Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr.-Ing. Dr. h. c. Theo Härder

Realisierung einer automatischen Klassifizierungs- und Beschlagwortungskomponente zur Nutzung im Projekt META-AKAD

Projektarbeit von

Christian Weber
c_weber@informatik.uni-kl.de

Betreuer:

Dipl.-Inform. Marcus Flehmig
(AG Datenbanken und Informationssysteme)

Dr. Markus Junker
(Deutsches Forschungszentrum für Künstliche Intelligenz)

April 2003

Inhaltsverzeichnis

1 Einführung

1.1 Vorstellung des Projektes META-AKAD

Das Projekt *Metadatenzugang für akademisches Lehr- und Lernmaterial* (nachfolgend nur noch META-AKAD [1] genannt) ist ein vom Deutschen Forschungsnetz e.V. (DFN [2]) mit Mitteln des Bundesministerium für Bildung und Forschung (BMBF [3]) gefördertes Gemeinschaftsprojekt der Universitäten Regensburg und Kaiserslautern. Ziel dieses Projektes ist es, einen einheitlichen Zugang zu dem im Web verfügbaren Lehr- und Lernmaterial aus verschiedenen Fachrichtungen bereitzustellen.

Im Web gibt es ein großes Angebot an elektronischem Lehr- und Lernmaterial, das für Lehrveranstaltungen oder für ein Selbststudium nutzbar ist. Diese Materialien sind weltweit auf Server von Bildungseinrichtungen, Verlagen oder Organisationen verteilt. Die Suche nach Informationen im Web liefert immer wieder Suchergebnisse von extrem schlechter Qualität, weil die heterogenen Dokumente weder systematisch nach ihrem Inhalt erschlossen wurden und kaum durch geeignete Metadaten beschrieben sind, noch die Suchmaschinen effektive Verfahren anwenden. Sie benutzen meist nur Schlüsselwort-basierte Suche und setzen vor allem keine Domänen-spezifischen Ontologien oder Klassifikationsschemata ein, um beispielsweise semantische Ähnlichkeit in den Griff zu bekommen [FH02].

Im Rahmen dieses Projektes soll ein Informationssystem geschaffen werden, das sowohl auf die Bedürfnisse von Lehrenden als auch auf die von Lernenden zugeschnitten ist. Das elektronische Lehr- und Lernmaterial soll auf kooperativer Basis gesammelt, durch standardisierte und materialspezifische Meta-Daten erschlossen, nach inhaltlichen und didaktischen Kriterien bewertet und dann in einer einheitlichen Nutzeroberfläche zugänglich gemacht werden. Dieser Service wird exemplarisch für die Fächer Mathematik, Physik, Germanistik, Psychologie und Biologie aufgebaut und soll so ausgelegt werden, dass er auf zusätzliche Disziplinen erweitert werden kann.

Der Erschließungsprozeß läßt sich in die in Abbildung 1 gezeigeten Aufgaben untergliedern. Für die einzelnen Erschließungsschritte sind unterschiedliche Kompetenzen der Bearbeiter notwendig [Sch01a].

Abbildung 1: Mehrschrittvorgang der Erschließung

Die Erschließung von Lehr- Lernmaterial soll so effizient wie möglich durchgeführt werden.

[1] `http://www.bibliothek.uni-regensburg.de/projekte/metaakad/metaakad.htm`
[2] `http://www.dfn.de/`
[3] `http://www.bmbf.de/`

Um dies zu erreichen, sollen auch automatische und semiautomatische Verfahren eingesetzt werden. Die einzelnen Erschließungswerkzeuge werden in Abbildung 2 dargestellt.

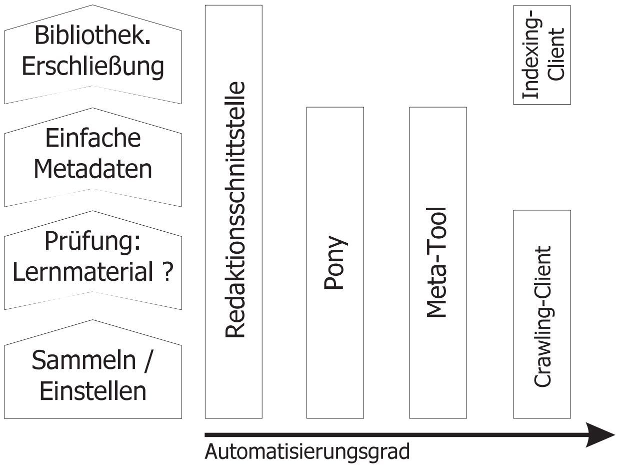

Abbildung 2: mauelle und automatische Erschließungswerkzeuge

In der Redaktionsschnittstelle, können alle Metadaten manuell editiert werden. Mit dem semiautomatischen Werkzeug *Pony* können neue Links in die Datenbank eingefügt werden. Der Crawling-Client sucht, nachdem er manuell konfiguriert wurde, automatisch nach Lehr- und Lernmaterial und fügt sie in die Datenbank ein. Der Indexing-Client, soll automatisch den META-AKAD Datenbestand nach fehlenden Klassifikationen und fehlenden Schlagworten durchsuchen und automatisch Vorschläge in die Datenbank eintragen. Diese automatisch erzeugten Metadaten sollen dann bei der nächsten manuellen Bearbeitung intellektuell validiert werden.

Das semi-automatisches Erschließungswerkzeug *Meta-Tool* ist ein Namenserkennungswerkzeug, das das Ziel hat Personennamen sowohl aus englischen, als auch einem deutschen semi-strukturierten Dokumenten (z. B. HTML) zu extrahieren und diese als sinnolle Metadaten vorzuschlagen [Cha02].

