder zu interpretieren. Ein neuer Interpreter könnte den Zugriff auf Datenbanken optimieren und eines der Hauptprobleme von BibTeX, nämlich die fehlende Modularität lösen. Ein Interpreter, der bestehende Style-Files verarbeitet, würde aber den oben angeführten Anforderungen nicht genügen, da viele Probleme, wie umständliche Programmierung, direkt im Style-File liegen.
Das Kernproblem liegt darin, dass BibTeX eine Programmiersprache ist und jede Art

der Textmanipulation offen lässt. Das heißt eine Lösung, die ausschließlich auf vorgegebene Konfigurationseinstellungen aufbaut, würde den theoretischen Leistungsumfang von BibTeX entscheidend einschränken.
Wenn der *bst*-Programmcode 1:1 in eine andere Sprache übersetzt werden würde, verschiebt sich das Problem nur. Der komplexe Programmablauf, der sich teilweise aus der Stack-Sprache ergibt, müsste nachgebildet werden.
Andere Projekte beschäftigen sich damit, Compiler zu verwenden, die Binär-Code[18] re-compilieren. Das Ergebnis wäre Programmcode in einer höheren Programmiersprache. Der Benutzer müsste Kenntnisse der Programmiersprache besitzen, um bestehende Styles zu verstehen und zu ändern oder neue Styles zu entwickeln. Da BibTeX sehr spezifische Funktionen zu Verfügung stellt, wäre das kaum ein Vorteil.
Eine Hauptaufgabe von BibTeX ist das Abfragen von Datenbanken. Dafür eignen sich Datenbanksprachen wie SQL besonders gut. Die Syntax von SQL ist relativ einfach aufgebaut und semantisch an die englische Umgangssprache angelehnt.

[18] Binär-Code ist ähnlich wie die Stack-Sprache aufgebaut.

4 Implementierung

Dieses Kapitel beschreibt die Implementierung von BibSQl. BibSQL ist eine Neuimplementierung von BibTeX auf SQL-Basis. Als technische Grundlage für die Entwicklung von BibSQL dient die Machbarkeitsstudie und die Funktionsanalyse von BibTeX.

4.1 Funktionale Anforderung

Die Beschreibung der funktionalen Anforderungen basiert auf die Analyse der bestehenden Funktionalitäten und Schwächen von BibTeX.

4.1.1 Einbindung in die LaTeX-„thebibliography“-Umgebung

Der Workflow LaTeX-BibTeX muss, wie in Abbildung 4.1 dargestellt wird, erhalten bleiben. Das *aux*-File muß in gewohnter Weise gelesen und verarbeitet werden. Der Output ist ein *bbl*-File, kompatibel mit der LaTeX-„thebibliography“-Umgebung. Bestehende

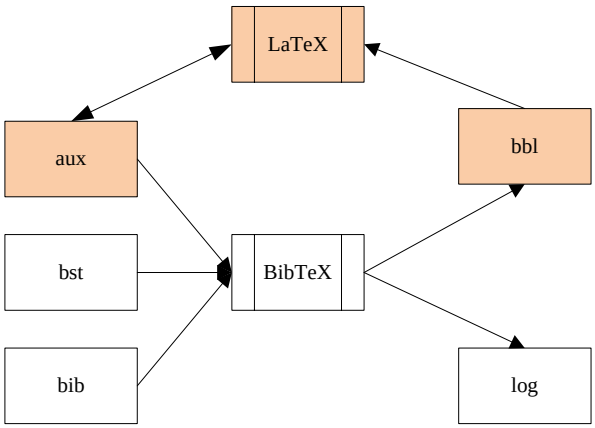

Abbildung 4.1: Datenfluss LaTeX-BibTeX

bst-Files müssen wiederverwendbar sein. Entweder werden sie interpretiert (Kompatibilitätsmodus) oder in eine andere Sprache übersetzt.
Bib-Dateien müssen indirekt oder direkt lesbar sein. Dabei müssen Elemente wie Macros und Präambeleinträge unterstützt werden.

4.1.2 Strukturierte Datenhaltung

Die Datenhaltung wird in strukturierter Weise umgesetzt. Dabei kommt ein relationales RDBMS zum Einsatz. Das Datenmodell muss so gestaltet werden, dass Eintragstypen und Feldtype jederzeit erweitert werden können.

4.1.3 bst-Standards

Die Funktionalität von Standard-*bst*-Dateien muss vorab mitgeliefert werden. Das heißt die Styles alpha, plain, unsrt und abbrv werden in der Basisversion standardmäßig unterstützt.

4.1.4 Verwendung von Standartsoftware

Es soll ein modernes portables System entstehen, das auch Multibyte-Encoding unterstützt. Die eingesetzten Programmiersprachen und Tools entsprechen dem Stand der Zeit. Die Programmiersprache sollte den objektorientierten Ansatz unterstützen, möglichst Plattform unabhängig und leicht erlernbar sein. Das RDBMS muss ANSI SQL unterstützen. Alle eingesetzten Mittel sollen Open Source oder zumindest frei zugänglich sein.

4.1.5 Modulare Bauweise

Das Programm wird modular aufgebaut, das heißt bestehende Funktionen werden in Bibliotheken abgelegt und können von späteren Weiterentwicklungen eingebunden werden.

4.1.6 Benutzerfreundliche Style-Entwicklung

Die Style-Entwicklung darf nicht komplex sein, sie sollte von einem programmierunerfahrenen Benutzer erlernbar sein. Das System muss genauso offen wie BibTeX sein.

4.1.7 Mehrsprachenfähigkeit

Das System muss Mehrsprachenfähigkeit unterstützen, oder so modular aufgebaut sein, dass solche Features jederzeit eingebaut werden können.

4.2 Machbarkeitsstudie Übersetzung BibTeX-bst nach SQL

Betrachtet man BibTeX-Anweisungen und Eintragsfunktionen im Detail, kann man gewisse Ähnlichkeiten zu SQL feststellen. Die Anweisung ITERATE selektiert in einer Schleife bestimmte Einträge aus der bibliographischen Datenbank. Dabei wird auf definierte Felder der Datenbank zugegriffen. Wenn zuvor die Anweisung SORT durchgeführt wird, wird nach einem Schlüssel sortiert abgefragt. Das entspricht in etwa der SQL-Anweisung:

Listing 4.1: Vergleich einer SQL-Anweisung mit der ITERATE-Anweisung

```
SELECT Entryfield||
       Entryfield
  FROM bib.File
 WHERE cite in aux
 ORDER BY sort.key$
```

Ein BibTeX-Anweisungsblock besteht in der Regel aus zwei Ausführungsanweisungen, mindestens zwei Sortieranweisungen und drei Iterationsanweisungen:

Listing 4.2: BibTeX-Anweisungsblock

```
EXECUTE      -> begin.bib
   ITERATE   -> calc.label
   ITERATE   -> calc.sort
     SORT
   ITERATE   -> call.type
EXECUTE      -> end.bib
```

In BibTeX werden drei unabhängige Iterationen durchgeführt, eine zum Formatieren des Labels, eine zum Formatieren des Sortierschlüssels und abschließend die Formatierung der bibliographischen Einträge. Die dritte und letzte Iteration ist auch zuständig für das Schreiben in das *bbl*-File. Genau genommen schreiben die Funktionen sort und label nur in eine Eintragsvariable, die im letzten Schritt verarbeitet wird.

Im Gegensatz dazu wird in SQL nur ein SQL-Cursor (Select Statement) durchgeführt. In einer Unterabfrage werden die Spalten BIBITEM, LABEL und SORTKEY in einem Schritt abgefragt.

Listing 4.3: Einfache SQL-Anweisung für einen bibliographischen Eintrag

```
SELECT "\bibitem"||"["LABEL"]{"cite"}" || BIBITEM|| BIBITEM
  FROM(
   SELECT Entryfield||
        Entryfield      AS BIBITEM
        Entryfield||
        Entryfield      AS LABEL
        Entryfield||
        Entryfield      AS SORTKEY
   FROM bib.File
  WHERE cite in aux
      ) OUTPUT
ORDER BY SORTKEY
```

Der Vorteil gegenüber BibTeX ist, dass SQL nur einmal den Datenbestand abfragen und sortieren muss. Um die Zugriffsoptimierung kümmert sich das Datenbanksystem intern.

BibTeX formatiert die verschiedenen Eintragstypen unterschiedlich. Dabei wird durch die Funktion call.type$ die dementsprechende *bst*-Funktion aufgerufen. Diese Logik kann in ANSI-SQL mit der CASE WHEN-Syntax abgebildet werden. Dabei wird für jede Eintragstypenfunktion ein CASE WHEN-Block erzeugt. Die Default-Funktion verzweigt im ELSE-Teil der Anweisung und ruft die Funktion misc auf.

Listing 4.4: SQL-Anweisung für unterschiedliche Eintragstypen

```
SELECT
  CASE
   WHEN ITEMTYPE = 'article' THEN
     function{article}
   WHEN ITEMTYPE = 'booklet' THEN
     function{booklet}
   WHEN ITEMTYPE = 'conference' THEN
     function{conference}
   WHEN ITEMTYPE = 'inbook' THEN
     function{inbook}
   WHEN ITEMTYPE = '.....' THEN
     function{.....}
   ELSE
     function{misc}
   END CASE
```

Für Sortier- und Labelformatierungen werden in *bst* solche Strukturen nicht verwendet. Typenspezifische Unterscheidungen werden innerhalb der aufgerufenen Funktion mit logischen Verzweigungen abgebildet.
Der Nachteil dieser Variante ist, dass während des Übersetzens bekannt sein muss, welche Funktion eine Eintragsfunktion ist und welche nicht. Im Gegensatz zu Eintragsfeldern werden Eintragstypen im *bst*-File nicht deklariert. Im Prinzip sind Eintragstypen gewöhnliche Formatierungsfunktionen, durch eindeutige Merkmale können sie aber von Formatierungsfunktionen unterschieden werden[1].

4.2.1 Übersetzung der Eintrags- und Formatfunktionen

Im nächsten Schritt müssen Eintragsfunktionen und folglich auch Formatierungsfunktionen in die Sprache SQL übersetzt werden. Da Eintragsfelder in der Regel nicht einfach aneinander gereiht werden, sondern mit Funktionen formatiert werden, müssen diese auch nach SQL übersetzbar sein. Betrachtet man eine BibTeX-Funktion, erkennt man, dass die UPN-Sprache viele Ähnlichkeiten mit SQL aufweist.
Variablen (Eintragsfelder, interne Variablen), Funktionen (Formatierungsfunktionen, interne Funktionen, *bst*-Funktionen) und Textkonstanten werden nacheinander auf den Stack gelegt und in den Output-Puffer geschrieben.

Listing 4.5: *bst*-Funktion mastersthesis

```
 FUNCTION {mastersthesis}
{ output.bibitem
  format.authors "author" output.check
  new.block
  format.title "title" output.check
  new.block
  "Master's thesis" format.thesis.type output.nonnull
  school "school" output.check
  address output
  format.date "year" output.check
  new.block
  note output
  fin.entry
}
```

Bereinigt um die Output-Funktionen, könnte eine übersetzte Funktion folgendes Aussehen haben:

[1] zu erkennen am Muster output.bibitem und fin.entry

Listing 4.6: SQL-Eintragsfunktion mastersthesis

```
CASE WHEN entrytype = ‘mastersthesis’
THEN
SELECT    output.bibitem      Funktion
       ||format.authors       Funktion
       ||"author"             Textkonstante
       ||format.title         Funktion
       ||"title"              Textkonstante
       ||"Master’s thesis"    Textkonstante
       ||format.thesis.type   Funktion
       ||school               Eintragsfeld
       ||"school"             Textkonstante
       ||address              Eintragsfeld
       ||format.date          Funktion
       ||"year"               Textkonstante
       ||note                 Eintragsfeld
       ||fin.entry            Funktion
```

Das heißt die Übersetzung erfolgt in einem mehrstufigen Prozess. Ausgangspunkt ist, dass alle internen BibTeX-Funktionen und -Variablen bekannt sind. Neben den internen Funktionen und Variablen werden auch die benutzerdefinierten Standard-*bst*-Funktionen in SQL übersetzt und in ein Repository geschrieben. Anschließend müssen alle benutzerdefinierten Variablen (hauptsächlich Eintragsfelder) und Funktionen geparst werden und die entsprechenden Ersetzungen vorgenommen werden. Für den Aufbau des SQL-Cursors ist es notwendig, Eintragsfunktionen von Formatierungs-, Sortier- und Labelfunktionen zu unterscheiden.
Die Struktur der *bst*-Funktionen bleibt aber erhalten. Eine Funktion, die in einer Funktion aufgerufen wird, steht für einen formatierten Text, der in den Stack geschrieben wird. Da ANSI SQL keine Funktionen kennt, muss das finale SQL-Statement Funktionsnamen mit den eigentlichen Funktionen ersetzen. Dieses Statement wird zur Laufzeit gebildet. Für den Benutzer bleibt der Code modular, weil jede Funktion für sich im Repository steht.

Finales SQL für die Eintragsfunktion mastersthesis (Listing 4.7):

Listing 4.7: SQL-Anweisungsblock aus SQL_alpha.sql

```
WHEN ITEMTYPE = ’mastersthesis’ THEN
  ’\n’          --fieldformat        : output_bibitem
 ||’\bibitem[’
 ||label
 ||’]{’
 ||CITEKEY
 ||’}’
 ||’\n’
 ||’’
 ||’%{before_all}’
```

```
||'%{output_state}'
||CASE WHEN  "author" is NULL   THEN          --fieldformat        : format_authors
      ''
  ELSE
      "author_FORMAT_NAMES"
  END
||'%{output} '
||'%{newblock}'
||CASE WHEN  "title" is NULL   THEN          --fieldformat        : format_title
      ''
  ELSE
      INITCAP("title")
  END
||'%{output} '
||'%{newblock}'
||'Master''s thesis'
||CASE WHEN  "type" is NULL   THEN          --fieldformat        : format_thesis_type
      NULL
  ELSE
      INITCAP("type")
  END
||'%{output} '
||"school"
||'%{output} '
||"address"
||'%{output} '
||CASE WHEN  "year" is NULL   THEN          --fieldformat        : format_date
      CASE WHEN  "month" is NULL   THEN
          ''
      ELSE
          '%{warning$'
||        'there''s a month but no year in '
||        CITEKEY
||        '}'
||        "month"
      END
  ELSE
      CASE WHEN  "month" is NULL   THEN
          "year"
      ELSE
          "month"
||        ' '
||        "year"
      END
  END
||'%{output} '
||'%{newblock}'
||"note"
||'%{output} '
||'%{add_period}'          --fieldformat        : fin_entry
||'\n'
```

Eintragsfelder einlesen

Das Parsen der Eintragsfelder aus der *bst*-Datei ist nicht nur für den Aufbau von Funktionen wichtig, sondern auch für die Generierung des FROM-Teils des SQL-Statements. Da BibTeX bezüglich der Eintragsfelder ein generisches Datenmodell unterstützt, das heißt Eintragsfelder werden nicht fix durch die Datenstruktur vorgegeben, ist die Menge der abzufragenden Felder für den Aufbau des Statements notwendig.

Listing 4.8: *bst*-Entries

```
        ENTRY
{  EntryVar1
   EntryVar2
   EntryVar3
}
```

BibTeX verwenden logische Kontrollstrukturen für Programmverzweigungen. Im folgenden Beispiel (Teil der Funktion format.authors) wird abgefragt, ob das Feld author einen Wert hat. Wenn nicht, wird ein Leerzeichen auf den Stack geschrieben, sonst wird die Funktion author.format.names aufgerufen.

Listing 4.9: Beispiel einer logischen Verzweigung

```
{ author empty$
    { "" }
    { author format.names }
  if$
}
```

Logische Programmverzweigungen werden in BibTeX mit der internen Funktion if$ abgebildet. Da es in SQL kein 'IF' gibt, müssen solche Strukturen mit der ANSI-SQL Anweisung CASE WHEN übersetzt werden. Dabei ist auf die umgekehrte Notation zu achten.
Beispiel einer übersetzten SQL-Funktion:

Listing 4.10: Beispiel einer logischen Verzweigung nach SQL übersetzt

```
  CASE WHEN  "author" is NULL   THEN
      ''
  ELSE
      "author_FORMAT_NAMES"
  END
```

Die Schlüsselwörter and und or sind in BibTeX programmierte Funktionen. Sie gehören zu den Funktionen, die als Standard angesehen werden können und wie normale interne Funktionen verwendet werden. Wenn man den umständlichen Stack-Mechanismus der hinter and und or liegt in SQL nachprogrammieren würde, wäre das Ergebnis nur unnötig komplex und schwer lesbar.
BibTeX verwendet das Symbol „*“ zum Zusammenfügen von Textelementen. In ANSI-SQL entspricht das einem Doppelpipe „||“. Beim Übersetzen ist das Zeichen zu ersetzen und wegen der UPN-Notation die Reihenfolge auszutauschen.

4.2.2 Übersetzung der internen BibTeX-Funktionen

Interne BibTeX-Funktionen werden manuell in SQL übersetzt. Dabei kann einerseits auf Literatur zurückgegriffen werden, andererseits muss das exakte Verhalten empirisch ermittelt werden. Einige Funktionen können auch einfach weggelassen werden, weil sie in SQL nicht notwendig sind. Andere wiederum können durch bestehende ersetzt werden, da sie in SQL dieselbe Bedeutung haben[2].
Argumente werden vom Stack in der Reihenfolge von links nach rechts übernommen.

Listing 4.11: Beispiele für Übersetzung *bst* in SQL mit Parameterübergabe

```
v 'l' change.case$      ->    LOWER(v1)
v 'u' change.case$      ->    UPPER(v1)
newline$                ->    '\\n'
v 't' change.case$      ->    INITCAP(v1)
v num.names$            ->    v1_NUM_NAMES
v v v substring$        ->    substr(v1,v2, v3)
```

4.2.3 Übersetzung der Standard-*bst*-Funktionen

Standard-*bst*-Funktionen, die per Definition nach SQL übersetzt werden:

Listing 4.12: Nach SQL übersetzte *bst*-Funktionen

```
mid.sentence            -> '%{mid_sentence}'
v emphasize             -> '{\em}'|| v1 ||'\/}'
after.sentence          -> '%{after_sentence}'
v tie.or.space.connect -> CASE WHEN LENGTH(v1)<3 THEN '~' ELSE ' ' END || v1
v v v chop.word         -> CASE WHEN SUBSTR(v3,1,v2) = v1 THEN SUBSTR(v3,v2+1) ELSE v3
  END
v n.dashify             -> REPLACE(REPLACE(REPLACE (REPLACE(REPLACE(v1,'--','"xxx"'),'"
  xxx"-','"xxx""yy"'),'-','--'),'"xxx"','--'),'"yy"','-')
output.state            -> '%{output_state}'
v sort.format.title     -> REPLACE(REPLACE(REPLACE(v1,'A '),'An '),'The ')
v multi.page.check      -> TO_CHAR(LENGTH(v1)-LENGTH(REPLACE(REPLACE(v1,'+',''),'-',''))
  ))>0
```

Funktionen, die nicht direkt in SQL übersetzt werden, da sie vom BibSQL-Compiler indirekt übersetzt werden:

```
and
forward.pass
initialize.et.al.char.used
new.block.checkb
not
```

[2]Bei der Übersetzung wurde auf die strenge Einhaltung der ANSI-SQL Syntax Rücksicht genommen.

```
new.block
or
output
format.tr.number
output.check
calc.label
output.nonnull
new.sentence.checkb
presort
format.vol.num.pages
new.block.checka
new.sentence
reverse.pass
new.sentence.checka
either.or.check
```

Funktionen, die in SQL nicht benötigt werden und somit ignoriert werden:

```
pop\$
write\$
default.type
```

Funktionen, die ausschließlich in BibTeX Bedeutung haben und somit ignoriert werden:

```
field.or.null
init.state.consts
sortify
longest.label.pass
empty.misc.check
initialize.longest.label
```

4.2.4 Output Funktionen

Die Funktionen output und output.check rufen immer die Funktion output.nonnull auf. Diese Funktion ist schwierig in SQL[3] darzustellen, da mit globalen Variablen und Konstanten gearbeitet wird. Die Funktion output.nonnull benötigt für die korrekte Darstellung den Output-Status der aktuellen Satzstellung. Dieser wiederum ist vom Vorhandensein bestimmter Werte und statussetzenden Funktionen abhängig. Das heißt eine Spalte im Statement würde sich auf den Zustand einer oder mehrerer anderen Spalten beziehen. Ein mehrfach verschachteltes case when wäre notwendig, und in Folge die Komplexität relativ hoch.