Schließlich sollen die Materialien von Fachleuten bewertet werden. Durch die Integration einer solchen Qualitätskontrolle wird die Verwertbarkeit der Materialien für Lehr- und Lernzwecke deutlich erhöht und damit ein wesentlicher Mehrwert des Dienstes geschaffen. Dieses Projekt ordnet sich in eine Reihe von Projekten ein, die dazu dienen durch die Nutzung des Internet die Lehre an den Hochschulen effektiver zu gestalten [Gei00].

1.2 Einordnung dieser Arbeit in das Projekt META-AKAD

In dieser Arbeit soll eine Softwarekomponente entwickelt werden, die in der Lage ist, automatisiert Klassifikationen nach der Regensburger Verbundklassifikation (RVK [4]) und Schlagworte aus der deutschen Schlagwortnormdatei (SWD [5]) für Dokumente, die als Lehr- oder Lernmaterial eingestuft wurden, zu vergeben.
Die Softwarekomponente soll sich in den allgemeinen Erschließungsprozeß eingliedern und bis auf die Anpassung von Konfigurationsdaten vollständig automatisiert im Hintergrund lauffähig sein. Die Entwicklung soll mit Hilfe von bereits gesammelten Dokumenten, die sich in der vorläufigen Datenbank befinden, durchgeführt werden.

Um die Auswertung und Verfeinerung des Verfahrens während der Entwicklung zu vereinfachen, wird das Verfahren nur mit deutschsprachigen Mathematikdokumenten getestet. Bei der Entwicklung soll natürlich darauf geachtet werden, das dass Verfahren ohne großen Änderungsaufwand auch auf andere Fächer und Sprachen erweitert werden kann.

1.3 Motivation

Die zunehmende Verfügbarkeit von Informationen in elektronischer Form durch das Internet erzeugt die Notwendigkeit, in dieses ”Chaos” von Information eine gewisse ”Ordnung” zu bringen. Es ist eine Herausforderung die ”rohen Daten”, die nichts anderes als eine Aneinandereihung von Zeichen sind, in einen bedeutungstragenden Kontext einzuordnen, so dass man von strukturiertem Wissen sprechen kann.

Auf Grund der immer schneller wachsenden Dokumentenmenge im Internet ist eine manuelle Strukturierung der Datenmenge kaum noch möglich. Deshalb müssen Verfahren entwickelt werden, die den Menschen bei der Suche nach spezifischen Informationen in der riesigen Dokumentenmenge unterstützen. Einige Schwerpunkte, an denen zur Zeit geforscht wird, sind Modelle zur Repräsentation von Dokumenten, Methoden zur Ähnlichkeitsbestimmung von Anfragen zu Dokumenten und zur Indexierung von Dokumentenmengen sowie die automatische Klassifikation.

Diese Methoden finden ihre Anwendung zum Beispiel in den Suchmaschinen des World Wide Web (WWW), mit denen das Auffinden von Informationen erleichtert werden soll. Um bei der Suche möglichst viele relevante Dokumente zu erhalten, müssen die Anfragen immer spezifischer gestellt werden. Aus diesem Grund müssen die oben genannten Methoden verfeinert und erweitert werden.

Mit dieser Arbeit soll ein kleiner Beitrag zur Lösung dieses Problem geleistet werden, indem die Möglichkeiten zur automatischen Klassifizierung und Beschlagwortung einer bereits vor-

[4] `http://www.bibliothek.uni-regensburg.de/Systematik/systemat.html`
[5] `http://www.ddb.de/professionell/swd.htm`

sortierten Datenmenge untersucht werden.

1.4 Gliederung der Arbeit

In Kapitel 2 werden zunächst verschiedene Verfahren, die zur automatischen Klassifizierung und Beschlagwortung genutzt werden, vorgestellt. Anschließend wird der zur Realisierung der automatischen Klassifizierungs- und Beschlagwortungskomponente gewählte Ansatz in Kapitel 3 veranschaulicht, bevor die eigentliche Entwicklung in Kapitel 4 beschrieben wird. Anschließend wird in Kapitel 5 eine abschließende Berwertung des entwickelten Verfahrens vorgenommen. Im Anhang ab Seite 55 befindet sich eine Dokumentation der entwickelten Javaklassen und eine Beschreibung zur Bedienung des Indexierungsclients.

2 Automatische Indexierung

2.1 Begriffsdefinition Indexierung

Bei der Speicherung von Dokumenten oder Dokumentreferenzen in Datenbanken ist es wichtig, den Dokumentinhalt aufzubereiten und entsprechende Metadaten (z.B. Schlagwörter) mit abzuspeichern, um bei einer späteren Suche relevante Dokumente leichter finden zu können. Eine Möglichkeit der Aufbereitung ist die Vergabe von Kennzeichnungen in einer künstlichen Sprache, einer Notation (z.B. einer Klassifikation) oder von natürlichsprachigen Deskriptoren (Inhalt beschreibende Schlagwörter).

Werden Deskriptoren von Bibliothekaren vergeben, spricht man von intellektueller Indexierung. Wenn diese Arbeit mit Computern erledigt wird, bezeichnet man dies als automatische (oder maschinelle) Indexierung. Sind Mensch und Computer daran beteiligt, handelt es sich um computergestützte Indexierung. Der Einsatz dieser drei Verfahren richtet sich nach ökonomischen und Qualitätserwägungen. Intellektuelle Indexierung wird von Fachleuten durchgeführt und ist zeit- und kostenaufwändig, erbringt aber die besten Ergebnisse. Vollautomatische Indexierung ist schnell, aber im allgemeinen auch von minderer Qualität [Luc01].