[3] Es ist nicht unmöglich, aber komplex.

Für die Ausgabe in SQL ist es nicht relevant, welche dieser Output-Funktionen verwendet werden, da sie alle in der Funktion output.nonnull resultieren. In SQL ist die Unterscheidung NULL, empty oder ein Leerstring für die Ausgabe nicht von Bedeutung. Warnungen werden nicht ausgegeben.
statussetzende Funktionen direkt nach SQL zu übersetzen macht in diesem Zusammenhang keinen Sinn. Funktionen wie new.sentence, setzen den Ausgabestatus auf after.sentence, wenn der aktuelle Status nicht after.block oder bevor.all ist. Alle statussetzenden Funktionen werden in einem ersten Schritt mit einem Substitut in das Zwischenergebnis geschrieben. Output-Funktionen werden vereinheitlicht und ebenfalls mit einem Substitut ersetzt. Substitute müssen durch eine spezielle Syntax vom normalen Text unterscheidbar sein; Zum Beispiel %{function}. Im zweiten Schritt wird der Text durch eine Prozedur richtig gestellt.

Listing 4.13: Beispiel für übersetzte Funktionen

```
before.all        -> '%{before_all}'
after.block       -> '%{after_block}'
v v text.prefix$ -> '%{text_prefix, '|| v1 ||', '|| v2 ||'}'
add.period$       -> '%{add_period}'
```

Listing 4.14: Beispiel für das Zwischenergebnis

```
bibitem[Lamp86]{lamp86}
%{before_all}%{output_state}Leslie ~Lamport %{output}
%{newblock}{\em}{\LaTeX \rm:} {A} Document Preparation System\/}
%{output} %{output} %{newblock}%{output} %{new_sentence}Addison-Wesley
%{output} %{output} %{output} 1986 %{output} %{newblock}%{output}
%{add_period}
```

Listing 4.15 zeigt eine mögliche Implementierung einer Output-Funktion in Python.

Listing 4.15: Mögliche Output-Funktion in Infix-Notation (Python)

```
def output_nonnull(s):    #s := argument on stack
    if output.state == mid.sentence:
        write(s+ ", ")       # write is a synonym for write$
    else:
        if output.state == after.block:
            write(add_period(s)) #add_period is a synonm for add.
              period$
            write("\n")
            write("\newblock ")
        else:
            if output.state == before.all:
                write(s)
            else:
                write(add.period(s) + " ")
                output.state = mid.sentence
```

4.2.5 Crossref

Bei Querverweisen arbeitet BibTeX mit einer internen Funktionalität, die auf den referenzierten Eintrag direkt zugreift. Auf SQL umgelegt würde dies bedeuten, eine Unterabfrage im SELECT-Teil oder eine Funktion zu verwenden. Beides ist in vielen SQL-Syntaxen möglich, aber nicht in ANSI-SQL. In ANSI-SQL lassen sich zwei Einträge einfach durch einen Join verknüpfen. Dabei verjoint man die Referenz mit dem referenzierenden Eintrag und erzeugt damit zusätzliche Spalten, die den Wert des referenzierten Datensatzes beinhalten. Dieses Problem lässt sich am einfachsten lösen, wenn man bereits beim Importieren der *bib*-Datei die notwendigen Referenzfelder persistent anlegt. Aus diesem Grund sollte möglichst selektiv vorgegangen werden. Aus den format.crossref Funktionen lässt sich ableiten, welche Eintragstypen mit welchen Eintragstypen referenzieren, beziehungsweise welche Eintragsfelder dabei benötigt werden.
Zum Beispiel wenn article auf article referenziert, werden die Spalten key und journal benötigt. Dafür werden beim referenzierenden Eintrag zwei zusätzliche Spalten angelegt[4].
Folgende Referenzen sind in den Standard *bst*'s implementiert:
1. Book -> Book
2. Inbook -> Book
3. Incollection -> Book
4. Inproceedings -> Proceeding
5. Article -> Article
Folgende Einträge werden von den referenzierten Einträgen benötigt: Das selektive Vor-

	ARTICLE	BOOK	INBOOK	INCOLLECTION	INPROCEEDINGS
volume		x	x		
title				x	x
series		x	x		
key	x	x	x	x	x
booktitle				x	x
journal	x				
editor		x	x	x	x
author		x	x	x	x

Tabelle 4.1: Felder für Querverweise

gehen hat einen entscheidenden Nachteil. Eine styledefinierte Methode ist hartcodiert implementiert. Wenn ein Style zusätzliche Felder benötigt, müsste man den kompletten

[4]Das Anlegen von zusätzlichen Spalten ist problemlos möglich, da bib-Items in einem generischen Datenmodell abgelegt werden.

Datenbestand überarbeiten.
Ein anderer Ansatz wäre für alle Typen alle Felder zu referenzieren. Das hätte aber eine enorme Vervielfachung der Daten zu Folge, was für persistierte Daten nicht sinnvoll erscheint.

Listing 4.16: Beispiel der Funktion format.article.crossref in SQL

```
  CASE WHEN  "key_cr" is NULL   THEN
       CASE WHEN  "journal_cr" is NULL   THEN
           '%{warning$'
||           'need key or journal for '
||           CITEKEY
||           ' to crossref '
||           "crossref"
||           '}'
||           ''
       ELSE
           'In {\em '
||           "journal_cr"
||           '\/}'
       END
  ELSE
       'In '
||       "key_cr"
  END
||   ' \cite{'
||   "crossref"
||   '}'
```

Eintragsfelder des Referenzeintrages sind an der Erweiterung '_cr' zu erkennen.

4.2.6 Purify$

Eine andere Funktion, die nicht mit ANSI-SQL gelöst werden kann, da der Algorithmus zu komplex ist, ist die interne Funktion purify$. Diese Funktion entfernt alphanumerische Zeichen aus Textketten, wenn sie nicht auf ein numerisches Zeichen folgen. In ANSI SQL ist es schwierig alphanumerische von numerischen Zeichen zu unterscheiden. Das würde ein mehrfach verschachteltes Statement ergeben. Erschwerend kommt dazu, dass sich die Prüfung auf das Zeichen davor bezieht.
Da in der Praxis diese Funktion nur für das Feld year verwendet wird, wird dieses ähnlich wie die crossref-Felder beim Importieren der bibliographischen Daten mittels Python-Funktion in die Datenbank geschrieben. Solche Felder erkennt man an der Erweiterung „_pu“, zum Beispiel „year_pu“.

Ein Grund dafür, es bereits in der Datenbank zu halten, ist dass dieses Feld zum Ermitteln des Labels und zum Sortieren notwendig ist. Ein ähnliches Vorgehen wie bei den Output-Funktionen wäre nur dann möglich, wenn man das Sortieren nicht in der Datenbank machen würde.

4.2.7 Dynamische Python-Funktionen

Für andere Funktionen, die nicht in SQL implementiert werden können, können die oben angeführten Methoden nicht angewandt werden. Der Grund dafür ist, dass die Kombination Datenbank und Style das Ergebnis beeinflussen. Das heißt jeder neue Style müsste die *bib*-Datenbank erweitern. Die Folge wäre, dass Styles nicht mehr von der Datenbasis unabhängig wären. Außerdem würde eine enorme Datenmenge erzeugt werden, da jede Kombination von Style und bibliographischem Eintrag zu speichern wäre.
Um dieses Problem zu lösen, ist eine Methode zu schaffen, wie Formatierungen bei Laufzeit auch von einer Python-Routine durchgeführt werden können. Dieses System muss erweiterbar, somit dynamisch sein.
In den Standard-Styles sind es fünf Funktionen, die mittels dynamischer Python-Funktion ersetzt werden müssen. Dafür müssen sie in einen Pseudo-Spaltennamen übersetzt werden, über die das SQL-Statement auf die temporär abgelegten Daten zugreift:

Listing 4.17: Pseudo-Spaltennamen für temporär abgelegte Daten

```
v format.names          -> v1_FORMAT_NAMES
v sort.format.names     -> v1_SORT_FORMAT_NAMES
format.crossref.editor -> editor_FORMAT_CROSSREF
v format.lab.names      -> v1_FORMAT_LAB_NAMES
format.crossref.editor -> editor_FORMAT_CROSSREF
```

Ein Grund ist die interne Funktion format.name$. Sie ist sehr komplex und wäre in SQL schwer umzusetzen. Das würde der Anforderung der Wartbarkeit widersprechen. Ein anderer Grund ist die Verwendung einer while$-Schleife. Schleifen sind in SQL nicht direkt nachbildbar. Die meisten SQL-Dialekte unterstützen prozedurale Spracherweiterungen. Oracle SQL kann zum Beispiel Funktionen verwenden, die in Oracle Pl/SQL geschrieben sind.

Die Methode:
Die Funktion format.name$ wird in einer Python-Klasse nachgebildet. Das steht noch nicht im Widerspruch zu den Anforderungen, da diese Funktion auch in BibTeX eine interne Funktion ist, die vom Anwender nicht veränderbar ist. Die eigentliche *bst*-Funktion, zum Beispiel format.names, wird in Python implementiert und ins Repository exportiert. Ähnlich wie benutzerdefinierte SQL-Funktionen, wird die Python-Funktion in den Metadaten konfiguriert.

Dynamische Python-Funktionen werden vor dem Ausführen der Abfrage auf Basis der bekannten bibliographischen Datenbank ausgeführt. Die Ergebnisse werden temporär in die Datenbank geschrieben. Diese temporäre Tabelle wird mit der Basisabfrage gejoint. Dadurch entstehen künstliche Eintragsfelder, die direkt selektiert werden können.

Die Nachteile:

- Abgehen von der SQL-Implementierung
- Zusätzlicher Schritt im Ablauf
- Aufwendige Implementierung
- Der Anwender muss Python-Code implementieren

Die Funktionen werden nur von den Standard-Styles übernommen. Werden neue Styles eingespielt, müssen sie auf einen bestehenden Style aufbauen, oder neu in Python implementiert werden. Diese Funktionen werden nicht automatisch nach Python übersetzt.

4.2.8 Portierung auf ein proprietäres Datenbanksystem

Verwendet man ein Datenbanksystem, das das Einbinden prozeduraler Funktionen unterstützt, wäre es möglich, bei Weiterentwicklungen auf externe Python Routinen zu verzichten. Der Nachteil wäre, dass die Portierung auf ein anderes Datenbanksystem, dadurch praktisch unmöglich wäre.

4.2.9 Syntaktische Unterschiede zwischen SQLite und Oracle 11g

Um die Portierbarkeit zu gewährleisten, wurde ausschließlich SQL-ANSI-Syntax für die Machbarkeit in Betracht gezogen. Für die Umsetzung wurden zwei Datenbanken gewählt, Oracle und SQLite. Oracle unterstützt seit der Version 9i so wie SQLite die ANSI-SQL Syntax. In der Praxis lässt die ANSI-Interpretation jedoch einen gewissen Spielraum offen. Besonders bei der Handhabung von NULL-Werten ist Vorsicht geboten, da proprietäre Systeme wie Oracle auf Abwärtskompatibilität Rücksicht nehmen müssen. Bei SQLite wiederum ist Vorsicht geboten, da aus Kompatibilitätsgründen viele Features angeboten werden, die aber nur syntaktisch funktionieren. Die folgenden Kapitel erläutern kurz mögliche Problempunkte im Bezug auf die Kompatiblität zwischen SQLite und Oracle 11g.

Outer-Join

Oracle verwendet ursprünglich die „(+)“ Syntax für Outer-Joins. Ab der Version Oracle 9i ist auch die ANSI-Syntax „OUTER JOIN“ möglich.

SQL String Funktionen

Replace()
Die Funktion Replace() ist für beide Datenbanksysteme gleich. Reguläre Ausdrücke sind nur in Oracle (ab Oracle 9i) mit der Funktion regexp_replace möglich.
Concate()
Die Syntax „||“ ist in beiden Datenbanken möglich, die Funktion concat() jedoch nur in Oracle.
Oracle gibt nicht NULL zurück, wenn ein Wert mit NULL konkatiniert wird, egal ob mit „||“ oder der Funktion CONCAT() gearbeitet wird.
Deshalb sollte in SQLITE, damit es äquivalent zu Oracle ist, ifnull(val1,'')||ifnull(val2,'') verwendet werden.
So liefert beispielweise die Funktion Ifnull('{\em'||field_name||'}','') in SQLite Null zurück, wenn der Wert für fieldname NULL ist. Ähnliches Verhalten könnte man in Oracle mit Nvl2 (field_name,'', '{\em'||field_name||'}') abbilden.

Die Funktion NULL-Values

Die Oracle-Funktion nvl() ist äquivalent zur SQLite ifnull(). Die Oracle-Funktion nvl2 gibt es in SQLite nicht. Nvl2() kann in SQLite nur mit einer CASE WHEN-Konstruktion nachgebildet werden.

Decode

Die Funktion decode() ist eine Oracle SQL-Erweiterung. Diese Funktion kann unter SQLite mit CASE WHEN (ANSI Syntax) nachkonstruiert werden.
Zum Beispiel decode(y,x,0,1) entspricht case when y=x then 0 else 1 end.
Ab der Version Oracle 8i wird ebenfalls die ANSI-Syntax unterstützt. Dabei ist auf den Vergleich von NULL-Werten zu achten:
NULL im Vergleich ist immer falsch, (NULL = NULL) ergibt falsch. In einer Anweisung decode (NULL,NULL,..) wäre die Prüfung richtig.

Instring

Eine wichtige Funktion für die Textmanipulation, die es in SQLite nicht gibt, ist instring(). Diese Funktion gibt die Position im String zurück, an der das erste Vorkommen eines Zeichens festgestellt wurde. Das macht auch einige substring-Methoden schwierig, wenn ab einer bestimmten Position eines vorkommenden Zeichen abgeschnitten werden soll.

Upper, Lower und Initcap

Die Funktion initcap() wandelt das erste Zeichen auf Großbuchstaben um, alle folgenden auf Kleinbuchstaben. Diese Funktion existiert in SQLite nicht.
Eine Alternative wäre:
replace(lower(name), substr(name,1,1),upper(substr(name,1,1)))

Select auf Spaltenname

Oracle unterscheidet im Gegensatz zu SQLite im Bezug auf Objekt- und Spaltennamen nicht zwischen Groß- und Kleinschreibung. Wird unter Oracle ein Objekt oder eine Spalte angelegt, und dabei nicht mit einem doppelten Hochkomma eingeschlossen, wird der Name implizit in Großbuchstaben angelegt. Der Zugriff verhält sich gleich, alles wird implizit auf Großbuchstaben umgewandelt.
Um die Groß- und Kleinschreibung zu berücksichtigen, müssen Namen mit doppeltem Hochkomma eingeschlossen werden[5]. SQLite unterscheidet sich dadurch, dass es keinen Unterschied zwischen "name" oder name gibt, in beiden Fällen wird auf die Spalte name zugegriffen.

Auswirkung auf die Implementierung

Um sicherzustellen, dass keine Konflikte mit Schlüsselwörtern eintreten, werden Spalten immer mit Hochkomma eingeschlossen[6]. So würde beispielweise das Eintragsfeld number, im Select-Teil eines SQL-Statements unter Oracle zu einem Syntaxfehler führen, da es sich um ein reserviertes Wort handelt. Wird eine Spalte mit Hochkomma eingeschlossen ist jeder beliebige Text möglich[7].

[5] Zum Beispiel SELECT "name" from TABLE entspricht nicht SELECT name from TABLE.
[6] Da BibTeX nicht zwischen Groß-/Kleinschreibung unterscheidet, stellt es kein Problem dar, in Oracle und SQLite alle Bezeichner in Kleinbuchstaben mit doppelten Hochkomma eingeschlossen darzustellen.
[7] In jedem Fall besteht unter Oracle das Limit von maximal 30 Zeichen.

Listing 4.18: Zugriff auf Spalten der bibliographischen Datenbank

```
SELECT b.NAME, bi.item_type, bi.cite_key, NVL ("address", ' ') "
  address",
       NVL ("address_cr", ' ') "address_cr", NVL ("author", ' ') "
         author",
       NVL ("author_cr", ' ') "author_cr",
       NVL ("booktitle", ' ') "booktitle",
       NVL ("booktitle_cr", ' ') "booktitle_cr",
       NVL ("chapter", ' ') "chapter", "crossref" "crossref",
       NVL ("edition", ' ') "edition", NVL ("editor", ' ') "editor"
         ,
```

Textkonstanten in SQL

Textkonstanten sind unter Oracle ausschließlich unter einfachem Hochkomma erlaubt[8]. In SQLite verhält es sich ähnlich wie in Python, sowohl doppelte als auch einfache Hochkommas sind möglich. Wenn ein doppeltes Hochkomma in der Textkonstante vorhanden ist, so stellt dies unter Oracle kein Problem dar. In SQLite ist es kein Problem, wenn mit einfachem Hochkomma eingeschlossen wird. Problematischer ist es bei einfachem Hochkomma. Unter Oracle wird es mit einer Doppelschreibung des Hochkommas eingefügt; So ergibt zum Beispiel select 'd''s' from dual den Wert d's. Diese Methode ist unter SQLite auch zulässig. Ein Hochkomma am Anfang und Ende einer Textkonstante wird mit drei aneinandergereihten Symbolen dargestellt, sowohl in SQLite als auch in Oracle.

Dummy Tabelle dual

In Oracle existiert eine Tabelle namens DUAL. Diese Tabelle gibt bei einer Abfrage genau eine Zeile zurück. Das ist besonders beim Generieren von Text wichtig. In SQLite gibt es weder eine Tabelle DUAL noch etwas Ähnliches. Um die Kompatibilität zu Oracle zu gewährleisten, kann diese Tabelle einfach angelegt werden[9].