2.2 Begriffsbestimmung

Unter automatischer Indexierung wird ganz allgemein die maschinelle Ermittlung von Metadaten aus Dokumenten verstanden. Das Spektrum der Möglichkeiten ist dabei weit gestreut [Lep02]:

- Einfache Stichwortextraktion (Stichwortsuche in nahezu jedem OPAC (Online Public Access Catalog))
- Statistische Verfahren
- Computerlinguistische Verfahren
  - Regelbasierte Verfahren (z.B. in OSIRIS [Uni99])
  - Wörterbuchbasierte Verfahren (z.B. MILOS [ULB01])
- Begriffsorientierte Verfahren

Andere Einteilungen sind denkbar, da die meisten Systeme Elemente mehrerer Verfahren integrieren.

2.3 Existierende Verfahren

Programme zur automatischen Indexierung gibt es bereits seit den 60er Jahren, doch liegt deren Anwendungsbereich vor allem bei Fachdatenbanken in englischer Sprache. Erfahrungen mit dem Einsatz der automatischen Indexierung in deutschen Bibliotheken gibt es bislang nur wenige. In den folgenden Abschnitten werden verschiedene Projekte, die sich mit der Erschließung und dem Retrieval (Informationssuche) von Bibliotheksdaten beschäftigen, vorgestellt. Dabei wird auf die Projekte, die sich mit deutschrachigen Daten beschäftigen, etwas genauer eingegangen.

2.3.1 MILOS

Zwischen 1993 und 1996 wurde an der Universität Düsseldorf und der Universität des Saarlandes an dem DFG-Projekt MILOS I & II (Maschinelle Indexierung zur erweiterten Literaturerschließung in Online-Systemen) gearbeitet. Dabei wurde untersucht inwieweit das Indexierungssystem IDX, das von Prof. Dr. Harald H. Zimmermann, Fachrichtung Informationswissenschaft der Universität des Saarlandes, entwickelt wurde, für den Einsatz in Bibliotheken geeignet ist.

Das Indexierungssystem IDX bietet folgende Funktionalität für die Sprachen Deutsch, Englisch und Französisch:

- Reduktion der im Text vorkommenden Wortformen auf Grundformen,
- Eliminierung von Stoppwörtern (Wörter im Text, die für die Analyse des Textes keine besondere Bedeutung haben),
- Zerlegung von Komposita in Wortableitungen und (sinnvolle) Bestandteile,
- Mehrwort-Erkennung und Wortbindestrichergänzung,
- Wortbezogene Übersetzung.

Ziel von MILOS I war es, das System IDX an die spezielle Arbeitsumgebung einer wissenschaftlichen Universalbibliothek anzupassen. Das System wurde mit Hilfe von Titeldaten der Universitäts- und Landesbibliothek Düsseldorf, die automatisch indexiert wurden, getestet. Aus dem praktischen Einsatz heraus wurden auf Grundlage der indexierten Daten neue Wörterbücher aufgebaut bzw. bereits bestehende stark erweitert [ULB01].

Ein abschließender Retrieval-Test, der auch ein wesentlicher Bestandteil des Projekts war, führte mit automatisch erzeugten Indexdaten zu durchweg positiven Ergebnissen, so dass die ULB Düsseldorf die automatische Indexierung als festen Bestandteil der Suchmöglichkeiten in ihren OPAC integriert hat.

Datengrundlage für den Retrieval-Test war eine repräsentative Teilmenge von 40.000 Dokumente aus dem maschinenlesbaren Datenbestand der ULB Düsseldorf. Die deutsch-, englisch- und französischsprachigen Titel der Dokumente wurden nach Anwendung von Funktionen zur Grundformerzeugung, Dekomposition und Derivation automatisch indexiert. Die Ergebnisse der automatischen Indexierung wurden nicht intellektuell nachbearbeitet, sondern unverändert zu den Datensätzen hinzugefügt.

Zur Bewertung des Retrieval-Tests wurden die beiden Kriterien Precision und Recall ausgewählt, da diese allgemein akzeptiert sind. Eine Erklärung dieser beiden Maße befindet sich in Kapitel 3.4. Recall und Precision hängen entscheidend von der Anzahl der Dokumente ab, die für die Suche als relevant erachtet werden, d.h. Dokumente die die gesuchten Informationen enthalten. Um die Anzahl der relevanten Dokumente zu bestimmen, wurde eine Suche mit den zur Verfügung stehenden Mitteln wie Indexsuche und Freitextsuche mit Operatoren durchgeführt. Auf diese Art und Weise wurde das Retrievalkriterium auf die Anfragen der Benutzer ausgerichtet und nicht auf den vorhandenen Datenbestand.

Das bei der Suche benutzte BISMAS (Bibliographisches Informations-System zur Maschinellen Ausgabe und Suche [6]) sieht als Standardsuchweg die Suche über einen Index vor. Für den Test wurden drei unterschiedliche Indices angelegt:

- Index 1:
  Der Index 1 wurde als reiner Titelstichwortindex aufgebaut.

- Index 2:
  Der Index 2 ist ein Titelstichwortindex, der um die Ergebnisse der automatischen Indexierung ergänzt wurde.

- Index 3:
  Der Index 3 besteht aus einem Titelstichwortindex und enthält zusätzlich intellektuell vergebene Schlagwörter.