Pseudo-Spalten

Unter Oracle gibt ROWNUM die Zeilennummer zurück. Eine Zeilennummernspalte gibt es in SQLite nicht. So etwas könnte nur im Programmcode angeführt werden, zum Beispiel während des Cursor-Loops. ROWID ist die interne Nummer einer Zeile. Um ein-

[8] zum Beispiel ‘text konstante‘

[9] Create table DUAL (DUMMY varchar(1)); insert into DUAL (DUMMY) values (‚X‘)

deutige Nummern für jede Zeile zu bekommen, kann man sich in einigen Fällen mit der Zeilennummer ROWID helfen. In SQLite kann die interne ROWID dafür genutzt werden, um aufsteigende Zahlen zu erhalten.
Unter Oracle ist dieser Ansatz nicht empfehlenswert, da ROWID keine Nummer ist, sondern ein Wert mit dem Datentyp ROWID.

4.3 Technisches Konzept

4.3.1 Grobkonzept der Umsetzung

Die Umsetzung erfolgt auf Basis einer klassischen Client-Server-Architektur. Die Datenhaltungsschicht erfolgt ausschließlich mit einem relationalen Datenbank-System. Die Datenbankschicht ist nach relationalen Kriterien in der 3NF modelliert. Die Datenbankschicht ist in Standard-SQL gehalten. Zugriffe erfolgen ausschließlich über Python-Klassen. Entitäten sind in eigenen Klassen gekapselt. Der Zugriff erfolgt über Methoden dieser Klassen. SQL-Abfragen und DML-Statements sind in ANSI-SQL implementiert. Dies erlaubt, sofern ein entsprechendes API für Python 2.4 zur Verfügung steht, ein beliebiges RDBMS als Datenbankschicht einzubinden. Komplexe Statements sind in eigenen SQL-Dateien gespeichert, und werden bei Laufzeit in das Programm eingebunden. Der Datenbankzugriff ist in einem eigenen Modul[10] in der Klasse „database“ gekapselt. Im Wesentlichen wird auf produktspezifische Features der Datenbanksysteme verzichtet. Das heißt es werden keine Funktionen, Prozeduren und Trigger verwendet. Views werden ausschließlich in ANSI-SQL umgesetzt. Eine Ausnahme sind „Oracle Sequences“, da es nicht zweckmäßig wäre, unter einer Serverdatenbank, die einen sehr hoch konkurrierenden Zugriff ermöglicht, die Vergabe von Surrogatschlüsseln auszuprogrammieren, anders als bei Datenbanken wie SQLite, die für den lokalen Zugriff konzipiert sind. Das Programm ist vollständig in Python geschrieben und sollte plattformunabhängig funktionieren. Metadaten werden in der Datenbank gehalten. Für die Wartung der Metadaten ist ein entsprechendes Tool zu verwenden. Konfiguration und Datenbankeinstellungen werden in zwei Python-Dateien gehalten.[11]
Auf “Object Relationale Mapper“, wie zum Beispiel SQLAlchemy beziehungsweise Modle-View-Mapper, wurde bewusst verzichtet. Da in dieser Version weder eine graphische Benutzeroberfläche noch eine Web-Applikation geplant sind und die Zugriffe über SQL eher speziell als generell sind, wurden Klassen mit speziellen Methoden ausprogrammiert. Komplexe Zugriffe werden direkt in SQL Abfragen implementiert. Das kann für spätere Weiterentwicklungen zum Nachteil werden, nimmt aber Komplexität aus dieser Version.

4.3.2 Beschreibung der technischen Architektur

Ausgehend von den funktionalen Anforderungen (siehe Abbildung 4.1) muss eine Applikation geschaffen werden, die den Workflow von BibTeX nachbildet.

[10] lib/database.py
[11] lib/config.py und lib/database.py

BibTeX wird durch die Neuimplementierung BibSQL komplett abgelöst. Der neuimplementierte Präprozessor muss die volle BibTeX-Funktionalität[12] übernehmen.

4.3.3 Datenfluss

Bibliographische Datenbanken (*bib*-Files) werden, wie in Abbildung 4.2 dargestellt, in einem vorangehenden Schritt direkt in der Datenbank persistiert. Beim Ausführen von BibSQL werden ausschließlich bibliographische Daten aus der Datenbank verwendet. *Bst*-Style-Files werden übersetzt und in einem relationalen Datenbankmodell abgelegt.

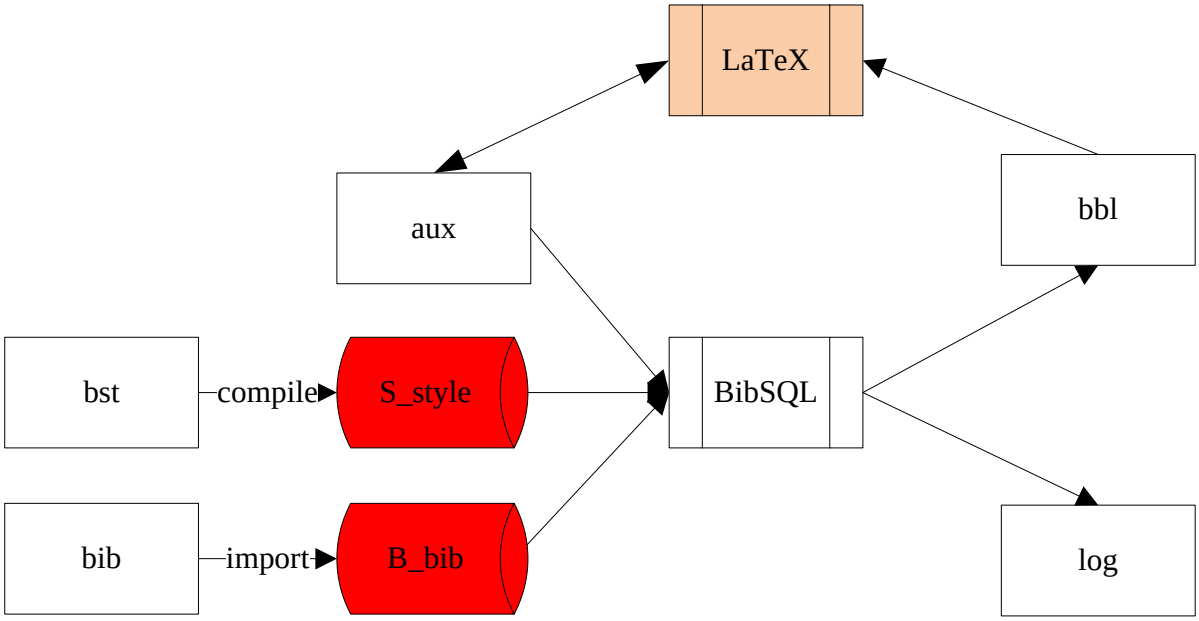

Abbildung 4.2: BibSQL-LaTeX Workflow

Bei Laufzeit wird ein SQL Statement generiert, das direkt den Output für das *bbl*-File erzeugt.

[12] Lesen von *bib*-, *aux*- und *bst*-Files; Erzeugen eines LaTeX-kompatiblen *bbl*-File

4.4 Umsetzung BibSQL

Ziel dieses Projektes ist es, die Strukturen von BibTeX in einem relationalen Modell abzubilden. Dabei soll die Anwendungslogik von BibTeX erhalten bleiben. Im ersten Schritt werden die Entitäten beschrieben, die dafür vorgesehen sind, BibTeX in einem Datenbanksystem zu halten.
Im zweiten Schritt wird auf die physische Tabellenstruktur eingegangen. Über die Entitäten des Modells hinausgehend werden auch Tabellen beschrieben, die für den funktionalen Ablauf notwendig sind.

4.4.1 ER-Modell

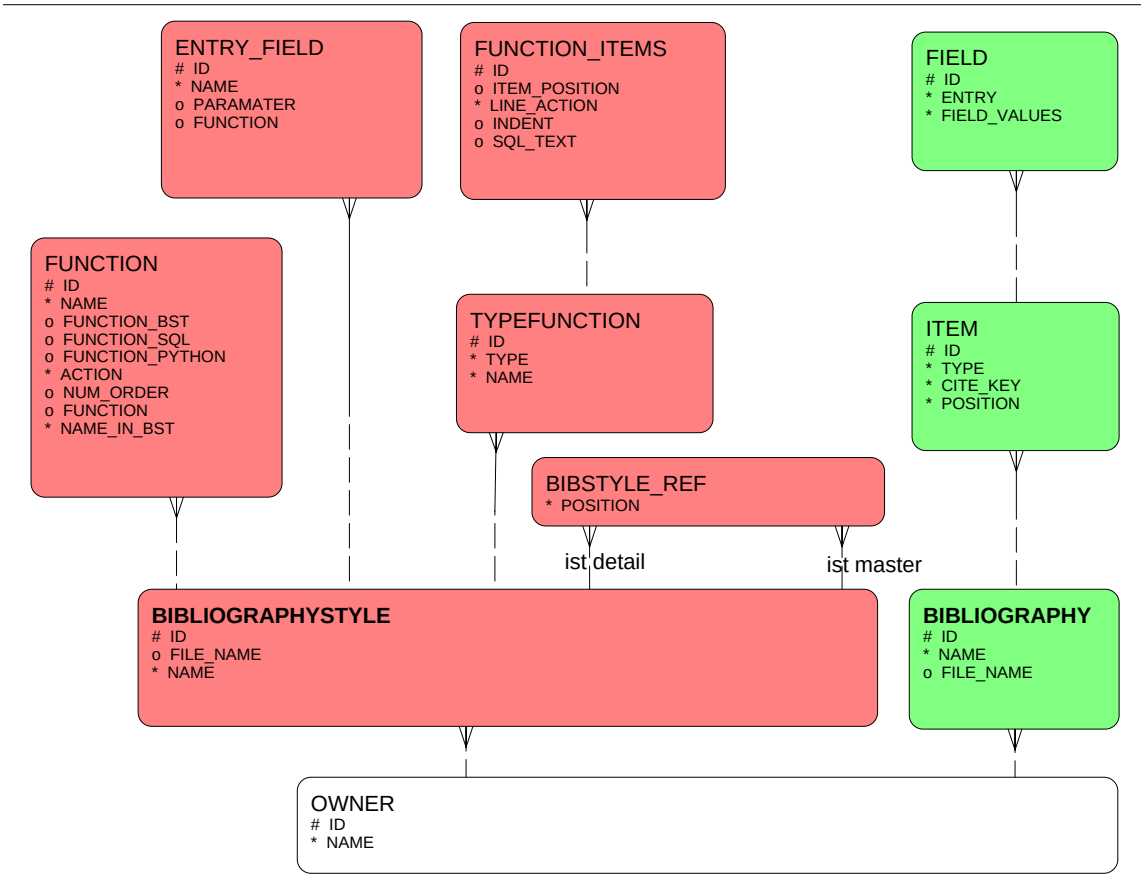

Abbildung 4.3: ER-Diagramm BibSql

Das ER-Modell wird in Abbildung 4.3 als ER-Diagramm dargestellt.

Grundsätzliches
Jede Entität hat einen Primärschlüssel (ID). Diese wird bei der Befüllung mit einem Surrogatwert befüllt. Das Attribut ID wird in Folge nicht mehr angeführt. Das Symbol „*“ beschreibt Pflichtfelder, das Symbol „o“ beschreibt optionale Felder.

OWNER

Diese Entität beschreibt den Eigentümer der darüber geordneten Einträge. Diese Entität wird benötigt, wenn die Applikation als Serverlösung implementiert wird. In diesem Fall kann jedem Benutzer ein OWNER zugeordnet werden, um zu verhindern, dass fremde Benutzer Einträge bearbeiten können.

Attribute:
* NAME: Beschreibt den Benutzer

Unique Key:
UK1: NAME

Beziehungen:
Ein Owner kann eine oder mehrere BIBLIOGRAPHY haben
Ein Owner kann einen oder mehrere BIBLIOGRAPHYSTYLE haben

BIBLIOGRAPHY

Diese Entität beschreibt eine Literaturliste. Diese entspricht in BibTeX einer bibliographischen Datei (*.bib).
Eine BIBLIOGRAPHY ist eine Sammlung von mehreren Literatureinträgen. Wenn eine *bib*-Datei importiert wird, dann wird für diese Datei ein Eintrag geschrieben.

Attribute:
* NAME: Name der *bib*-Datei ohne Erweiterung .bib
o FILE_NAME: Name der Datei

Unique Key:
UK1: NAME
Der Name muss eindeutig sein, nicht aber der Dateiname. Alle Folgeschritte beziehen sich auf den Namen. Dies gewährleistet, dass eine Datei mit demselben Namen öfters eingelesen werden kann wenn der Name vorher geändert wurde. Die Eindeutigkeit bezieht sich nur auf den Namen, nicht auf den OWNER, was bedeutet, dass unterschiedliche Benutzer nicht denselben Namen verwenden können. Dadurch wird ermöglicht, dass mehrere Benutzer eindeutig auf einen bibliographischen Eintrag zugreifen können, auch dann, wenn er einem anderen OWNER gehört.

Beziehungen:
FK1: OWN_ID
Eine BIBLIOGRAPHY muss einen OWNER haben
Eine BIBLIOGRAPHY kann ein oder mehrere ITEM haben

ITEM

Ein ITEM beschreibt einen bibliographischen Eintrag, ein Buch, ein Journal, einen Artikel etc. Dieser entspricht dem führenden Eintrag einer Bibliographie in einer *bib*-Datei (zum Beispiel: `@Book{herrmann02,..`).
Der Eintrag nach dem at-Zeichen entspricht dem Typ. Nach der geschwungenen Klammer folgt der Zitierschlüssel. Wie in BibTeX ist die Position eines Eintrages nicht relevant. Damit die Originalinformation erhalten bleibt, wird diese aber auch als Attribut übernommen. Damit ist es möglich, aus der *bib*-Datenbank *bib*-Dateien im ursprünglichen Zustand zu erzeugen (Export)[13].

Attribute:
* TYPE: Typ des bibliographischen Eintrages (zum Beispiel „book“, „article“, „proceedings“ usw.).
* CITE_KEY: Zitierschlüssel, der in der *bib*-Datei sowie im LaTeX-Dokument verwendet wird.
* POSITION: Numerischer Wert, der die Position des Eintrages beschreibt[14]

Unique Key:
UK1:POSITION, B_BIB_ID

[13] Dzt. nicht im Funktionsumfang
[14] In aufsteigender Reihenfolge

Die Position eines ITEMs muss innerhalb einer BIBLIOGRAPHY eindeutig sein.
UK2: CITE_KEY
Der CITE_KEY muss global eindeutig sein. Das heißt unterschiedliche Bibliographien und Benutzer können nicht zweimal denselben Zitierschlüssel verwenden.

Beziehungen:
FK1: B_BIB_ID
Ein ITEM muss einer BIBLOGRAPHY zugeordnet sein
Ein ITEM kann ein oder mehrere FIELD haben

Logisch steht das Attribut TYPE mit dem Attribut NAME der Entität TYPEFUNCTION in Beziehung, die den Typ BIBTYPE beschreit. In BibTeX gibt es diese Beziehung nicht, da eine Stildatei unabhängig von den bibliographischen Einträgen ist. Das heißt beim bibliographischen Eintrag wird nicht geprüft, ob es in einer Stildefinition diesen Typ gibt. In BibTeX werden unbekannte Typen mit der Typenfunktion misc formatiert.

FIELD

Diese Entität beinhaltet die einzelnen Felder (ENTRY) eines bibliographischen Eintrages. Um BibTeX zu entsprechen, wurde der generische Ansatz gewählt. Da BibTeX die unterschiedlichen Feldarten nicht vorgibt, wird die Struktur eines ITEM nicht vorgegeben. Erst der Stil definiert in welcher Weise Felder formatiert werden. Feldarten, die vom Stil nicht erkannt werden, werden bei der Formatierung ignoriert. Für die unterschiedlichen Eintragstypen gibt es aber Pflichtfelder. Diese werden in der *bib*-Datenbank nicht überprüft, da sie vom Stil unabhängig sind. Das heißt fehlende Felder erzeugen erst beim Formatieren eine Fehlermeldung.
Felder entsprechen dem Teil in der *bib*-Datei, der zwischen Zitierschlüssel und der schließenden Klammer liegt, wie im folgenden Beispiel:

Listing 4.19: Felder eines bibliographischen Eintrages

```
author =  {Gregor von Laszewski and Patrick Wagstrom},
title =   {Gestalt of the Grid},
booktitle = {Tools and Environments for Parallel and Distributed
  Computing},
publisher = {JW},
year =   {2004},
editor =  {Salim Hariri and Manish Parashar},
chapter = {5},
pages =     {149--188}
```

Attribute:
* ENTRY: beschreibt den Feldtyp, zum Beispiel: „author“, „title“, etc.
* FIELD_VALUES: beinhaltet den eigentlichen Eintrag.

Unique Key:
UK1: ENTRY, B_ITEM_ID
Ein Feldtyp muss innerhalb eines ITEM eindeutig sein.

Beziehungen:
FK1: B_ITEM_ID
Ein FIELD muss einem ITEM zugeordnet sein

BIBLIOGRAPHYSTYLE

Diese Entität beschreibt einen bibliographischen Stil. In BibTeX entspricht ein BIBLIOGRAPHYSTYLE einem Style-File (*.bst). Sie bildet die übergeordnete Tabelle der Stilmetadaten. Bei der Übersetzung einer *bst*-Datei wird ein BIBLIOGRAPHYSTYLE angelegt. Der Name bildet sich aus dem Dateinamen ohne Erweiterung.

Attribute:
* NAME: Name des bibliographischen Stils, zum Beispiel: „plain“, „alpha“, „unsrt“ usw.
o FILE_NAME: Der Dateiname ist optional und wird befüllt, wenn eine bestehende *bst*-Datei importiert wird.

Unique Key:
UK1: NAME
Der Stilname muss eindeutig sein. Das heißt unterschiedliche OWNER können nicht denselben NAME für einen Stil verwenden. Der Dateiname kann beliebig oft verwendet werden. Wenn eine *bst*-Datei mit demselben Namen ein weiteres mal eingelesen wird, muss der Namen vorher geändert werden.

Beziehungen:
FK1: OWN_ID
Ein BIBLIOGRAPHYSTYLE muss einem Owner zugeordnet sein.
Eine BIBLIOGRAPHY kann eine oder mehrere FUNCTION haben.
Eine BIBLIOGRAPHY kann eine oder mehrere TYPEFUNCTION haben.

Eine BIBLIOGRAPHY kann ein oder mehrere ENTRY_FIELD haben.
Eine BIBLIOGRAPHY kann ein oder mehrere BIBLOGRAPHYSTYLE haben.

BIBSTYLE_REF

Diese Entität dient zur Verknüpfung eines bibliographischen Stils mit sich selbst. Dies geschieht, um Stile zu modularisieren. Einem BIBLOGRAPHYSTYLE kann ein anderer zugeordnet werden, der seine Eigenschaften vererbt. In der Praxis bedeutet das, dass Funktionen, Typenformatierungen und Eintragsfelder von einem Stil auf den anderen übertragen werden können.

Attribute:
* POSITION: POSITION ist ein numerischer Wert, der die Reihenfolge der Vererbung definiert. Verknüpfungen werden in aufsteigender Reihenfolge der POSITON gewichtet. Die höchste Priorität hat der Haupteintrag selbst.

Unique Key:
UK1: S_BST_MASTER_ID, POSITION
Die Position für einen Haupteintrag ist eindeutig.
UK2: S_BST_MASTER_ID, S_BST_DETAIL_ID
Für einen Haupteintrag kann es denselben Detaileintrag nur einmal geben.

Beziehungen:
FK1: S_BST_MASTER_ID
FK2: S_BST_DETAIL_ID
Die Entität BIBSTYL_REF stellt eine n:m Beziehung der Entität BIBLOGRAPHYSTYLE mit sich selbst dar. Diese Beziehung ist optional und kann mehrfach sein.

ENTRY_FIELD

ENTRY_FIELD bezeichnet Feldtypen eines bibliographischen Stils. Sie stellen das Gegenstück zur Entität FIELD einer BIBLIOGRAPHY dar, stehen aber nicht in einer direkten Beziehung. Felder eines Stils sind die generischen Attribute der bibliographischen Datenbank. Attribute (Feld-Typen) werden ausschließlich vom bibliographischen Stil vorgegeben. Die Menge aller ENTRY_FIELD eines Stils stellt somit die Attributliste dar, die beim Abfragen der Datenbank dynamisch zusammengestellt wird.

Im Normalfall wird ENTRY_FIELD aus der *bst*-Datei übernommen, in welcher die Felder in einem Deklarationsblock definiert sind. Andere Feldtypen, die künstlich erzeugt werden, greifen bei der Abfrage auf einen temporären Datenbestand zu, der während der Verarbeitung durch Python-Funktionen erzeugt wurde.

Attribute:
* NAME: Name des Feldtyps (zum Beispiel „author“, „titel“, etc.)

o PARAMATER: Dieses Attribut ist ein optionales Feld und wird verwendet wenn hinter dem Feldtyp eine Funktion steht. Der Parameter wird ähnlich wie eine ENTRY_FIELD verwendet.
o FUNCTION: FUNCTION gibt den Namen der Funktion an, die hinter einem errechneten Feld liegt. Die Funktion selbst ist in der Entität FUNCTION beschrieben.

Unique Key:
UK1: NAME, S_BST_ID
Für einen bibliographischen Stil ist der Name des Feldtyps eindeutig.

Beziehungen:
FK1: S_BST_ID
Ein ENTRY_FIELD muss einem BIBLIOGRAPHYSTYLE eindeutig zugeordnet sein.

FUNCTION

Die Entität FUNCTION beschreibt *bst*-Funktionen und gibt Übersetzungsregeln vor. Im Gegensatz zu anderen Entitäten des bibliographischen Stils wird der Inhalt dieser Tabelle nicht aus der *bst*-Datei übertragen, sondern ist eine Vorgabe von Regeln für die Übersetzung. Diese Metainformationen werden beim Übersetzen aus einer Konfigurationstabelle gelesen und mit dem übersetzten Stil verknüpft. Das ist notwendig, weil diese Metainformationen nicht nur beim Übersetzten notwendig sind, sondern auch bei Laufzeit[15] benötigt werden.

Attribute:
* NAME: Dieses Attribut beschreibt den Namen einer Funktion. Dieser entspricht der Funktion, wie sie in BibTeX verwendet wird. In den meisten Fällen ist das der Name

[15] Modul bibtex

der Funktion. In einzelnen Fällen steht auch ein Parameter als Textkonstante vor dem Funktionsnamen, zum Beispiel: ‘I‘ change.case$‘.
o FUNCTION_BST: Dieses Attribut gibt die Funktion inklusive ihren Parametern an.
o FUNCTION_SQL: Dieses Attribut beschreibt die Übersetzung nach SQL.
o FUNCTION_PYTHON: Dieses Attribut beinhaltet die Übersetzung nach Python.
* ACTION: ACTION gibt an, wie diese Funktion beim Übersetzen behandelt werden soll.
o NUM_ORDER: Dieses Attribut gibt die Reihenfolge beim Übersetzen an.
o FUNCTION: FUNCTION beschreibt den Funktionsnamen nach der Übersetzung.
* NAME_IN_BST: Dieses Attribut ist der Name der Funktion in bst. Er entspricht exakt dem Namen aus der *bst*-Datei.

Unique Key:
UK1: NAME, S_BST_ID
Für jeden bibliographischen Stil muss der Name der Funktion eindeutig sein.

Beziehungen:
FK1: S_BST_ID
Ein ENTRY_FIELD muss einem BIBLIOGRAPHYSTYLE eindeutig zugeordnet sein.

TYPEFUNCTION

Diese Entität beinhaltet die übersetzten Funktionen. Sie beschreibt den Namen und typisiert die Funktion. Typen können die Ausprägung „FORMAT“, „BIBTYPE“, „SORT“ oder „LABEL“ haben.

Attribute:
* NAME: NAME ist der Name der übersetzten Funktion und entspricht in den meisten Fällen dem Original in der *bst*-Datei.
* TYPE: TYPE klassifiziert die Funktion. Das ist für das Erzeugen der Abfrage auf die bibliographischen Daten relevant.

Unique Key:
UK1: NAME, S_BST_ID
Für einen bibliographischen Stil ist der Name des Feldtyps eindeutig.

Beziehungen:
FK1: S_BST_ID
Ein ENTRY_FIELD muss einem BIBLIOGRAPHYSTYLE eindeutig zugeordnet sein.

FUNCTION_ITEMS

Diese Entität beinhaltet die einzelnen Zeilen der übersetzten Funktionen. Bei der Übersetzung der *bst*-Datei wird das Ergebnis zeilenweise in diese Tabelle geschrieben.

Attribute:
* SQL_TEXT: Dieses Attribut beinhaltet den Text der Funktion. Anzumerken ist, dass Funktionen in einer eigenen Zeile stehen müssen, da sie beim Aufbau der Abfrage mit dem Funktionsnamen der Entität TYPEFUNCTION verknüpft werden. SQL-Text steht ohne Einrückungen, dieses Formatierungsmerkmal wird in einer eigenen Spalte beschrieben.
* ITEM_POSITION: Dieses Attribut ist ein numerisches Feld und gibt in aufsteigender Reihenfolge die Zeilenposition an.
* LINE_ACTION: Dieses Attribut beschreibt, wie die Zeile mit der vorhergehenden Zeile verknüpft wird. Dieses Attribut ist optional, da nicht alle Zeilen miteinander verknüpft werden müssen. In den meisten Fällen ist das ein Konkatinierungssymbol „||“ oder leer.
* INDENT: INDENT ist ein reines Formatierungsmerkmal, das die Anzahl der eingerückten Leerzeichen für die formatierte Ausgabe angibt.

Unique Key:
UK1: ITEM_POSITION, S_TYPEF_ID
Für jede TYPEFUNCTION muss die Position eindeutig sein.

Beziehungen:
Jedes FUNCTION_ITEM muss einer TYPEFUNCTION zugeordnet sein.

4.4.2 Physisches Tabellenmodell

In Abbildung 4.4 wird das physische Datenmodell beschrieben. Entitäten werden als Tabellen dargestellt, Attribute, Primär- und Fremdschlüssel als Spalten. Außerdem sind Constrains, Datentypen und Indexe angeführt.

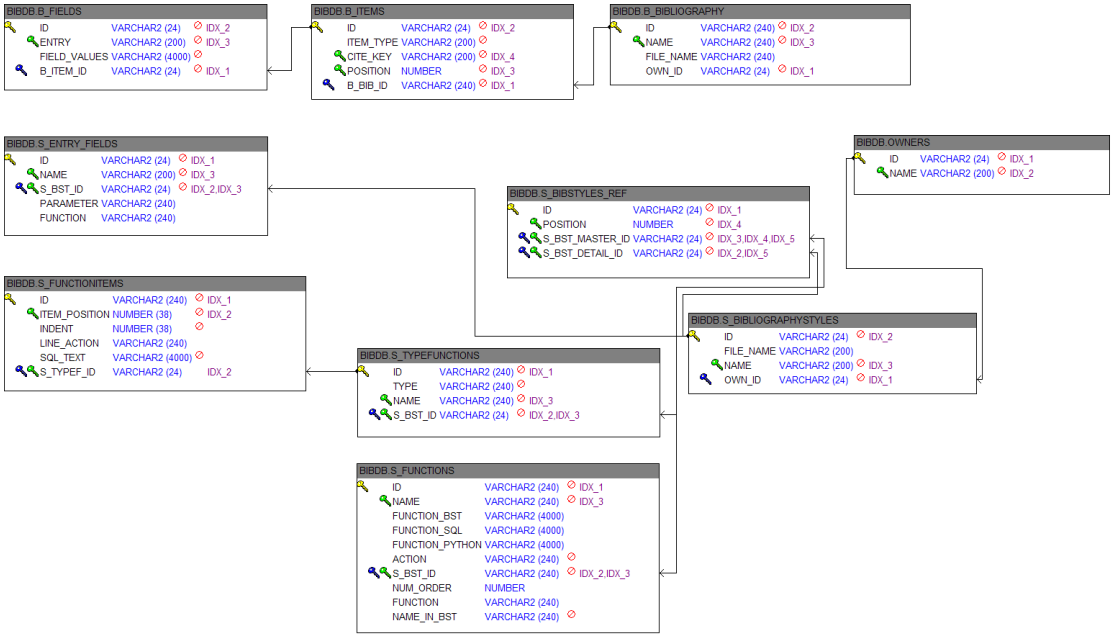

Abbildung 4.4: Physisches Datenmodell BibSQL

4.4.3 Technische Tabellen

In diesem Kapitel werden Tabellen beschrieben, die nicht direkt dem BibSQL Datenmodell angehören. Diese Tabellen werden zum Halten von Metadaten (Konfigurationen, Hilfebeschreibungen) oder zum Speichern von Zwischenergebnissen verwendet.

META

Tabellen mit dem Präfix „META“ sind Tabellen, die vordefinierte Informationen beinhalten, die für den Übersetzungsvorgang benötigt werden. In den meisten Fällen sind sie gleich aufgebaut wie die eigentlichen Repository-Tabellen. Sie können auch als Konfigurationstabellen für den Benutzer betrachtet werden.

META_ENTRY_FIELDS
Beinhaltet vordefinierte Feldtypen für spezielle Funktionen, wie zum Beispiel Crossrefe-

renzen.

META_TYPEFUNCTIONS, **META_FUNCTIONITEMS**
Diese Tabellen beinhalten vordefinierte Funktionen. Das sind Funktionen, die nicht übersetzt werden. Die Spalte BST_NAME gibt an, für welchen bibliographischen Stil diese Funktionen Verwendung finden. Wenn diese Spalte leer ist, wird diese Definition universal verwendet.

META_FUNCTIONS
Diese Tabelle beinhaltet Benutzereinstellungen, die vom Übersetzungsvorgang verwendet werden.

TEMP

Tabellen mit dem Präfix „TEMP“ sind temporäre Tabellen, die hauptsächlich zwischen dem Parsen und dem Übersetzungsvorgang genutzt werden. Sie beinhalten Zwischenergebnisse der geparsten *bst*-Dateien. Die Inhalte dieser Tabellen werden vor jedem Parsevorgang automatisch gelöscht.

TEMP_ENTRY_FIELDS
TEMP_FUNCTIONS
TEMP_MACROS
TEMP_STR
TEMP_VARS

Eine Ausnahme stellt die Tabelle TEMP_B_FIELDS dar. Sie wird für den BibSQL-bibtex-Vorgang[16] als Zwischenspeicher für vorberechnete Feldeinträge verwendet.

INFO

Die Tabelle INFO_BUILDIN beinhaltet alle BibTeX spezifischen *bst*-Funktionen mit einer Beschreibung. Diese Tabelle wird in der aktuellen Version nicht aktiv genutzt, könnte aber in Zukunft als Datenbasis für eine Hilfefunktion verwendet werden.

[16] Python-Module bibtex

4.4.4 Überblick der Umsetzung BibSQL

Grundsätze der Umsetzung
Die BibTeX-Kernapplikation wurde komplett neuimplementiert. Die Schnittstelle sowie Art und Weise, wie BibTeX mit LaTeX zusammen arbeitet, wurde unverändert gelassen. Die Kommunikation zwischen LaTeX und BibTeX erfolgt ausschließlich über Files. Der eigentliche Satz der Bibliographie erfolgt mit dem internen LaTeX-Modul „the bibliograph environment“.
Der Informationsaustausch zwischen LaTeX und BibTeX passiert über das *aux*-File, das von LaTeX geschrieben wird, und das *bbl*-File, das von BibTeX erzeugt wird.
BibTeX verwendet als Input das *aux*-File von LaTeX und erzeugt als Output das *bbl*-File, dies dient wiederum LaTeX als Input. Weitere Input-Files von BibTeX sind das *bib*-File und das *bst*-File, beide sind reine BibTeX-Files. LaTeX übergibt die Information, welche *bib*-Datei (bibliographische Datenbank) und welche *bst*-Files (bibliographischer Style) verwendet werden, und zwar über das *aux*-File.
Für die Neuimplementierung von BibTeX bedeutet das, dass das *aux*-File bei Laufzeit gelesen und interpretiert werden muss, *bib*-und *bst*-Files werden in einem separaten Schritt geparst und in die Datenbank geschrieben. Bestehende *bst*-und *bib*-Files können wiederverwendet werden, indem sie ins Repository importiert werden. Abgesehen davon können bibliographische Daten auch direkt in die Datenbank eingegeben werden.

4.4.5 Grundarchitektur

Die Art und Weise, wie der BibTeX-Präprozessor arbeitet, wird im Wesentlichen vom Stylefile (bst) bestimmt. Das Stylefile legt fest, wie bibliographische Einträge formatiert und sortiert werden.
Im Gegensatz zu BibTeX hält BibSQL Informationen zur Formatierung und Sortierung in einem Datenbankrepository (siehe Abbildung 4.5). Aus diesem Repository wird bei Laufzeit ein SQL-Statement generiert, das den vorformatierten Text und den Zitierschlüssel in einer eigenen Spalte ausgibt. Dieser Cursor wird in einer dem Stile entsprechenden Sortierung ausgegeben. Ziel der neuen Architektur war es, die Ausgabe direkt aus dem SQL-Cursor ins File zu schreiben. Dieses Ziel wurde aus programmtechnischen Gründen nicht zu 100% erreicht, da es nur sehr schwierig möglich ist, Satzsetzungen und Textfluss wie Beistriche, Punkte und Leerzeichen in einem ANSI-SQL-Statement zu verwenden. Um das zu erreichen, wäre es notwendig gewesen, mehrfach verschachtelte SQL-Statements zu erzeugen. Vielmehr wurde auf das Architekturziel, ausschließlich ANSI-SQL ohne prozedurale Elemente zu verwenden, Rücksicht genommen[17]. Aus dem-

[17] Satzsetzungsfunktionen wären in Sprachen wie Oracle-PL/SQL einfach zu implementieren gewesen, dies hätte aber die Auswirkung gehabt, dass BibSQL nicht von der Datenbank unabhängig ist.

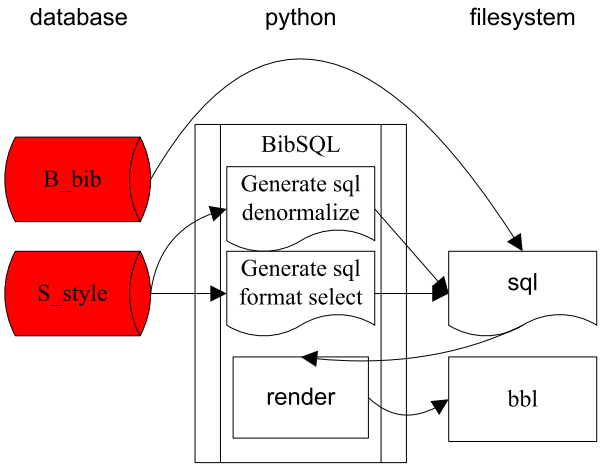

Abbildung 4.5: Grundarchitektur von BibSQL

selben Grund wurde auf alle produktspezifischen SQL-Spracherweiterungen verzichtet. Der Großteil der internen BibTEX-Funktionen konnte in eine äquivalente SQL-Funktion übersetzt werden. In einigen Fällen war das nicht möglich. Dadurch ergab sich die Notwendigkeit, die Möglichkeit zu schaffen, Python-Funktionen einzubinden.
Abbildung 4.6 zeigt wie das Ergebnis der SQL Abfrage von der Python Funktion render verwendet wird. Diese Python-Funktionen werden vor dem *bst*-SQL Übersetzungsvor-

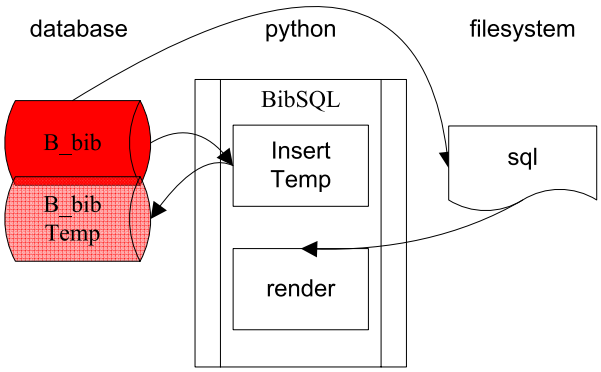

Abbildung 4.6: Grundarchitektur von Python-Funktionen

gang erstellt und im SQL-Code als Pseudofunktion eingebunden. Der Wert wird bei Laufzeit in eine temporäre Tabelle eingefügt und anschließend von einer SQL-Abfrage wie eine gewöhnliche Spalte angesprochen. In den Standard *bst*-Files war nur die Funk-

tion Format.Name betroffen[18].
Die bibliographische Datenbank ist generisch aufgebaut. Das *bib*-File beinhaltet neben den bibliographischen Daten auch die Beschreibung der Struktur. In der Theorie können beliebige Eintragstypen und Feldbezeichnungen verwendet werden. Erst die Verarbeitung, die durch das Stylefile vorgegeben ist, legt fest, welche Typen verarbeitet werden und welche Felder zwingend notwendig sind.
Auf dieses generische Verhalten wurde beim Datenbankmodell Rücksicht genommen und es wurde ebenfalls generisch angelegt. Beim Importieren der Files wird keine Überprüfung vorgenommen[19].
Besonderheit der *bib*-Importfunktion sind die Cross-Referenzen. Wenn der Feldbezeichner _crossreference erkannt wird, werden dem referenzierten Eintrag die Querverweise zugewiesen und in einer Pseudospalte mit der Endung _cr abgelegt. Dabei werden die aus der Standardverarbeitung bekannten querverweisenden Felder berücksichtigt.

[18] Insgesamt wurden für die gesamte Übersetzung der Standard *bst*-Files vier Python Funktionen benötigt.

[19] Ausnahme sind Basisprüfungen der Datenbank, wie NOT NULL Constraints, maximale Länge einer Zeichenkette oder Unique Key Überprüfungen.