Auf den drei angelegten Indices wurden jeweils 50 verschiedene Suchanfragen durchgeführt. Die Mittelwerte der Ergebnisse sind in Tabelle 1 abgebildet.

	Precision	Recall
Index 1	59%	14%
Index 2	83%	51%
Index 3	83%	39%

Tabelle 1: Ergebnisse des Retrieval-Tests

Bei der Suche über den Index 1 ist der Recall erwartungsgemäß niedrig, da dieser Index mit den sprachlichen Problemen wie Pluralendungen, Komposita und ähnlichem belastet ist. Die Steigerung des Recall um den Faktor 3 bei der Suche über den Index 2 entspricht den Erwartungen, die an eine sprachliche Bereinigung, die bei der automatischen Indexierung

[6] http://www.bismas.de/

durchgeführt wird, gestellt werden. Mit der Suche über den Index 3 sollten die Auswirkungen einer intellektuellen Beschlagwortung auf den Retrival-Erfolg untersucht werden. Der Recall bei der Suche über den Index 3 ist schlechter als bei der automatischen Indexierung, da bei der intellektuellen Beschlagwortung nicht alle Dokumente beschlagwortet wurden.

In MILOS II wurde das System durch Integration der Schlagwortnormdatei (SWD) als elektronischem Thesaurus erweitert.

2.3.2 KASCADE

Das Projekt KASCADE (KAtalogerweiterung durch SCanning und Automatische DokumentErschließung, [ULB99]) ist das Nachfolgerprojekt von MILOS. Ziel des Projekts ist die Erweiterung konventioneller bibliothekarischer Titeldaten um zusätzliche inhaltsrelevante Informationen und deren automatische Erschließung. Die so erreichte umfassende verbale und klassifikatorische Erschließung der Dokumente schafft die Basis für den Einsatz fortgeschrittener Navigations- und Suchverfahren auf der Retrievalseite.

An diesem Projekt waren die Juristischen Fakultät der Heinrich-Heine-Universität Düsseldorf, die Fachhochschule Köln, mit dem Fachbereich Bibliotheks- und Informationswesen, und die Fachrichtung Informationswissenschaft an der Universität des Saarlandes sowie die Universitätsbibliothek Bielefeld beteiligt.

Die Datenbasis für KASCADE war der Datenbestand des Fachgebiets Jura der ULB Düsseldorf mit ca. 30.000 Dokumenten. Dieser Bestand war wegen des hohen deutschsprachigen Anteils für die Zielsetzung des Projekts besonders gut geeignet. Es wurde ein Konzept zur Anreicherung mit Erschließungsdaten erarbeitet. Berücksichtigt wurden je nach Verfügbarkeit bzw. Eignung Inhaltsverzeichnisse, Zusammenfassungen und Begriffe aus Sachregistern. Soweit diese nicht bereits in maschinenlesbarer Form vorlagen, wurden sie eingescannt, durch automatische Texterkennung (OCR) umgesetzt und mit den Titelaufnahmen verknüpft.

Grundlage für das Erschließungskonzept zur KASCADE-Datenbasis ist die Berücksichtigung der unterschiedlichen Dokumenttypen in ihren verschiedenen Erschließungsgraden. Die Größe der Datenbasis und die mögliche Tiefe der Erschließung bis hin zum Volltext legen es nahe, weitesgehend automatische Verfahren zu verwenden. Zu diesen automatischen Verfahren gehören unter anderem die *selektive automatische Indexierung* und die *Themen-Aspekt-Identifizierung*, die in den folgenden Abschnitten vorgestellt werden.

SELIX-JB: Selektive automatische Indexierung

Die Übernahme unterschiedlicher additiver Dokumentinformationen führt zu einer starken Anreicherung der Dokumente mit unkontrollierten Stichwörtern. Die große Menge an Stichwörtern ist jedoch nicht immer vertretbar. Aus diesem Grund muss das Ziel einer automatischen Indexierung umfangreicher textueller Daten eine gewichtete Extraktion sein, d.h. eine maschinelle Selektion und Indexierung.

Mit dem im Teilprojekt SELIX-JB entwickelten Verfahren zur selektiven Indexierung werden auf der Basis von Häufigkeiten in der Gesamtkollektion der Dokumente in Relation zu den Häufigkeiten innerhalb eines Dokumentes Entscheidungen getroffen, ob bestimmte Begriffe für die Beschreibung eines spezifischen Dokumentes wichtig sind. Dieser Ansatz kann als Ausgangsbasis eines lernenden Systems benutzt werden, indem beim weiteren Aufbau alle 'Erfahrungen' in Wörterbüchern oder Datenbanken festgehalten werden, die dann bei der Indexierung als Entscheidungshilfen zur Verfügung stehen.

Durch dieses Gewichtungsverfahren soll die Volltext-Retrievalfunktion nicht ersetzt, sondern bei Bedarf ergänzt werden. Das Verfahren muss zudem dem wachsenden Datenbestand in regelmäßigen Abständen angepasst werden.

THEAS-JB: Themen-Aspekt-Identifizierung

Durch die Anreicherung von Dokumenten sowie die Verfügbarkeit von elektronischen Volltexten steigt die Zahl verbaler Sucheinstiege über Stichwörter im Retrieval enorm an. Deshalb soll zusätzlich eine klassifikatorische Erschließungskomponente entwickelt werden.

Mit der Funktion einer sogenannten Themen-Aspekt-Identifizierung (THEAS-JB) wird versucht, auf der Basis der Ergebnisse der automatischen Indexierung mit MILOS eine Zuordnung des Dokuments zu einer möglichen Thematik und möglichen Aspektierungen dieser Thematik automatisiert durchzuführen. Die Identifizierung erfolgt dabei auf der Grundlage eines semantischen Bezugssystems, das anhand der maschinenlesbaren Datensammlung durch statistische Analyse und intellektuelle Markierungen spezifischer lexikalischer Einträge aufgebaut wird.

Anhand der Analyse der verfügbaren Materialien wird versucht ein Kategorien- und Merkmalsystem abzuleiten. Hier finden sich einige Beispiele wie derartige Regeln aussehen können:

- Ein erster Ansatz ist der Versuch, aus den Volltext-Daten die Nutzerzielgruppe des Dokumentes abzuleiten. Dazu werden Wörter und Wortgruppen, die entsprechende Hinweise geben können, entsprechend markiert. Beispielsweise geben Wörter wie *„ein Kommentar zu“* oder *„ein praktisches Handbuch“*, aber auch Wörter wie *„...prinzip“* Hinweise auf eine eher theoretische oder praxisorientierte bzw. auf Experten oder auch Laien ausgerichtete Zielgruppe.

- Ein weiteres Kriterium für eine Selektion ist der Dokumenttyp. Soweit dieser nicht aus den Formaldaten zu erschließen ist, ergeben sich vielfach wiederum aus dem Titel oder Zusammenfassung entsprechende Hinweise etwa auf eine Gesetzessammlung, eine Festschrift, einen Kommentar, ein Wörterbuch oder ähnliches.

- Die eigentliche Aufgabe ist jedoch die Differenzierung von Themen- und Aspekt-Beziehungen. Hierzu wird eine Sammlung verbaler und struktureller Indikatoren entwickelt. So geben beispielsweise Elemente wie *„neue Erkenntnis“* oder *„aktuelle Entwicklung“* ein Hinweis auf die Aktualität eines Themas; Elemente wie *„ein Vergleich zwischen“* stellen entsprechende Themen-Beziehungen her, Lokalisierungen wie *„in Bayern“* können ein Thema einschränken. Aber auch lexikalisierte Strukturen dienen der Differenzierung, etwa bei Komposita und Mehrwortbegriffen wie *„Prinzipien des ...“* bzw. *„...prinzip“*.

Ziel ist es jedoch ein allgemeineres, offenes Regelsystem abzuleiten, das es nach entsprechender technischer Implementierung erlaubt, diese Zuordnungen weitgehend ohne lexikalische Unterstützung zu realisieren.

Der abschließende Retrievaltest führte zu Ergebnissen, die nicht signifikat besser waren, als bei einer Suche mit MILOS bzw. einer Freitextsuche. Als mögliche Schwachpunkte des Retrievaltests wurden eine ungleiche Verteilung der Fragetypen sowie eine starke Streuung der Relevanzurteile angegeben.

2.3.3 OSIRIS

OSIRIS war ein gemeinsames Projekt der Universitätsbibliothek Osnabrück und des Instituts für Semantische Informationsverarbeitung (ISIV) der Universität Osnabrück und wurde durch die Deutsche Forschungsgemeinschaft (DFG) von 1996 - 1999 gefördert. Die Abkürzung OSIRIS steht für Osnabrück Intelligent Research Information System.

OSIRIS ist ein multilinguales intuitiv-natürlichsprachlich zu benutzendes Retrievalsystem insbesondere für bibliographische Datenbanken, das die Anwendung klassischer Retrievaltechniken (spezielle Kommandos, Boolesche Verknüpfungen, Trunkierung etc.) für den Benutzer überflüssig macht. Dabei erzielt OSIRIS im Vergleich zu einem konventionellen OPAC qualitativ bessere Suchergebnisse [Uni99]. Als Nebenprodukt ensteht dabei eine Fachreferentenoberfläche, die effektive Möglichkeiten im Rahmen eines Computer Aided Indexing (CAI) erbringen kann. Mit den OSIRIS-Werkzeugen kann auch eine Klassifikation automatisch erzeugt werden.

Die Eingabe zur thematischen Recherche in einem OSIRIS-System ist die Vervollständigung des auf der Eingabemaske vorgegebenen Teilsatzes *„Ich suche Literatur zum Thema ...“*. Auf der Basis der Halbsatzergänzung genügt zur Analyse der Benutzereingabe ein auf bestimmte syntaktische Phänomene optimierter Parser, der aus einer deklarativ spezifizierten Grammatik, in der die Regeln des modellierten Sprachausschnitts enthalten sind, besteht. Die Arbeit des Parsers wird von verschiedenen Modulen unterstützt:

- Komponente zur Behandlung fehlerhafter Eingaben
  Schreibfehler können aufgrund phonetischer Ähnlichkeiten in vielen Fällen erkannt

werden. Auch das Verdrehen von Buchstaben kann erkannt werden. Diese Korrekturen werden natürlich nicht durchgeführt, ohne den Benutzer zu informieren.

- Lexikon-Komponente
  Um komplexe Nominalphrasen, bestehend aus Eigennamen, Einschränkungen und Ergänzungen verarbeiten zu können, müssen grammatische Informationen zu einzelnen Wörtern verfügbar sein. Das Lexikon enthält unter anderem Angaben über Wortart, Numerus und Genus sowie eine rudimentäre Semantik der Funktionswörter wie *„in“* und *„zur Zeit von“*, die eine Einschränkung oder Modifikation darstellen.