4.4.6 Detailbeschreibung der Umsetzung BibSQL

BibSQL wird ähnlich wie BibTeX eingesetzt. Der wesentliche Unterschied zu BibTeX ist, dass nicht mehr ein *bib*-File beziehungsweise ein *bst*-Stylefile als Input dient. Bibliographische Daten werden direkt aus der Datenbank abgefragt. Styledefinitionen werden nicht mehr aus dem *bst*-File übernommen, sondern aus dem Datenbank-Repository ermittelt. Dabei wird bei Laufzeit ein SQL-Statement generiert, das den formatierten Output (*bbl*-File) erzeugt. Wenn sich bibliographische Daten bereits in der Datenbank und der gewünschte Zitierstil im Repository befindet, kann BibSQL analog zu BibTeX verwendet werden. Der LaTeX-Compiler erzeugt ein *aux*-file, das als Basiseingabe für BibSQL dient. Wenn der Zitiersil nicht im Repository vorhanden ist, muss dieser aus dem *bst*-File in die Datenbank importiert werden. BibSQL besteht aus drei Komponenten (Abbildung 4.8):

1. Import-*bib* (Modul: bibreader)
2. *bst*-Übersetzer (Modul: bstreader)
3. BibTeX (Modul: bibtex)

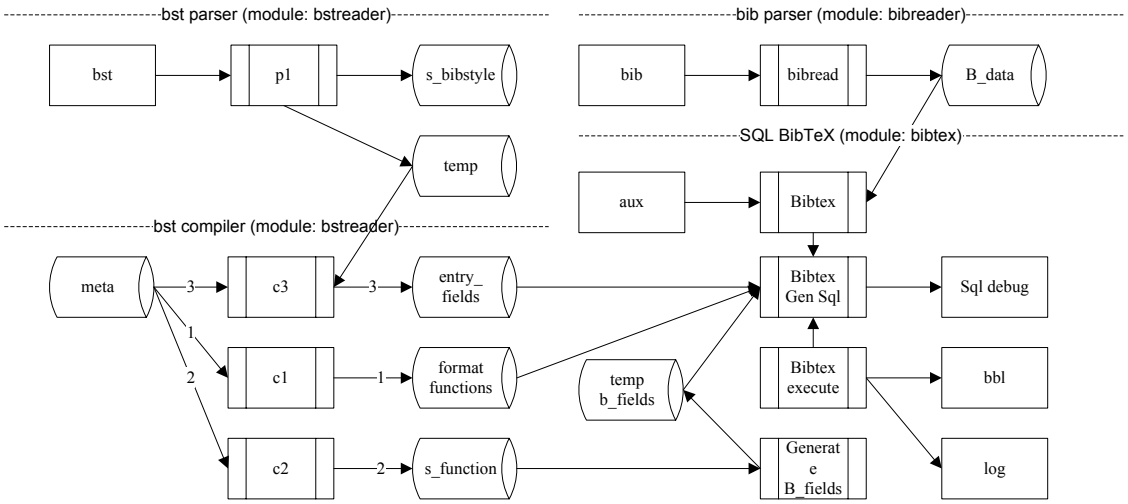

Abbildung 4.7: Funktionsweise BibSQL

Modul 1. liest und parst ein *bib*-File und speichert die Daten in der Datenbank ab.
Modul 2. liest, parst und übersetzt ein *bst*-File und legt die Daten (SQL-Code) in der Datenbank ab.
Modul 3. ist das eigentliche Anwendungsprogramm. Bedingung für die Verwendung dieses Moduls ist die Durchführung der Schritte 1. und 2.

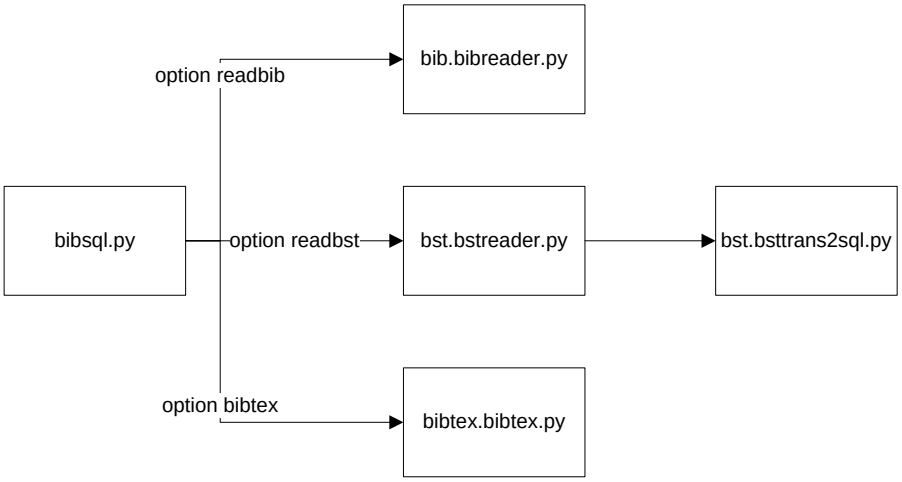

Abbildung 4.8: Überblick BibSQL-Module

Arbeitsweise des Programms bibreader:

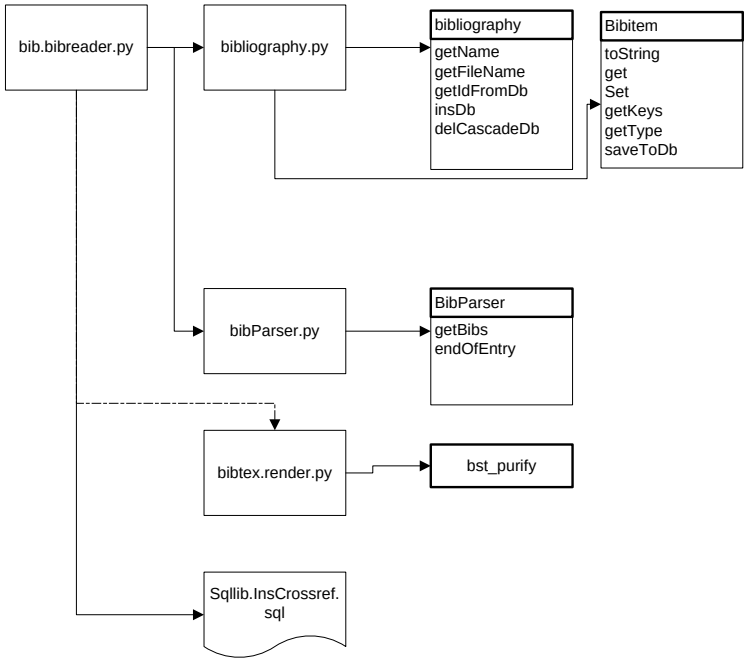

Abbildung 4.9: Modul: bibreader

Das Programm BibSQL importiert das Modul *bibreader* (Abbildung 4.9), das die Funktion *bibread* beinhaltet und den Einstieg ins Programm bildet.

Die Funktion *bibread*:
Der Name der Bibliographie ist in der Datenbank als eindeutiger Schlüssel definiert. Daher wird aus der Datenbank ermittelt, ob dieser Schlüsselwert bereits vorhanden ist. Wenn ja, werden alle Daten inklusive kaskatierender Tabellen zu diesem Schlüssel gelöscht. Dieser Fall entspricht dem Überschreiben eines bestehenden *bib*-Files.
Anschließend wird die neue Bibliographie in die Datenbank geschrieben und ein Objekt der Klasse BibParser instanziiert. Der Konstruktor parst das *bib*-File in einer Schleife über die einzelnen Zeilen. Dabei wird ein List-Objekt __bibs[] erzeugt, das alle bibliographischen Einträge beinhaltet. Es wird ein Objekt der Klasse bibitem instanziiert. In einer Schleife über alle geparsten *bib*-Einträge werden die jeweiligen *bibitems* mit der Methode saveToDB der Instanz bibItem (Klasse bibitem) in die Datenbank geschrieben.

Querverweise:
Im nächsten Schritt werden alle Querverweise („crossrefs“) ermittelt und eingefügt. Das SQL-Statement, das die Querverweise erzeugt, ist aufgrund der Größe in einer eigenen Datei ausgelagert \SQLlib\InsCrossre\InsCrossref_[db].SQL:

Listing 4.20: InsCrossref_ORACLE.SQL

```
insert
  into  BIBDB.B_FIELDS (ID
                      , ENTRY
                      , FIELD_VALUES
                      , B_ITEM_ID)
select    BIBDB.BIB_SEQ.nextval  id
         ,cross_referenced.ENTRY
         ,cross_referenced.FIELD_VALUES
         ,cross_referencing.B_ITEM_ID
   from
        (select i.ID
              , f.FIELD_VALUES
              , F.B_ITEM_ID
              , b.name
              , b.id                          bib_Id
          FROM b_bibliography  b
             , b_items         i
             , b_fields        f
         WHERE b.ID = i.b_bib_id
           AND i.ID = f.b_item_id
           and lower(F.ENTRY) = 'crossref'
        )cross_referencing
       ,(select f.entry||'_cr'           ENTRY
              , f.FIELD_VALUES            FIELD_VALUES
```

```
25              , F.B_ITEM_ID                B_ITEM_ID
26              , b.name                     name
27              , b.id                       bib_id
28              , I.CITE_KEY
29           FROM b_bibliography  b
30              , b_items         i
31              , b_fields        f
32          WHERE b.ID = i.b_bib_id
33            AND i.ID = f.b_item_id
34       )cross_referenced
35  where cross_referencing.FIELD_VALUES = cross_referenced.cite_key
36    and cross_referencing.name = cross_referenced.name
37    and cross_referenced.ENTRY in (select name
38                                     from meta_entry_fields
39                                    where function = 'crossref')
40    and cross_referencing.bib_id = cross_referenced.bib_id
41    and cross_referencing.bib_id = :bibid
```

Erzeugen der Cross-Referenzen:
Das SQL Statement ist als Insert-Select aufgebaut. Der Select-Teil besteht aus zwei Unterabfragen:
1. Der referenzierende Eintrag (Alias "cross_referencing")
2. Referenzierende Einträge (am Eintragstyp „crossref" zu erkennen)
In der Unterabfrage für den referenzierenden Eintrag wird vorerst über die gesamte Menge der Einträge abgefragt. Dem jeweiligen Feldtyp wird die Endung _cr angehängt. Die beiden Mengen der Unterabfragen werden verjoint. Dabei muss der Feldwert des referenzierenden Eintrags dem citekey des referenzierten Eintrags entsprechen.

Arbeitsweise Programm bstReader

Das Modul bstreader dient zur Übersetzung einer *bst*-Style-Datei ins BibSQL-Style-Repository. Dabei wird unter Berücksichtigung der Sprachelemente von BibTeX 0.98 das Style-File geparst und in SQL-Code übersetzt.
Das Modul bstreader setzt sich aus zwei Hauptkomponenten zusammen, dem Parser und dem Übersetzer: **Style-Datei-Parser**:
Das ausgewählte *bst*-File wird gelesen und geparst. Die geparsten Elemente werden in ein Repository geschrieben(TEMP_% Tabellen).
Auf Basis der geparsten Elemente aus den oben genannten TEMP-Tabellen und Meta-Informationen für den Übersetzer werden BibTeX-Format-Funktionen in SQL übersetzt und anschließend ins Style-Repository geschrieben.
Beide Schritte können unabhängig voneinander durchgeführt werden. In jedem Fall muss zuerst geparst werden. Durch Änderungen der Metadaten kann aber auch ein Neuübersetzen notwendig werden, ohne das *bst*-File neu zu parsen. Umgekehrt kann auch ein

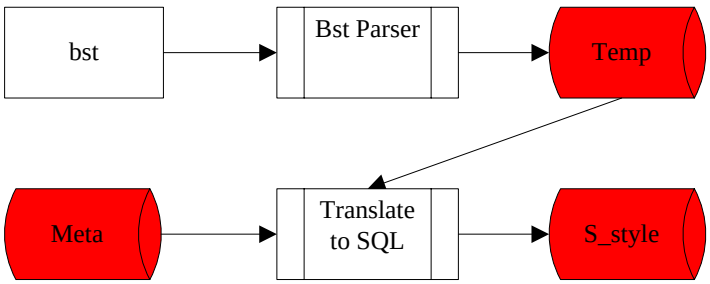

Abbildung 4.10: Dataflow bstreader

bst-File eingelesen werden, ohne es anschließend zu übersetzen. Die Elemente in den TEMP-Tabellen beziehen sich immer auf den Dateinamen (zum Beispiel plain) und bleiben solange erhalten, bis eine Datei mit demselben Namen neu gelesen wird. Das

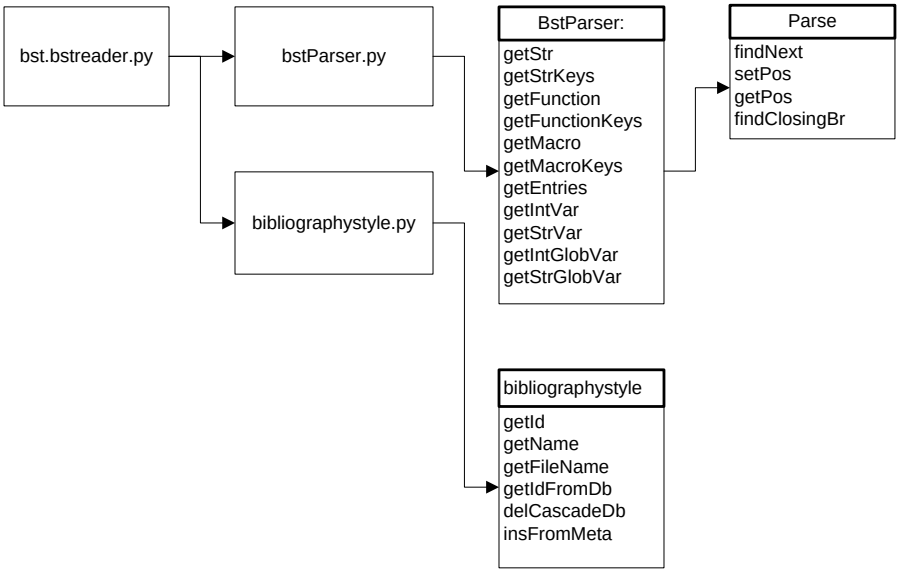

Abbildung 4.11: Modul: bstreader

Programm startet mit dem Modul bstreader. Dabei wird ein Objekt der Klasse bibliographystyle instanziiert. Der Konstruktor legt den Namen[20] des bibliographischen Stils fest und ermittelt die ID. Wenn diese ID noch nicht vorhanden ist, wird dieser Stil in der Datenbank neu angelegt.

Der Parservorgang im Detail:
Schritt 1 - Suche der Textkonstanten in Doppelhochkomma

[20] ergibt sich aus dem Dateinamen ohne Dateierweiterung

Textelemente werden im ersten Schritt geparst, damit sie im originalen Zustand erhalten bleiben. Dabei wird jedes gefundene Textelement durch einen Schlüssel ersetzt. Die Schlüssel-Textkombination wird in einem Dictionary abgelegt.
Schritt2: Kommentare
Kommentare, beginnend mit einem '%'-Symbol bis zum Ende einer Zeile, werden gelöscht.
Schritt 3: Variablen
In *bst* unterscheidet man zwischen den deklarierten, globalen Variablen und den lokalen Variablen. Sie unterscheiden sich durch ihren Gültigkeitsbereich. Die Deklaration von Entry-Variablen wird durch das Schlüsselwort Entry eingeleitet. Alle nachfolgenden Ausdrücke sind durch Whitespace getrennt, in geschwungenem Klammernpaar eingeschlossen.
4 Schritt: Funktionen und Macros
Die Deklaration von Funktionen und Macros ist syntaktisch ähnlich, sie beginnt mit dem Schlüsselwort function oder macro, gefolgt von einer geschwungen Klammer, die den Funktionsnamen einschließt und einem weiteren Klammernpaar, das den Funktionstext einschließt.

Funktionen werden in drei Kategorien eingeteilt:

- BIBTYPE - Hauptfunktion eines bibliographischen Eintrages (zum Beispiel book, article)
- FORMAT - Formatierungsfunktionen
- IGNORE – Funktionen, die später vom Compiler nicht übersetzt werden

Funktionen, die in den Tabellen META_FUNCTIONS oder META_TYPEFUNCTIONS vordefiniert werden, werden mit dem Kennzeichen IGNORE übernommen. In Folge werden sie vom Compiler nicht übersetzt, jedoch in den Funktionen verwendet.