- Morphologiekomponente
  Es ist wünschenswert, dass Eingaben des Benutzers wie *„Massenmedien“* und *„Massenmedium“* nicht als voneinander unabhängig betrachtet werden. Um die einzelnen morphologisch markierten Formen eines Wortes zueinander in Beziehung zu setzen, ist ein regelbasiertes Wissen notwendig. Mit dieser Komponente können auch Eingaben wie *„Marktwirtschaft in China“* und *„chinesische Marktwirtschaft“* zueinander in Bezug gesetzt werden.

- Komponente zur Zerlegung von Komposita
  Aufgrund der in der deutschen Sprache häufig auftretenden Kompositabildung ist es nicht sinnvoll, Wörter wie *„Pädagogikstudium“* als eigenständige Wörter zu betrachten. Eine Zurückführung auf *„Studium der Pädagogik“* oder *„Studieren von Pädagogik“* ist sinnvoll.

2.3.4 DESIRE

DESIRE (Development of a European Service for Information on Research and Education, [Lun97]) ist ein internationales Projekt im Rahmen des ”Telematics Applications Programme” der EU und verfolgt das Ziel, Informations-Netzwerke für die Forschung aufzubauen. Hier liegen Ansätze und Lösungen zum Einsatz intellektueller und vor allem automatischer Klassifikation in Fachinformationsdiensten im Internet vor, und es stehen entsprechende Werkzeuge zur Verfügung, die frei genutzt werden können. Die Klassifikation in DESIRE benutzt einen speziellen Thesaurus für die Ingenieurswissenschaften.

2.3.5 CORC

Anfang 1999 startete das Projekt CORC (Cooperative Online Resource Catalog, [Hic98]) des Online Computer Library Center (OCLC). Ziel ist der Aufbau einer internationalen, kooperativ geführten Nachweis-Datenbank für Internet-Ressourcen. Die Erschließung geschieht zum einen durch die Sammlung und Generierung von Metadaten, zum anderen durch die (teil)automatische Umsetzung in Katalogdaten nach MARC II (maschine readable cataloging [7] und MARC 21. Es werden außerdem strukturierte Web-Seiten erzeugt. Dabei wird auch eine automatische Vergabe von Notationen der Dewey Decimal Classification (DCC [8])

[7] `http://www.loc.gov/marc/`
[8] `http://www.oclc.org/dewey/`

verwendet.

3 Beschreibung des gewählten Ansatzes

3.1 Vorüberlegungen

Bei den Überlegungen, welcher Ansatz zur Klassifizierung und Beschlagwortung verfolgt wird, musste vor allem darauf geachtet werden, dass das Problem auch in der vorgegebenen Zeit und mit den zur Verfügung stehenden Mitteln gelöst werden kann. Deshalb scheiden Verfahren, bei denen Wörterbücher und Ähnliches zum Einsatz kommen, aus, da diese nicht frei erhältlich sind und es sehr zeitaufwändig ist, diese selber zu erstellen. Außerdem wäre man bei der Verwendung von Wörterbüchern nicht mehr sprachunabhängig.

Aus diesen Gründen soll ein Lernverfahren benutzt werden, um aus bereits klassifizierten und beschlagworteten Dokumenten die Informationen, die zur Klassifizierung und Beschlagwortung neuer Dokumente benötigt werden, zu extrahieren. Als eines der besten Lernverfahren hat sich die sogenannte Support Vector Machine (SVM) herausgestellt. Die Support Vector Machine ist ein aus der Lerntheorie hervorgeganges Verfahren, welches lineare und nichtlineare statistische Klassifikatoren lernt [Joa98].

3.2 Das Support-Vector-Machine Verfahren

Das Lernverfahren Support-Vector-Machine (SVM) wurde ursprünglich von Vladimir Vapnik und seinen Mitarbeitern entwickelt und ist eine sehr mächtige Methode. Es ist ein noch recht junges Verfahren, das bereits in vielen Anwendungsgebieten die meisten anderen Systeme übertroffen hat. SVM-Verfahren werden nicht nur zur Klassifizierung von Texten genutzt, sondern finden auch in Bereichen der Klassifizierung von Bildern, Schrifterkennung, Objekterkennung und vielen anderen Bereichen Verwendung [9].

SVM bilden ein System zum Lernen von linearen Funktionen in einem kerninduzierten Merkmalsraum unter Berücksichtigung der Ergebnisse der Generalisierungstheorie und der Ausnutzung der Optimierungstheorie. Man versucht lineare Funktionen zu nutzen, da sie recht gut erforscht und einfach realisierbar sind. Jedoch sind lineare Funktionen nicht mächtig genug, um reale Probleme zu lösen. Daher geht man in einen hochdimensionalen kerninduzierten Merkmalsraum über, in dem viele Probleme linear trennbar werden. In diesem hochdimensionalen Merkmalsraum werden effiziente Algorithmen benötigt, die von der Generalisierungs- und Optimierungstheorie zur Verfügung gestellt werden [Sch01b].