Compiler:
Der Compiler übersetzt alle Funktionen, die vom Parser in die Metadatentabelle[21] geschrieben wurden, und nicht mit IGNORE gekennzeichnet sind, in SQL. Diese übersetzten Funktionen werden ins Repository[22] geschrieben. Außerdem werden die geparsten Entryfields und die vordefinierten Funktionen in die Tabellen S_FUNCTIONS und S_ENTRY_FIELDS übertragen. Abbildung 4.12 beschreibt den Aufbau des *bst*-Compilers.
In einer Schleife werden alle relevanten Funktionen, die vom Parser in die Tabelle

[21] Tabelle: META_FUNCTIONS
[22] Tabellen: S_FUNCTIONS, S_TYPEFUNCTIONS

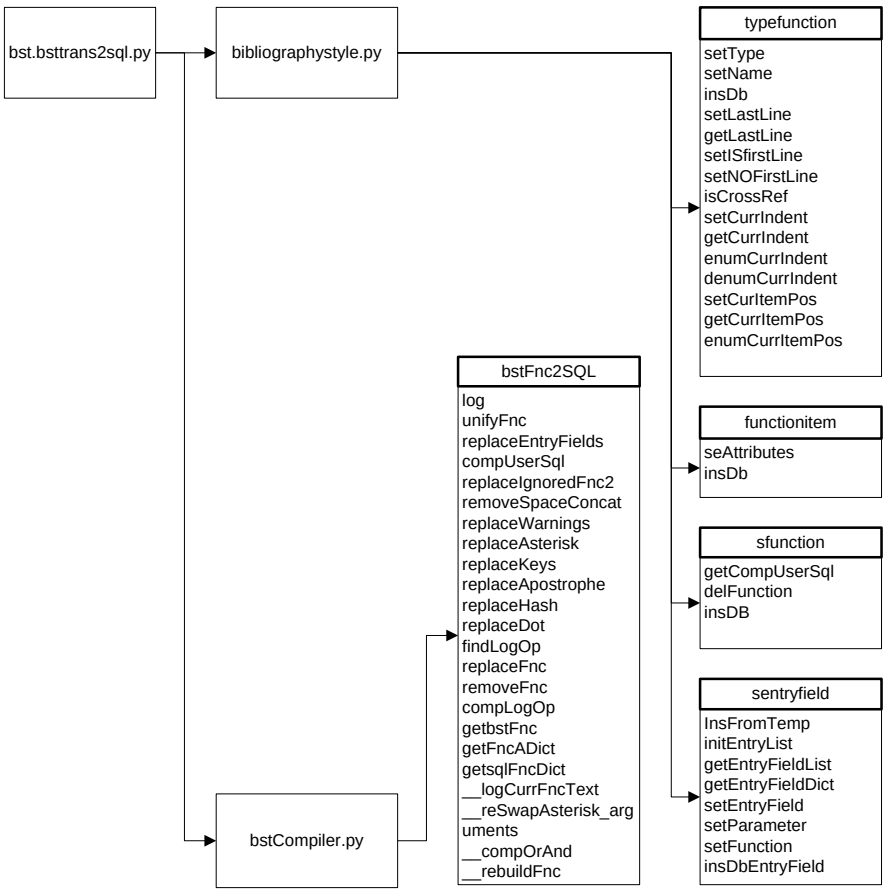

Abbildung 4.12: Modul: bstCompiler

TEMP_FUNCTIONS geschrieben wurden, einzeln abgearbeitet. Die Reihenfolge ist zu diesem Zeitpunkt nicht relevant, da der Compiler nicht überprüft, ob eine Funktion, die in einer anderen Funktion aufgerufen wird, vorhanden und gültig ist.
In jedem Schleifendurchlauf wird eine neue Instanz auf die Klasse bstCompiler.bstFnc2SQL aufgebaut. Auf das Objekt cFunc werden schrittweise die Übersetzermethoden angewandt.
Nach dem Parsen hat eine Funktion folgendes Aussehen:

Listing 4.21: Beispiel: Übersetzung der Funktion article

```
output.bibitem format.authors [109] output.check new.block format.
  title [110]
output.check new.block crossref missing$ { journal emphasize [111]
  output.check
format.vol.num.pages output format.date [112] output.check } { format.
```

```
  article.
crossref output.nonnull format.pages output } if$ new.block note
  output fin.entry
```

Alle Zeichen sind lowercase. Alle Leerräume (inklusive Zeilenumbrüche) wurden genau durch ein Leerzeichen ersetzt. Kommentare wurden entfernt. Textkonstanten wurden durch Schlüsselwerte ersetzt (zum Beispiel [109]), wie im oben angeführten Beispiel (Listing 4.21)der Funktion article.
Ein Grundsatz des Compilers ist, dass Funktionen voneinander unabhängig, hintereinander übersetzt werden. Dabei werden schrittweise Veränderungen am Text der Funktion durchgeführt. Das Ergebnis ist ein der ANSI-SQL syntaktisch entsprechender Teil einer Anweisung.
Ein Beispiel der Funktion article in SQL übersetzt ist in Abbildung 4.13) dargestellt.
Das Ergebnis der Übersetzung findet sich in der Tabelle S_FUNCTIONITEMS wieder. Das oben angeführte Beispiel bildet die Funktion articel ab. Die Spalte ITEM_POSITION gibt die Reihenfolge der einzelnen Elemente wieder. Das Attribut INDENT beschreibt die Einrückung der Code Teile[23]. Das Attribut LINE_ACTION gibt an, wie die einzelnen Elemente miteinander verbunden werden, in den meisten Fällen ist das ein Konkatinierungssymbol.
Wie Funktionen schlussendlich zu einem ausführbaren, syntaktisch richtigen SQL-Statement zusammengefügt werden, wird im Kapitel Module bibtex beschrieben.
Für die Übersetzung einiger Funktionen werden reguläre Ausdrücke dynamisch erzeugt. Dabei wird vom Muster a (FUNCTION_BST) auf Muster b (FUNCTION_SQL) übersetzt.
In diesem Schritt wird Muster a und b durch Ersetzungen und Konkatenationen soweit bearbeitet, dass sie syntaktisch Python‘s regulären Ausdrücken entsprechen. Das Zielmuster wird mit einem umschließenden [lqs(.+?)lqs] versehen. Im zweiten Schritt werden diese Funktionen je nach Typ auf [boo(.+?)] oder [SQL(.+?)] umgewandelt. Das dient zur Unterscheidung zwischen normalen SQL-Funktionen und Funktionen, die eine Wahrheitsprüfung darstellen.

Logische Operationen übersetzen
Dieser Schritt ist der komplexeste Schritt des Compilers. Dabei müssen Funktionen und deren Verschachtelungen im Zusammenhang mit der if$-Anweisung identifiziert werden, and, or und die zugehörigen logischen Bedingungen gefunden werden und in ein SQL CASE WHEN THEN ELSE END Konstrukt umgewandelt werden. In einem weiteren Schritt werden die and or if-Verschachtelungen aufgelöst und die zugehörigen logischen Bedingungen sowie der Anweisungsblock für Then- beziehungsweise für den Else-Zweig ermittelt.

[23] ausschließlich eine Formatierungsmaßnahme

Eine weitere Besonderheit sind benutzerdefinierte SQL–Python-Funktionen. Diese Funktionen werden durch ein Python-Programm repräsentiert. Beim Übersetzen von *bst* in SQL wird die entsprechende *bst*-Funktion zusammen mit den zugehörigen Parametern in eine Pseudo-Spalte übersetzt. Der Spaltenname setzt sich aus den verwendeten Variablen und dem Funktionsnamen zusammen. Hinter diesem Spaltennamen verbirgt sich eine Python-Routine, die bei Laufzeit durchgeführt wird, und den errechneten Wert in eine temporäre Tabelle einträgt. Analog zu den „echten“ Feldeinträgen in der Tabelle B_FIELDS befinden sich diese Einträge in der Tabelle TEMP_B_FIELDS. In Folge wird mit einem Union-Select über diesen künstlich generierten Spaltennamen abgefragt. Abschließend werden die Eigenschaften des Objektes SQLItem in die Datenbank eingefügt.
Die Funktionselemente werden zeilenweise in die Tabelle S_FUNKTIONITEMS (Abbildung: 4.13) eingetragen. Dabei werden neben dem Funktionstext eine fortlaufende Zeilennummer, der errechnete Indent und die Spalte LINE_ACTION befüllt. Letzteres bestimmt, wie im endgültigen SQL-Statement die einzelnen Elemente miteinander verbunden werden[24].

Arbeitsweise des Moduls *bibtex*

Das Modul *bibtex.py* ist das eigentliche Programm, das BibTeX ablöst. Es wird im Arbeitsablauf in gleicher Weise wie LaTeX und BibTeX eingesetzt. Nachdem LaTeX eine *aux*-Datei erzeugt hat, wird *bibtex.py* aufgerufen, das eine *bbl*-Datei erzeugt.
***aux*-Datei lesen:**
Ein Objekt aux der Klasse AuxReader wird instanziiert. Der Konstruktor der Klasse AuxReader parst die *aux*-Datei. Dabei werden die Zitierschlüssel (cite keys) ausgelesen und in ein Listenobjekt geschrieben. Im weiteren Schritt werden Informationen bezüglich \bibdata und \bibstyle geparst und in Variablen gespeichert.
Temp Felder schreiben:
In diesem Schritt werden temporäre Feldinhalte mittels Python-Prozeduren ermittelt und anschließend in die Tabelle TEMP_B_FIELDS geschrieben. Dafür wird die Funktion insBFields aus dem Modul TempPySQL.py ausgeführt. Rückgabewert dieser Funktion ist eine Session-Nummer. Anhand dieser eindeutigen Nummer können diese Werte nachträglich wieder gelöscht werden (Funktion delBFields()).
***bbl*-Datei erzeugen:**
In diesem Schritt geht es darum, ein SQL-Statement zu erzeugen, das formatierte und sortierte Ergebnisse in eine *bbl*-Datei einfügt. Dieses Statement muss bei Laufzeit generiert werden, da es ein Produkt aus bibliographischem Stil und bibliographischen Einträgen ist.

[24] In der SQL-Syntax haben weder Einrückungen, noch Zeilenumbrüche (abgesehen von Kommentaren) Bedeutung.

ID	ITEM_POSITION	INDENT	LINE_ACTION	SQL_TEXT
407631	0	0		output_bibitem
407632	10	0	||	format_authors
407633	20	0	||	'%{output}'
407634	30	0	||	'%{newblock}'
407635	40	0	||	format_title
407636	50	0	||	'%{output}'
407637	60	0	||	'%{newblock}'
407638	70	0	||	CASE WHEN "crossref" is NULL THEN
407639	80	4		'{\em}'|| "journal" ||'\/}'
407640	90	4	||	'%{output}'
407641	100	4	||	format_vol_num_pages
407642	110	4	||	'%{output}'
407643	120	4	||	format_date
407644	130	4	||	'%{output}'
407645	140	0		ELSE
407646	150	4		format_article_crossref
407647	160	4	||	'%{output}'
407648	170	4	||	format_pages
407649	180	4	||	'%{output}'
407650	190	0		END
407651	200	0	||	'%{newblock}'
407652	210	0	||	"note"
407653	220	0	||	'%{output}'
407654	230	0	||	fin_entry

Abbildung 4.13: Table S_FUNCTIONITEMS

Dieses SQL-Statement besteht aus zwei Hauptteilen:
1. Dem Select-Teil (bibliographischer Stil)
Dieser Teil setzt sich aus Funktionen der Tabelle S_FUNCTION_ITEMS zusammen.
2. Dem From-Teil (bibliographische Einträge)
Der Umstand, dass BibTeX eine generische bibliographische Datenbank verwendet, macht es notwendig, den From-Teil vom Statement bei Laufzeit zu generieren.

Generieren des Select-Teils (Listing 4.22)

Listing 4.22: GenSelectSQLOracle.SQL

```
1  SELECT
2     nvl(ENTRYTYPE,' ') ENTRYTYPE
3    ,nvl(SQL_TEXT,' ')  text
4  from(
5    select case when type_bt = 'BIBTYPE' then 10
6                when type_bt = 'LABEL' then 20
7                when type_bt = 'SORT' then 30
8                end  pos,
9           CASE
10             WHEN BF_POS + NVL (F1_POS, 0) + NVL (F2_POS, 0) = 0
11              AND type_bt = 'BIBTYPE'
```

```
12                 THEN    'WHEN ITEMTYPE = '''
13                      || FNC
14                      || ''' THEN'
15                      || CHR (10)
16           END ENTRYTYPE,
17           LINE_ACTION || LPAD (' ', INDENT, ' ') || SQL_TEXT || COMM SQL_TEXT,
18           FNC, BF_POS, F1_POS, F2_POS, BST
19      FROM (SELECT DECODE (FF.F1_POS,
20                           0, BT.LINE_ACTION,
21                           NVL (FF.LINE_ACTION, BT.LINE_ACTION)
22                          ) LINE_ACTION,
23                   NVL (FF.SQL_TEXT, BT.SQL_TEXT) SQL_TEXT,
24                   NVL (FF.INDENT, 0) + NVL (BT.INDENT, 0) INDENT,
25                   FF.COMM COMM, BT.FNC, BT.BF_POS, FF.F1_POS, FF.F2_POS,
26                   BT.BST,type_bt
27              FROM (SELECT BT.BST BST, BT.FNC FNC,bt.type type_bt,
28                           NVL (BT2.INDENT, 0) + NVL (BT.INDENT, 0) INDENT,
29                           DECODE (BT2.ITEM_POSITION,
30                                   0, BT.LINE_ACTION,
31                                   NVL (BT2.LINE_ACTION, BT.LINE_ACTION)
32                                  ) LINE_ACTION,
33                           NVL (BT2.SQL_TEXT, BT.SQL_TEXT) SQL_TEXT,
34                           NVL (BT2.ITEM_POSITION, BT.ITEM_POSITION) BF_POS
35                      FROM (SELECT T.S_BST_ID BST,
36                                   REPLACE (T.NAME, '.', '_') FNC, F.INDENT,
37                                   NVL (F.LINE_ACTION, '  ') LINE_ACTION,
38                                   F.SQL_TEXT, F.ITEM_POSITION, T.TYPE
39                              FROM S_FUNCTIONITEMS F
40                                 , (SELECT ID, TYPE, NAME, BST_ID S_BST_ID
41                                      from S_INCL_TYPEFUNCTIONS_V1) T
42                             WHERE F.S_TYPEF_ID = T.ID
43                                   AND T.TYPE IN ('BIBTYPE','SORT','LABEL')) BT,
44                           (SELECT T.S_BST_ID BST,
45                                   REPLACE (T.NAME, '.', '_') FNC, F.INDENT,
46                                   NVL (F.LINE_ACTION, '  ') LINE_ACTION,
47                                   F.SQL_TEXT, F.ITEM_POSITION
48                              FROM S_FUNCTIONITEMS F
49                                 , (SELECT ID, TYPE, NAME, BST_ID S_BST_ID
50                                      from S_INCL_TYPEFUNCTIONS_V1) T
51                             WHERE F.S_TYPEF_ID = T.ID AND T.TYPE in ('SORT', 'BIBTYPE','LABEL')) BT2
52                     WHERE BT.BST = BT2.BST(+) AND BT.SQL_TEXT = BT2.FNC(+)) BT,
53                   (SELECT FF.BST BST, FF.FNC FNC,
54                           NVL (FF2.INDENT, 0) + NVL (FF.INDENT, 0) INDENT,
55                           DECODE (FF2.ITEM_POSITION,
56                                   0, FF.LINE_ACTION,
57                                   NVL (FF2.LINE_ACTION, FF.LINE_ACTION)
58                                  ) LINE_ACTION,
59                           NVL (FF2.SQL_TEXT, FF.SQL_TEXT) SQL_TEXT,
60                           DECODE
61                              (FF.ITEM_POSITION,
62                               0, '          --fieldformat        : ' || FF.FNC,
63                               DECODE
64                                    (FF2.ITEM_POSITION,
65                                     0,  '              --fieldformat recurs2: '
66                                      || FF2.FNC
67                                    )
68                              ) COMM,
69                           FF.ITEM_POSITION F1_POS, FF2.ITEM_POSITION F2_POS
70                      FROM (SELECT T.S_BST_ID BST,
71                                   REPLACE (T.NAME, '.', '_') FNC, F.INDENT,
72                                   NVL (F.LINE_ACTION, '  ') LINE_ACTION,
73                                   F.SQL_TEXT, F.ITEM_POSITION
74                              FROM S_FUNCTIONITEMS F
75                                 , (SELECT ID, TYPE, NAME, BST_ID S_BST_ID
76                                      from S_INCL_TYPEFUNCTIONS_V1) T
77                             WHERE F.S_TYPEF_ID = T.ID
78                               --AND T.TYPE IN ('FIELDFORMAT')
79                               ) FF,
80                           (SELECT T.S_BST_ID BST,
81                                   REPLACE (T.NAME, '.', '_') FNC, F.INDENT,
82                                   NVL (F.LINE_ACTION, '  ') LINE_ACTION,
83                                   F.SQL_TEXT, F.ITEM_POSITION
84                              FROM S_FUNCTIONITEMS F
85                                 , (SELECT ID, TYPE, NAME, BST_ID S_BST_ID
86                                      from S_INCL_TYPEFUNCTIONS_V1) T
87                             WHERE F.S_TYPEF_ID = T.ID
88                              -- AND T.TYPE = 'FIELDFORMAT'
89                               ) FF2
90                     WHERE FF.BST = FF2.BST(+) AND FF.SQL_TEXT = FF2.FNC(+)) FF
91             WHERE BT.BST = FF.BST(+) AND BT.SQL_TEXT = FF.FNC(+))
92       where bst = :BST_ID
93 union all
94   SELECT 5 POS  , 'SELECT CASE '    ENTRYTYPE, NULL SQL_TEXT, NULL  FNC, NULL  BF_POS, NULL  F1_POS, NULL F2_POS, :
```

```
    BST_ID BST from dual
union all
  SELECT 15 POS , 'end ENTRYTEXT, ' ENTRYTYPE, NULL SQL_TEXT, NULL  FNC, NULL  BF_POS, NULL  F1_POS, NULL F2_POS, :
    BST_ID BST from dual
union all
  SELECT 25 POS , 'LABEL, '         ENTRYTYPE, NULL SQL_TEXT, NULL  FNC, NULL  BF_POS, NULL  F1_POS, NULL F2_POS, :
    BST_ID BST from dual
union all
  SELECT 35 POS , 'SORT from  '     ENTRYTYPE, NULL SQL_TEXT, NULL  FNC, NULL  BF_POS, NULL  F1_POS, NULL F2_POS, :
    BST_ID BST from dual
)
    ORDER BY POS ASC
      , BST ASC NULLS LAST
      , FNC ASC NULLS LAST
      , BF_POS ASC NULLS LAST
      , F1_POS ASC NULLS LAST
      , F2_POS ASC NULLS LAST
```

Aufgabe dieses Generators ist es, Funktionen des ausgewählten bibliographischen Stils aus dem Datenbankrepository zu ermitteln.

Dabei werden drei Prinzipien berücksichtigt:
1. Prinzip der Strukturiertheit
Funktionen müssen miteinander verschachtelt werden können. Verschachtelt heißt, Funktionen die in Funktionen aufgerufen werden, müssen an dieser Stelle eingefügt werden. Das passiert auf maximal drei Ebenen.
2. Prinzip der Hierarchie
Es wird von den BIBTYPE-Funktionen ausgegangen. Diese werden in einem CASE WHEN THEN ELSE END Konstrukt untereinandergereiht:

Listing 4.23: SQL-BIBTYPE-Funktionen

```
SELECT
  CASE
   WHEN ITEMTYPE = 'article' THEN
     function{article}
   WHEN ITEMTYPE = 'booklet' THEN
     function{booklet}
   WHEN ITEMTYPE = 'conference' THEN
     function{conference}
   WHEN ITEMTYPE = 'inbook' THEN
     function{inbook}
   WHEN ITEMTYPE = '.....' THEN
     function{.....}
   ELSE
     function{misc}
   END CASE     ENTRYTEXT
```

Dieser Anweisungsblock endet mit dem Spalten-Alias ENTRYTEXT, der die Spalte für den formatierten Text darstellt.
Neben BIBTYPE-Funktionen gibt es auch Funktionen des Typs SORT und LABEL. Diese

stehen auch an erster Stelle der Hierarchie und sind Ausgangsfunktion für die Spalten LABEL und SORT im Ziel-Statement.
Alle anderen Funktionen sind vom Typ Format und werden gleichwertig behandelt. Sie beginnen strukturell in der zweiten Ebene. Diese Funktionen können wiederum Funktionen jedes beliebigen Typs aufrufen. BIBTYPE-, LABEL- und SORT-Funktionen können in jeder Ebene aufgerufen werden[25].
Technisch wird diese Verschachtelung durch Outer-Joins erreicht. Dabei werden eine Reihe von Unterabfragen, die die Formatierung und Einrückung übernehmen, miteinander verjoint. Funktionsname und Funktionsinhalt werden spaltenweise nebeneinander gestellt und durch „NULL Value“-Logik und Sortierung zu einem Statement zusammengesetzt.
An der Funktionalität und der SQL-Syntax der Funktion aus dem Datenbankrepository wird in diesem Schritt nichts geändert. Funktionen werden lediglich zusammengesetzt und die Einrückung formatiert.

Ein Beispiel der generierten SQL-Abfrage für die Funktion article für den Style alpha wird im Listing 4.24 dargestellt.

Listing 4.24: SQL_alpha.SQL (Funktion:article)

```
SELECT CASE
WHEN ITEMTYPE = 'article' THEN
  '\n'          --fieldformat        : output_bibitem
 ||'\bibitem['
 ||label
 ||']{'
 ||CITEKEY
 ||'}'
 ||'\n'
 ||''
 ||'%{before_all}'
 ||'%{output_state}'
 ||CASE WHEN  "author" is NULL   THEN          --fieldformat        : format_authors
       ''
   ELSE
       "author_FORMAT_NAMES"
   END
 ||'%{output} '
 ||'%{newblock}'
 ||CASE WHEN  "title" is NULL   THEN          --fieldformat        : format_title
       ''
   ELSE
       INITCAP("title")
   END
 ||'%{output} '
 ||'%{newblock}'
 ||CASE WHEN  "crossref" is NULL   THEN
       '{\em}'|| "journal" ||'\/}'
 ||    '%{output} '
 ||    "volume"          --fieldformat        : format_vol_num_pages
 ||    CASE WHEN  "number" is NULL   THEN
           NULL
       ELSE
           '('
 ||        "number"
 ||        ')'
       END
 ||    CASE WHEN  "pages" is not NULL   THEN
           CASE WHEN  "volume" is NULL and "number" is NULL THEN
               CASE WHEN  "pages" is NULL   THEN             --fieldformat recurs2: format_pages
                   ''
```

[25]Die Funktion LABEL entpricht der *bst*-Funktion calc.label, die Funktion SORT der Funktion presort.

```
42                ELSE
43                    CASE WHEN  TO_CHAR(LENGTH("pages")-LENGTH(REPLACE(REPLACE("pages",'+',''),'-',''
                  )))>0  THEN
44                        'pages'
45  ||                     CASE WHEN LENGTH(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE("pages",'--',
                  '"xxx"'),'"xxx"-','"xxx""yy"'),'-','--'),'"xxx"','--'),'"yy"','-'))<3 THEN '~' ELSE ' ' END || REPLACE(REPLACE(REPLACE(REPLACE(
                  REPLACE("pages",'--','"xxx"'),'"xxx"-','"xxx""yy"'),'-','--'),'"xxx"','--'),'"yy"','-')
46                    ELSE
47                        'page'
48  ||                     CASE WHEN LENGTH("pages")<3 THEN '~' ELSE ' ' END || "pages"
49                    END
50                END
51            ELSE
52                ':'
53  ||            REPLACE(REPLACE(REPLACE(REPLACE(REPLACE("pages",'--','"xxx"'),'"xxx"-','"xxx""yy"'),'-','--'),'"xxx"
                  ','--'),'"yy"','-')
54            END
55        END
56  ||    '%{output} '
57  ||    CASE WHEN  "year" is NULL   THEN          --fieldformat        : format_date
58            CASE WHEN  "month" is NULL   THEN
59                ''
60            ELSE
61                '%{warning$'
62  ||            'there''s a month but no year in '
63  ||            CITEKEY
64  ||            '}'
65  ||            "month"
66            END
67        ELSE
68            CASE WHEN  "month" is NULL   THEN
69                "year"
70            ELSE
71                "month"
72  ||            ' '
73  ||            "year"
74            END
75        END
76  ||    '%{output} '
77    ELSE
78        CASE WHEN  "key" is NULL   THEN          --fieldformat        : format_article_crossref
79            CASE WHEN  "journal" is NULL   THEN
80                '%{warning$'
81  ||            'need key or journal for '
82  ||            CITEKEY
83  ||            ' to crossref '
84  ||            "crossref"
85  ||            '}'
86  ||            ''
87            ELSE
88                'In {\em '
89  ||            "journal"
90  ||            '\/}'
91            END
92        ELSE
93            'In '
94  ||        "key"
95        END
96  ||    ' \cite{'
97  ||    "crossref"
98  ||    '}'
99  ||    '%{output} '
100 ||    CASE WHEN  "pages" is NULL   THEN          --fieldformat        : format_pages
101           ''
102       ELSE
103           CASE WHEN  TO_CHAR(LENGTH("pages")-LENGTH(REPLACE(REPLACE("pages",'+',''),'-',''
                  )))>0  THEN
104               'pages'
105 ||             CASE WHEN LENGTH(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE("pages",'--','"xxx"'),'"xxx"-','"xxx""yy"'
                  ),'-','--'),'"xxx"','--'),'"yy"','-'))<3 THEN '~' ELSE ' ' END || REPLACE(REPLACE(REPLACE(REPLACE(REPLACE("pages
                  ",'--','"xxx"'),'"xxx"-','"xxx""yy"'),'-','--'),'"xxx"','--'),'"yy"','-')
106           ELSE
107               'page'
108 ||            CASE WHEN LENGTH("pages")<3 THEN '~' ELSE ' ' END || "pages"
109           END
110       END
111 ||    '%{output} '
112   END
113 ||'%{newblock}'
114 ||"note"
115 ||'%{output} '
116 ||'%{add_period}'          --fieldformat        : fin_entry
117 ||'\n'

119 ......
```

```
120 END CASE ENTRYTEXT,
121 .......
```

Durch die Sprache SQL ist die Tiefe der Ebenen vorgegeben. In einigen SQL-Spracherweiterungen gibt es die Möglichkeit der hierarchischen Abfrage. Zum Beispiel in Oracle-SQL ist es durch die „Connect by“-Syntax möglich, beliebig viele hierarchische Ebenen abzufragen. Hierarchische Abfragen würden eine beliebige Rekursionstiefe zulassen. Derartige Spracherweiterungen wurden nicht verwendet, um die Portabilität zu gewährleisten.

3. Prinzip der Modularität

Im Gegensatz zu BibTeX hat BibSQL den Anspruch modular zu sein. Funktionen, die von mehreren Stilen verwendet werden, müssen nicht immer wieder neu implementiert werden, sondern werden einfach aus einer Bibliothek eingebunden.

Diese Verbindung ist in der Tabelle S_BIBSTYLE_REF definiert. In dieser Tabelle werden einzelne Stile miteinander verknüpft. Wenn ein Stil bereits übersetzt wurde, kann er als Input für weitere Stile verwendet werden. Funktionen die denselben Inhalt haben, müssen nicht neu übersetzt werden[26].

Die technische Umsetzung erfolgt durch eine View. Funktionen werden durch die Tabellen S_TYPEFUNCTIONS und der Detailtabelle S_FUNCTIONITEMS, die den Funktionstext beinhaltet, verknüpft. Die Tabelle S_TYPEFUNCTIONS wird durch die View S_INCL_TYPEFUNCTIONS_V1 (Listing 4.25) ersetzt, die durch Verjoinung der einzelnen Stile einen erweiterten Stil bildet. Funktionen des verknüpften Stils werden beim Zusammenbauen der Funktionen mitberücksichtigt.

Listing 4.25: VIEW S_INCL_TYPEFUNCTIONS_V1

```
   CREATE OR REPLACE FORCE VIEW bibdb.s_incl_typefunctions_v1 (ID,
                                                              NAME,
                                                              TYPE,
                                                              bst_id
                                                             )
AS
   SELECT tf.ID ID, tf.NAME, tf.TYPE, REF.ma bst_id
     FROM s_typefunctions tf,
          (SELECT ID ma, ID REF, 0 pos
             FROM s_bibliographystyles
           UNION
           SELECT ma.ID, NVL (rd.s_bst_detail_id, ma.ID),
                  NVL (rd.POSITION, 0) pso
             FROM s_bibliographystyles ma, s_bibstyles_ref rd
            WHERE ma.ID = rd.s_bst_master_id(+)) REF
    WHERE REF.REF = tf.s_bst_id
      AND (tf.NAME, tf.TYPE, pos, REF.ma) IN (
```

[26] Ein Großteil der Funktionen der Standardstile wie alpha und plain gleichen sich zu 100%.

```
            SELECT   tf.NAME, tf.TYPE, MIN (pos), REF.ma bst
                FROM s_typefunctions tf,
                     (SELECT ID ma, ID REF, 0 pos
                        FROM s_bibliographystyles
                      UNION
                      SELECT ma.ID, NVL (rd.s_bst_detail_id, ma.ID),
                             NVL (rd.POSITION, 0) pso
                        FROM s_bibliographystyles ma, s_bibstyles_ref
                           rd
                       WHERE ma.ID = rd.s_bst_master_id(+)) REF
               WHERE REF.REF = tf.s_bst_id
            GROUP BY tf.NAME, tf.TYPE, REF.ma);
```

Diese Abfrage ist gleich strukturiert wie die Tabelle S_TYPEFUNCTIONS, beinhaltet aber die Gesamtmenge der Funktionen aller verknüpften Stile. Das Resultat dieser View gewährleistet bereits die Eindeutigkeit einer Funktion. Bei Mehrfachvorkommen von Funktionen wird immer die Originalfunktion eines Stiles bevorzugt.

Generieren des From-Teiles

Das Prinzip des Generators ist es ein SQL-Statement zu erzeugen, das die generische vertikale Datenstruktur von BibTEX horizontal denormalisiert. Das passiert zum Zeitpunkt der Verbindung von Dateninhalten und dem Style, da der bibliographische Stil die verwendeten Spalten (Entryfields) vorgibt.

Der Generator besteht aus einem Union Select, das die einzelnen Komponenten des Statements in mehreren Unterabfragen beinhaltet. Jedes einzelne Select hat eine TEXT-Spalte und eine SORT-Spalte. Die TEXT-Spalte beinhaltet den Text des zu generierenden Statements, die SORT-Spalte sorgt für die richtige Reihenfolge bei der Ausgabe. Dabei werden die Tabellen DUAL und die View S_INCL_ENTRY_FIELDS_V1 abgefragt. S_INCL_ENTRY_FIELDS_V1 (Listing 4.26) erzeugt eine globale Sicht auf die Tabelle S_ENTRY_FIELDS, das heißt verknüpfte Stile werden mitberücksichtigt.

Listing 4.26: VIEW s_incl_entry_fields_v1

```
CREATE OR REPLACE FORCE VIEW bibdb.s_incl_entry_fields_v1 (NAME,
                                                           FUNCTION,
                                                           bst_id
                                                          )
AS
   SELECT   ef.NAME, ef.FUNCTION, REF.ma bst_id
       FROM s_entry_fields ef,
            (SELECT ID ma, ID REF, 0 pos
               FROM s_bibliographystyles
             UNION
             SELECT ma.ID, NVL (rd.s_bst_detail_id, ma.ID),
                    NVL (rd.POSITION, 0) pso
               FROM s_bibliographystyles ma, s_bibstyles_ref rd
              WHERE ma.ID = rd.s_bst_master_id(+)) REF
```

```
      WHERE ef.s_bst_id = REF.REF
   GROUP BY ef.NAME, ef.FUNCTION, REF.ma;
```

Ein Beispiel des generierten FROM-Teil (horizontale Denormalisierung) für den Style alpha (id:364838) der Bibliographien mit den Id's ('366186','399095') und den Einträgen: ('whole-set','whole-set','whole-collection','whole-proceedings','book-full') wird im Listing 4.27 dargestellt.

Listing 4.27: SQL_alpha.SQL (FROM-Teil)

```
(select  B.ID BIB_ID,b.NAME BIB_NAME, bi.ITEM_TYPE ITEMTYPE, bi.CITE_KEY CITEKEY, '%{label}' label
       ,"address"
       ,"author"
       ,"author_cr"
       ,"author_FORMAT_LAB_NAMES"
       ,"author_FORMAT_NAMES"
       ,"author_SORT_FORMAT_NAMES"
       ,"booktitle"
       ,"booktitle_cr"
       ,"chapter"
       ,"crossref"
       ,"edition"
       ,"editor"
       ,"editor_cr"
       ,"editor_FORMAT_CROSSREF"
       ,"editor_FORMAT_LAB_NAMES"
       ,"editor_FORMAT_NAMES"
       ,"editor_SORT_FORMAT_NAMES"
       ,"howpublished"
       ,"institution"
       ,"journal"
       ,"journal_cr"
       ,"key"
       ,"key_cr"
       ,"month"
       ,"note"
       ,"number"
       ,"organization"
       ,"pages"
       ,"publisher"
       ,"school"
       ,"series"
       ,"series_cr"
       ,"title"
       ,"title_cr"
       ,"type"
       ,"volume"
       ,"volume_cr"
       ,"year"
       ,"year_pu"
       ,nvl((length("author")-length(replace("author",' and ')))/5+1,0) "author_NUM_NAMES"
       ,nvl((length("editor")-length(replace("editor",' and ')))/5+1,0) "editor_NUM_NAMES"
  from BIBDB.B_ITEMS    bi,BIBDB.B_BIBLIOGRAPHY  b,BIBDB.S_BIBLIOGRAPHYSTYLES sty
      ,(select  EF.S_BST_ID
               ,max("address".field_values)                        "address"
               ,max("author_cr".field_values)                      "author_cr"
               ,max("author".field_values)                         "author"
               ,max("author_FORMAT_LAB_NAMES".field_values)        "author_FORMAT_LAB_NAMES"
               ,max("author_FORMAT_NAMES".field_values)            "author_FORMAT_NAMES"
               ,max("author_SORT_FORMAT_NAMES".field_values)       "author_SORT_FORMAT_NAMES"
               ,max("booktitle_cr".field_values)                   "booktitle_cr"
               ,max("booktitle".field_values)                      "booktitle"
               ,max("chapter".field_values)                        "chapter"
               ,max("crossref".field_values)                       "crossref"
               ,max("edition".field_values)                        "edition"
               ,max("editor_cr".field_values)                      "editor_cr"
               ,max("editor".field_values)                         "editor"
               ,max("editor_FORMAT_CROSSREF".field_values)         "editor_FORMAT_CROSSREF"
               ,max("editor_FORMAT_LAB_NAMES".field_values)        "editor_FORMAT_LAB_NAMES"
               ,max("editor_FORMAT_NAMES".field_values)            "editor_FORMAT_NAMES"
               ,max("editor_SORT_FORMAT_NAMES".field_values)       "editor_SORT_FORMAT_NAMES"
               ,max("howpublished".field_values)                   "howpublished"
               ,max("institution".field_values)                    "institution"
               ,max("journal_cr".field_values)                     "journal_cr"
               ,max("journal".field_values)                        "journal"
               ,max("key_cr".field_values)                         "key_cr"
```

```
                ,max("key".field_values)                             "key"
                ,max("month".field_values)                           "month"
                ,max("note".field_values)                            "note"
                ,max("number".field_values)                          "number"
                ,max("organization".field_values)                    "organization"
                ,max("pages".field_values)                           "pages"
                ,max("publisher".field_values)                       "publisher"
                ,max("school".field_values)                          "school"
                ,max("series_cr".field_values)                       "series_cr"
                ,max("series".field_values)                          "series"
                ,max("title_cr".field_values)                        "title_cr"
                ,max("title".field_values)                           "title"
                ,max("type".field_values)                            "type"
                ,max("volume_cr".field_values)                       "volume_cr"
                ,max("volume".field_values)                          "volume"
                ,max("year".field_values)                            "year"
                ,max("year_pu".field_values)                         "year_pu"
                ,nvl("address".B_ITEM_ID
                ,nvl("author".B_ITEM_ID
                ,nvl("author_cr".B_ITEM_ID
                ,nvl("author_FORMAT_LAB_NAMES".B_ITEM_ID
                ,nvl("author_FORMAT_NAMES".B_ITEM_ID
                ,nvl("author_SORT_FORMAT_NAMES".B_ITEM_ID
                ,nvl("booktitle".B_ITEM_ID
                ,nvl("booktitle_cr".B_ITEM_ID
                ,nvl("chapter".B_ITEM_ID
                ,nvl("crossref".B_ITEM_ID
                ,nvl("edition".B_ITEM_ID
                ,nvl("editor".B_ITEM_ID
                ,nvl("editor_cr".B_ITEM_ID
                ,nvl("editor_FORMAT_CROSSREF".B_ITEM_ID
                ,nvl("editor_FORMAT_LAB_NAMES".B_ITEM_ID
                ,nvl("editor_FORMAT_NAMES".B_ITEM_ID
                ,nvl("editor_SORT_FORMAT_NAMES".B_ITEM_ID
                ,nvl("howpublished".B_ITEM_ID
                ,nvl("institution".B_ITEM_ID
                ,nvl("journal".B_ITEM_ID
                ,nvl("journal_cr".B_ITEM_ID
                ,nvl("key".B_ITEM_ID
                ,nvl("key_cr".B_ITEM_ID
                ,nvl("month".B_ITEM_ID
                ,nvl("note".B_ITEM_ID
                ,nvl("number".B_ITEM_ID
                ,nvl("organization".B_ITEM_ID
                ,nvl("pages".B_ITEM_ID
                ,nvl("publisher".B_ITEM_ID
                ,nvl("school".B_ITEM_ID
                ,nvl("series".B_ITEM_ID
                ,nvl("series_cr".B_ITEM_ID
                ,nvl("title".B_ITEM_ID
                ,nvl("title_cr".B_ITEM_ID
                ,nvl("type".B_ITEM_ID
                ,nvl("volume".B_ITEM_ID
                ,nvl("volume_cr".B_ITEM_ID
                ,nvl("year".B_ITEM_ID
                ,nvl("year_pu".B_ITEM_ID
                             ,NULL
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
```

```
150                                 )
151                                 )
152                                 )
153                                 )
154                                 )
155                                 )
156                                 )
157                                 )
158                                 )
159                                 )
160                                 )
161                                 )
162                                 )
163                             id
164          from  BIBDB.S_ENTRY_FIELDS  ef
165               left outer Join BIBDB.b_fields "address" on  ef.name = "address".entry and "address".entry = 'address
                    '
166               left outer Join BIBDB.b_fields "author" on  ef.name = "author".entry and "author".entry = 'author'
167               left outer Join BIBDB.b_fields "author_cr" on  ef.name = "author_cr".entry and "author_cr".entry = '
                    author_cr'
168               left outer Join BIBDB.b_fields "booktitle" on  ef.name = "booktitle".entry and "booktitle".entry = '
                    booktitle'
169               left outer Join BIBDB.b_fields "booktitle_cr" on  ef.name = "booktitle_cr".entry and "booktitle_cr".
                    entry = 'booktitle_cr'
170               left outer Join BIBDB.b_fields "chapter" on  ef.name = "chapter".entry and "chapter".entry = 'chapter
                    '
171               left outer Join BIBDB.b_fields "crossref" on  ef.name = "crossref".entry and "crossref".entry = '
                    crossref'
172               left outer Join BIBDB.b_fields "edition" on  ef.name = "edition".entry and "edition".entry = 'edition
                    '
173               left outer Join BIBDB.b_fields "editor" on  ef.name = "editor".entry and "editor".entry = 'editor'
174               left outer Join BIBDB.b_fields "editor_cr" on  ef.name = "editor_cr".entry and "editor_cr".entry = '
                    editor_cr'
175               left outer Join BIBDB.b_fields "howpublished" on  ef.name = "howpublished".entry and "howpublished".
                    entry = 'howpublished'
176               left outer Join BIBDB.b_fields "institution" on  ef.name = "institution".entry and "institution".
                    entry = 'institution'
177               left outer Join BIBDB.b_fields "journal" on  ef.name = "journal".entry and "journal".entry = 'journal
                    '
178               left outer Join BIBDB.b_fields "journal_cr" on  ef.name = "journal_cr".entry and "journal_cr".entry =
                     'journal_cr'
179               left outer Join BIBDB.b_fields "key" on  ef.name = "key".entry and "key".entry = 'key'
180               left outer Join BIBDB.b_fields "key_cr" on  ef.name = "key_cr".entry and "key_cr".entry = 'key_cr'
181               left outer Join BIBDB.b_fields "month" on  ef.name = "month".entry and "month".entry = 'month'
182               left outer Join BIBDB.b_fields "note" on  ef.name = "note".entry and "note".entry = 'note'
183               left outer Join BIBDB.b_fields "number" on  ef.name = "number".entry and "number".entry = 'number'
184               left outer Join BIBDB.b_fields "organization" on  ef.name = "organization".entry and "organization".
                    entry = 'organization'
185               left outer Join BIBDB.b_fields "pages" on  ef.name = "pages".entry and "pages".entry = 'pages'
186               left outer Join BIBDB.b_fields "publisher" on  ef.name = "publisher".entry and "publisher".entry = '
                    publisher'
187               left outer Join BIBDB.b_fields "school" on  ef.name = "school".entry and "school".entry = 'school'
188               left outer Join BIBDB.b_fields "series" on  ef.name = "series".entry and "series".entry = 'series'
189               left outer Join BIBDB.b_fields "series_cr" on  ef.name = "series_cr".entry and "series_cr".entry = '
                    series_cr'
190               left outer Join BIBDB.b_fields "title" on  ef.name = "title".entry and "title".entry = 'title'
191               left outer Join BIBDB.b_fields "title_cr" on  ef.name = "title_cr".entry and "title_cr".entry = '
                    title_cr'
192               left outer Join BIBDB.b_fields "type" on  ef.name = "type".entry and "type".entry = 'type'
193               left outer Join BIBDB.b_fields "volume" on  ef.name = "volume".entry and "volume".entry = 'volume'
194               left outer Join BIBDB.b_fields "volume_cr" on  ef.name = "volume_cr".entry and "volume_cr".entry = '
                    volume_cr'
195               left outer Join BIBDB.b_fields "year" on  ef.name = "year".entry and "year".entry = 'year'
196               left outer Join BIBDB.b_fields "year_pu" on  ef.name = "year_pu".entry and "year_pu".entry = 'year_pu
                    '
197               left outer Join BIBDB.temp_fields "author_FORMAT_LAB_NAMES" on  ef.name = "author_FORMAT_LAB_NAMES".
                    entry and "author_FORMAT_LAB_NAMES".entry = 'author_FORMAT_LAB_NAMES'
198               left outer Join BIBDB.temp_fields "author_FORMAT_NAMES" on  ef.name = "author_FORMAT_NAMES".entry and
                     "author_FORMAT_NAMES".entry = 'author_FORMAT_NAMES'
199               left outer Join BIBDB.temp_fields "author_SORT_FORMAT_NAMES" on  ef.name = "author_SORT_FORMAT_NAMES"
                    .entry and "author_SORT_FORMAT_NAMES".entry = 'author_SORT_FORMAT_NAMES'
200               left outer Join BIBDB.temp_fields "editor_FORMAT_CROSSREF" on  ef.name = "editor_FORMAT_CROSSREF".
                    entry and "editor_FORMAT_CROSSREF".entry = 'editor_FORMAT_CROSSREF'
201               left outer Join BIBDB.temp_fields "editor_FORMAT_LAB_NAMES" on  ef.name = "editor_FORMAT_LAB_NAMES".
                    entry and "editor_FORMAT_LAB_NAMES".entry = 'editor_FORMAT_LAB_NAMES'
202               left outer Join BIBDB.temp_fields "editor_FORMAT_NAMES" on  ef.name = "editor_FORMAT_NAMES".entry and
                     "editor_FORMAT_NAMES".entry = 'editor_FORMAT_NAMES'
203               left outer Join BIBDB.temp_fields "editor_SORT_FORMAT_NAMES" on  ef.name = "editor_SORT_FORMAT_NAMES"
                    .entry and "editor_SORT_FORMAT_NAMES".entry = 'editor_SORT_FORMAT_NAMES'
204          group by  EF.S_BST_ID
205                 ,nvl("address".B_ITEM_ID
206                 ,nvl("author".B_ITEM_ID
207                 ,nvl("author_cr".B_ITEM_ID
```

```
                ,nvl("author_FORMAT_LAB_NAMES".B_ITEM_ID
                ,nvl("author_FORMAT_NAMES".B_ITEM_ID
                ,nvl("author_SORT_FORMAT_NAMES".B_ITEM_ID
                ,nvl("booktitle".B_ITEM_ID
                ,nvl("booktitle_cr".B_ITEM_ID
                ,nvl("chapter".B_ITEM_ID
                ,nvl("crossref".B_ITEM_ID
                ,nvl("edition".B_ITEM_ID
                ,nvl("editor".B_ITEM_ID
                ,nvl("editor_cr".B_ITEM_ID
                ,nvl("editor_FORMAT_CROSSREF".B_ITEM_ID
                ,nvl("editor_FORMAT_LAB_NAMES".B_ITEM_ID
                ,nvl("editor_FORMAT_NAMES".B_ITEM_ID
                ,nvl("editor_SORT_FORMAT_NAMES".B_ITEM_ID
                ,nvl("howpublished".B_ITEM_ID
                ,nvl("institution".B_ITEM_ID
                ,nvl("journal".B_ITEM_ID
                ,nvl("journal_cr".B_ITEM_ID
                ,nvl("key".B_ITEM_ID
                ,nvl("key_cr".B_ITEM_ID
                ,nvl("month".B_ITEM_ID
                ,nvl("note".B_ITEM_ID
                ,nvl("number".B_ITEM_ID
                ,nvl("organization".B_ITEM_ID
                ,nvl("pages".B_ITEM_ID
                ,nvl("publisher".B_ITEM_ID
                ,nvl("school".B_ITEM_ID
                ,nvl("series".B_ITEM_ID
                ,nvl("series_cr".B_ITEM_ID
                ,nvl("title".B_ITEM_ID
                ,nvl("title_cr".B_ITEM_ID
                ,nvl("type".B_ITEM_ID
                ,nvl("volume".B_ITEM_ID
                ,nvl("volume_cr".B_ITEM_ID
                ,nvl("year".B_ITEM_ID
                ,nvl("year_pu".B_ITEM_ID
                             ,NULL
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
                                   )
               ) dn
        where bi.id = dn.id and b.ID = bi.b_bib_id and  sty.id =364838 and  dn.S_BST_ID =364838
        and b.id in ('366186','399095')
        and bi.CITE_KEY ('whole-set','whole-set','whole-collection','whole-proceedings','book-full')
)
```