SVM werden häufig zur binären Klassifikation eingesetzt, d.h. sie entscheiden ob ein Dokument zur entsprechenden Klasse gehört oder nicht. Um eine SVM zur Klassifikation einsetzen zu können, müssen die Dokumente in eine entsprechende Vektorrepräsentation $\vec{x_i}$ überführt werden (siehe Kapitel 3.3). Bei SVM, die zur binären Klassifikation genutzt werden, muss für eine gegebene Trainingsmenge, die sowohl positive als auch negative Beispiele enthält,

[9] `http://www.clopinet.com/isabelle/Projects/SVM/applist.html`

$$(\vec{x_1}, y_1), (\vec{x_2}, y_2), ..., (\vec{x_n}, y_n) \times \{-1, 1\} \tag{1}$$
$$\vec{x} \in \mathbb{R}^n \tag{2}$$

eine Funktion

$$f : \mathbb{R}^n \longmapsto \{-1, 1\} \tag{3}$$

gesucht werden, die die Trainingsmenge korrekt in zwei verschiedene Klassen unterteilt. Falls die Trainingsmenge im Merkmalsraum $\mathbb{R}^n$ nicht linear trennbar ist, werden die Daten in einen höherdimensionalen Vektorraum transformiert. Dort wird dann versucht die Trainingsdaten linear zu trennen. Je höher die Dimension des neuen Vektorraums, desto größer ist die Wahrscheinlichkeit, dass die Daten linear trennbar sind.

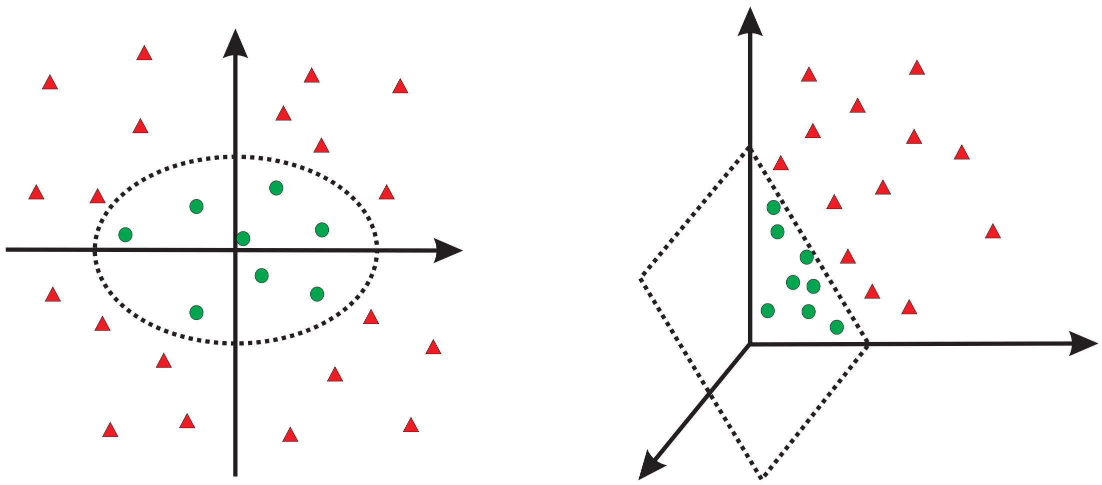

Abbildung 3: Transformation in einen höherdimensionalen Merkmalsraum

Abbildung 3 zeigt eine Menge von Trainingsdaten, die im $\mathbb{R}^2$ nicht linear trennbar sind. Nach Anwendung der Funktion

$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \tag{4}$$
$$(x_1, x_2) \mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2) \tag{5}$$

kann man anhand der eingezeichneten Ebene erkennen, dass die Daten im $\mathbb{R}^3$ linear trennbar sind. Dieses Beispiel macht deutlich, wie entscheidend die Wahl der Dimension des Vektorraums für die lineare Trennung der Trainingsdaten ist. Man versucht, die Dimension des Vektorraum möglichst gering zu halten, obwohl mit steigender Dimension des Vektorraums auch die Wahrscheinlichkeit steigt, dass eine linare Funktion gefunden wird, die die Trainingsdaten trennen kann. Bei steigender Dimension des Vektorraums nimmt auch der Rechenaufwand zu. Außerdem ist mit steigender Dimension auch eine Verschlechterung der Generalisierung zu erwarten [Hem01].

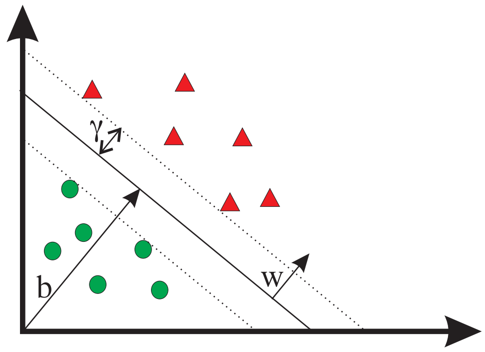

Abbildung 4: Hyperebene (w, b) und Trainingsdaten mit Trenngüte γ

Nachdem die Trainingsdaten in einen höherdimensionalen Vektorraum transformiert worden sind, muss nun die lineare Entscheidungsfunktion f (Formel 3) gefunden werden, die die positiven und negativen Trainingsdaten (Formel 2) aus dem Merkmalsraum X trennt. Ein Eingabevektor $\vec{x}$ wird dann positiv klassifiziert, wenn $f(\vec{x}) \geq 0$. Abbildung 4 zeigt die lineare Entscheidungsfunktion f, die der Trennhyperebene $(\vec{w}, b)$ entspricht. Eine Hyperebene ist eine Ebene in einem höherdimensionalen Vektorraum. Die lineare Entscheidungsfunktion hat die Form

$$f(\vec{x}) = \langle \vec{w} \cdot \vec{x} \rangle + b \tag{6}$$

wobei $\langle \cdot \rangle$ das Skalarprodukt in X ist. Die Qualität der Entscheidungsfunktion wird durch den Gerneralisierungsfehler wiedergegeben und hängt von einer Reihe von Parametern ab, die durch verschiedene Algorithmen optimiert werden können. So kann die Maximierung der Trenngüte durch folgendes Optimierungsproblem beschrieben werden [DPHS98].

$$minimiere \qquad \frac{1}{2}||\vec{w}||^2 \tag{7}$$

$$mit \qquad Nebenbedingung \qquad y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1, \forall i \tag{8}$$

Die Generalisierungstheorie hat gezeigt, dass der Generalisierungsfehler mit zunehmender Trenngüte γ kleiner wird. Abbildung 5 zeigt zwei verschiedene Hyperebenen mit unterschiedlicher Trenngüte.