4.4.7 Allgemeine Funktionsbeschreibungen

Konfiguration

Globale Konfigurationen werden im Modul config.py durchgeführt. Der Konstruktor der Klasse bibSQL beinhaltet Variablen, die mit den entsprechenden Methoden verwendet werden können. Die Konfigurationswerte werden direkt in die Datei config.py eingetragen.

Protokollierung

Mit der Methode setLog(self) aus der Klasse config.bibSQL wird die Protokollierung (Logging) gesteuert. Dafür wird das Python-Standardmodul logging verwendet. Alle notwendigen Parameter wie Dateiname, Dateimodus, Level und Format werden in dieser Methode gesetzt. Diese Log-Datei wird bei jedem Aufruf überschrieben. Über den gesamten Programmverlauf wird die Instanz logging mit der Methode log zum Schreiben in die Log-Datei verwendet.

Bildschirmausgabe

Das Modul bibSQL beinhaltet die Funktion stout(). Abhängig vom Parameter verbose gibt es Meldungen auf der Konsole (stout) aus. Die Eigenschaft verbose wird in der Klasse config.bibSQL gesetzt. Alle Bildschirmausgaben werden infolge mit dieser Funktion durchgeführt.

Transaktionslogik

Alle schreibenden Aktionen werden innerhalb eines kompletten Programmablaufs in einer Transaktion durchgeführt. Nach der letzten DML-Anweisung erfolgt die Funktion commit(). Wenn ein Fehler vor dem Beenden der Transaktion eintritt, wird die Transaktion mit der Funktion rollback() beendet und die Datenbank Session ordnungsgemäß beendet.

Fehlerbehandlung

Im aufrufenden Modul (bibSQL) wird nach dem Instanziieren der Klassen database.database(), config.bibSQL, logging und dem OptionParser, in einem try-Block weitergearbeitet. Im Fall einer Ausnahme wird die aufgetretene Fehlermeldung am Bildschirm ausgegeben und in die Protokolldatei geschrieben. Die Datenbanktransaktion und Session werden geschlossen. In den untergeordneten Modulen werden keine Ausnahmeblöcke verwendet, aber besonders bei den Parserkomponenten werden benutzerdefinierte Fehler erzeugt.

4.4.8 Datenbank API

Datenbankspezifische Elemente werden in der Klasse database.database() behandelt. An dieser Stelle wird das verwendete Python Datenbank-API angegeben. Im Konstruktor dieser Klasse wird das entsprechende API definiert. Alle weiteren datenbankspezifischen Unterschiede sind in dieser Klasse gekapselt:

5 Zusammenfassung

Ziel dieser Arbeit war es einen Ersatz für BibTeX zu entwickeln, um die Gestaltung von Styles zu vereinfachen und modularer zu machen. Eine Anforderung war es, bestehende *bst* in eine neue Sprache zu übersetzen. Durch eine höhere Modularität soll es leichter möglich sein, zusätzliche Eigenschaften wie Mehrsprachenfähigkeit zu implementieren. BibTeX hat im Gegensatz zu LaTeX ein Problem beim Umgang mit Multibyte-Encoding. Dieses Problem bezieht sich nicht auf den Umgang mit UTF8-Dateien, sondern steckt im Kern von BibTeX. Alphanumerische Labels werden, wenn sie Sonderzeichen beinhalten, nicht korrekt sortiert. Dieser Umstand macht es notwendig, BibTeX neu zu implementieren. Ausgehend von der LaTeX-„thebibliography“-Umgebung, wurde ein Programm geschaffen, das analog zu BibTeX, LaTeX-*aux*-Dateien liest und formatierte Bibliographien in einer *bbl*-Datei ausgibt.
Ähnlich wie bei anderen Versuchen BibTeX abzulösen, lagen die Schwierigkeiten beim Übersetzen bestehender Style-Files. Dabei mussten Sprachmerkmale von BibTeX genauso berücksichtigt werden wie die Besonderheiten von bestehenden *bst*-Programmen. Da BibTeX-*bst* keine beschreibende Sprache ist, die ausschließlich Konfigurationen zulässt, sondern eine vollwertige prozedurale Programmiersprache, musste ein Weg gefunden werden, wie der volle Funktionsumfang von Style-Dateien erhalten werden kann.
Als Zielsprache bot sich SQL an. Neben dem Vorteil der strukturierten Datenhaltung bietet SQL umfangreiche Möglichkeiten für die Manipulation von Textketten, was sich für eine Markup-Sprache wie LaTeX besonders anbietet.

6 Ausblick

Nach Abschluss der Arbeit liegt eine neue Applikation BibSQL vor. BibSQL liegt in einer unveröffentlichten Beta-Version vor. BibSQL wurde als voll funktionsfähige Applikation und nicht als Prototyp konzipiert. Regressionstests mit BibTeX-Style- und *bib*-Dateien liegen nicht vor. Der meiste Entwicklungsaufwand wurde für den *bst*-Übersetzer aufgebracht. Dementsprechend komplex ist dieses Modul. Die Verwendung dieses Moduls in der vorliegenden Version wurde ausschließlich an Standard-*bst*-Styles getestet. Die Neuentwicklung oder das Verändern von bestehenden Styles verlangt ein gewisses Maß an SQL-Kenntnissen. Für die Eingabe von Metadaten oder Style-Informationen ins Repository wird ein datenbankspezifisches Tool benötigt. Für die Weiterentwicklung von BibSQL bietet sich die Implementierung eines GUI-unterstützten Tools an. Da die Applikation mehrbenutzerfähig konzipiert ist und auf einer Server-Datenbank entwickelt wurde, könnte sie als Instituts- oder unternehmensweite Applikation eingesetzt werden. Für das notwendige Berechtigungskonzept fehlt jedoch die entsprechende Client-Applikation. Die Umsetzung einer Web-Server Version könnte die verteilte Nutzung von bibliographischen Daten und Style-Informationen ermöglichen.

Literaturverzeichnis

[AP97] Dimitrios Andreadis and Ahmed Patel. Developing a multistack iso-sr/z39.50 application gateway. *Computer Standard Interfaces*, 18(5):397–415, 1997.

[Bee93] Nelson Beebe. Bibliography prettyprinting and syntax checking. *TUGboat*, 14:395 – 415, 1993.

[Bee04] Nelson H. F. Beebe. A bibliographer's toolbox. *TUGboat*, 25(1):89–104, 2004. Proceedings of the Pratical TEX 2004 Conference, http://www.tug.org/TUGboat/Articles/tb25-1/beebe-bib.pdf.

[BPLW01] Luca Previtali Brenno, Luca Previtali, Brenno Lurati, and Erik Wilde. Bibtexml: An xml representation of bibtex. In *Poster Proceedings of the Tenth International World Wide Web Conference*, pages 64–65. ACM Press, 2001.

[Dal03] Patrick W. Daly. Customizing bibliographic style files. This paper describes program makebst version 4.1. ftp://ftp.ctan.org/tex-archive/macros/latex/contrib/custom-bib, Sept 2003.

[Dal07] Patrick W. Daly. A master bibliographic style file. This paper describes file merlin.mbs, version 4.20. http://www.andy-roberts.net/misc/latex/tutorial3/merlin.pdf, Apr 2007.

[Dal09] Patrick W. Daly. Natural sciences citations and references. This paper describes package natbib version 8.31. http://jmlr.csail.mit.edu/format/natbib.pdf, Jul 2009.

[Ert92] Anton Ertl. A new approach to forth nativ code generation. In *Euro Forth'92*, pages 73–78, Institut für Computersprachen. Technische Universität Wien, 1992.

[Eur03] EuroTeX. *The EuroTeX2003 paper about Bibulus*, 2003.

[Fen07] Jürgen Fenn. Managing citations and your bibliography with BibTEX. *The PracTEX Journal*, 4:–19, May 2007.

[Gar03] Richard Gartner. Mods: Metadata object description schema. Pearson New Media Librarian Oxford University Library Services, October 2003.

[Har02] Harald Harders. Multilingual bibliographies: Using and extending the babelbib package. *TUGboat*, 23(3/4):344–353, 2002.

[Har03] Harald Harders. Mehrsprachige Literaturverzeichnisse: Anwendung und Erweiterung des Pakets babelbib. *Die TEXnische Komödie*, 4/03(4):39–63, November 2003.

[Huf01] J.-M. Hufflen. MlBibTeX: a New Implementation of BibTeX. In *EuroTeX'2001*, pages 74–94, Kerkrade, The Netherlands, September 2001.

[Huf02a] J.-M. Hufflen. Lessons from a bibliography program's reimplementation. *Theoretical Computer Science*, 65(3):124 –138, 2002.

[Huf02b] J.-M. Hufflen. Multilingual features for bibliography programs: from XML to MlBibTeX. In *Euro TeX 2002*, pages 46–59, Bachotex, Poland, May 2002.

[Huf03a] J.-M. Hufflen, editor. *European Bibliography Styles and MlBibTeX*, volume 24 of *EuroTEX 2003*. TUGboat, TUGboat, 2003.

[Huf03b] J.-M. Hufflen. MlBibTEX's Version 1.3. *TUGboat*, 24:249 –2 62, 2003.

[Huf04a] J.-M. Hufflen. Making MlBibTeX fit for a particular language. Example of the Polish language. *Biuletyn GUST*, 21:14–26, December 2004.

[Huf04b] J.-M. Hufflen. A tour around MlBibTeX and its implementation(s). *Biuletyn gust*, 20:21–28, April 2004. In *BachoTeX Conference*.

[Huf05a] J.-M. Hufflen. Bibliography styles easier with MlBibTeX. In *Euro Tex 2005*, pages 108 –118, 2005.

[Huf05b] J.-M. Hufflen. Implementing a Bibliography Processor in Scheme. In *BachoTEX*, volume 22 of *First*, pages 17–22, March 2005.

[Huf05c] J.-M. Hufflen. MlBibTeX in Scheme (first part). *Biuletyn GUST*, 22:17–22, April 2005. in BachoTeX 2005 conference.

[Huf07] J.-M. Hufflen. MlBibTeX: Reporting the experience. *TUGboat*, 29:157 –162, 2007.

[Huf08] Jean-Michel Hufflen. Languages for bibliography styles. *TUGboat*, 29(3):401–413, 2008.

[KD03] Ronan Keryell and Fabien Dagnat, editors. *BIBTEX++: Toward Higher-order BIBTEXing*, volume 24 of *EuroTEX 2003*. TUGboat, 2003.

[KS09] Maxi Kindling and Matti Stöhr. Literaturverwaltung zur Unterstützung des wissenschaftlichen Publizierens. *cms-journal*, (32):109–111, 2009.

[Lam95] Leslie Lamport. *Das LaTeX-Handbuch*. Addison-Wesley Publishing Company, Bonn, Paris u. a., 1995.

[Leh09] Philipp Lehman. The biblatex package. programmable bibliographies and citations. ftp://ftp.ctan.org/tex-archive/macros/latex/exptl/biblatex, 2009.

[Lin07] Anselm Lingnau. *LaTeX Hacks*. O'Reilly, 1. edition, 2007.

[Mar05] Nicolas Markey. Tame the beast the b to x of BibTeX. 45-page tutorial presents and explains, as clearly and exhaustively as possible, what BibTeX can do. http://www.tex.ac.uk/tex-archive/info/bibtex/tamethebeast/ttb_en.pdf, Oct 2005.

[MG00] Rune Mathisen and Vidar Bronken Gundersen. *SGML/XML Character Entity Reference*, 08 2000. http://www.bitjungle.com/~isoent/.

[Mil] Paul Miller. Z39.50 for all. http:www.ariadne.ac.uk/issue2.

[Mil05] Tristan Miller, editor. *Biblet: A portable* BibTeX *bibliography style for generating highly customizable XHTML*, volume 26, 2005. Includes Practical TeX 2005 Conference Proceedings.

[Neu97] Gerd Neugebauer. *BibTool – A Tool to Manipulate* BibTeX *Files*, 2.41 edition, 1997. http://www.gerd-neugebauer.de/software/TeX/BibTool/bibtool.dvi.

[NO07] Michael Norris and Charles Oppenheim. Comparing alternatives to the web of science for coverage of the social sciences' literature. *Journal of Informetrics*, 1(2):161 – 169, 2007.

[Pat88a] Oren Patashnik. *BibTeXing*, 2 1988. CTAN://biblio/bibtex/distribs/doc/btxdoc.tex.

[Pat88b] Oren Patashnik. *Designing BibTeX Styles*, February 1988. CTAN://biblio/bibtex/distribs/doc/btxhak.tex.

[Rai02] Bernd Raichle. Tutorium: Einführung in die BibTeX-Programmierung, 2002. Tutorial held at the DANTE conference 2002 at Erlangen.

[Sia08] Uwe Siart. Verwendung von BibTeX zur Erzeugung von Literaturverzeichnissen. *Die TeXnische Komödie*, 4:51–61, 2008.