Natürlich kann man in der Realität nicht immer eine perfekte Trennhyperebene bestimmen, da die Trainingsdaten meistens verrauscht sind, d.h. im Allgemeinen nicht linear trennbar sind. Deshalb sucht man in der Realität nicht die Hyperebene mit maximalem Abstand, sondern eine Hyperebene (wie in Abbildung 6), die das Rauschen in der Trainingsmenge toleriert. Dies kann erreicht werden, indem man die Nebenbedingung abschwächt und eine Schlupfvariable ξ_i einführt:

$$y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1 - \xi_i, \forall i \tag{9}$$

$$\xi_i \geq 0, \forall i \tag{10}$$

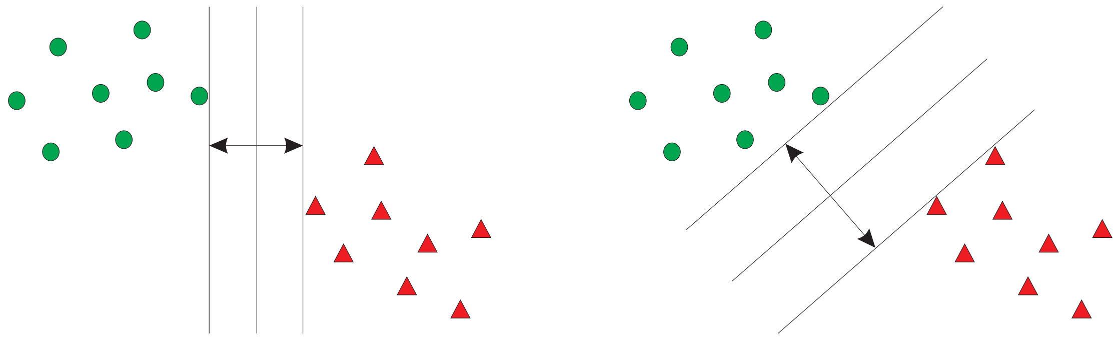

Abbildung 5: Bestimmung der Hyperebene mit maximaler Trenngüte

Durch die Abschwächung der Nebenbedingung erhält man bei der Optimierung sowohl gute als auch schlechte Hyperebenen. Bei schlechten Hyperebenen erhält man große ξ_i und bei guten Hyperebenen werden die Schlupfvariablen $\xi_i \approx 0$ sein. Aus diesem Grund versucht man, gleichzeitig die Norm des Gewichtsvektors $\vec{w}$ und die Norm der Schlupfvariablen zu minimieren.

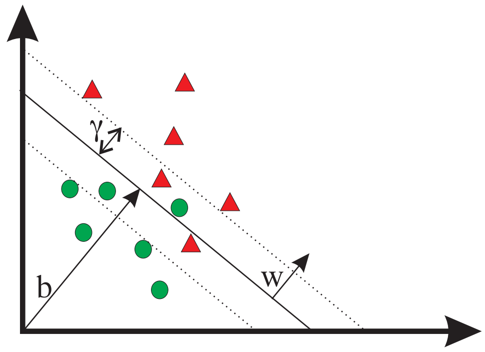

Abbildung 6: Hyperebene (w, b) mit Rauschen

3.3 Konvertierung von Dokumenten in Vektoren

Wie bereits auf Seite 17 angesprochen, müssen die Dokumente, die typischerweise aus einer Aneinanderreihung von einzelnen Wörtern bestehen, in eine Repräsentation überführt werden, die für den Lernalgorithmus und damit für die Klassifikation nutzbar ist. Ergebnisse aus dem Bereich Information Retrieval zeigen, dass einzelne Wörter als Darstellungseinheiten geeignet sind und ihre Reihenfolge für viele Anwendungszwecke nur eine untergeordnete Rolle spielt [Joa98].

Dies führt zu einer Attribut - Wert Darstellung eines Dokuments wie sie in Abbildung 7 dargestellt ist. Jedes Wort des Dokumentes stimmt mit einem Attribut überein und der Wert entspricht der Häufigkeit des Auftretens des Wortes im Dokument. Um unnötig lange Vektoren zu vermeiden, müssen die Wörter eine Mindestlänge aufweisen, um in die Attribut-

Wert-Darstellung aufgenommen zu werden. Dies hat den Vorteil, das kurze zufällige Buchstabenkombinationen, die sehr häufig bei der Konvertierung von PS (Post-Script) oder PDF (Portable Document Format) Dokumenten in ASCII- Text entstehen in der Attribut - Wert Darstellung ignoriert werden. Ebenfalls werden Stoppwörter nicht berücksichtigt, da diese für die Analyse des Textes keine besondere Bedeutung haben. Diese Form der Dokumentendarstellung beinhaltet jedoch einen Informationsverlust, da Ausdrücke, die aus verschiedenen Wörtern zusammengesetzt sind und eine bestimmte Bedeutung haben, in einzelne Worte zerlegt werden und damit auch die Bedeutung im zusammengesetzten Fall verloren geht.

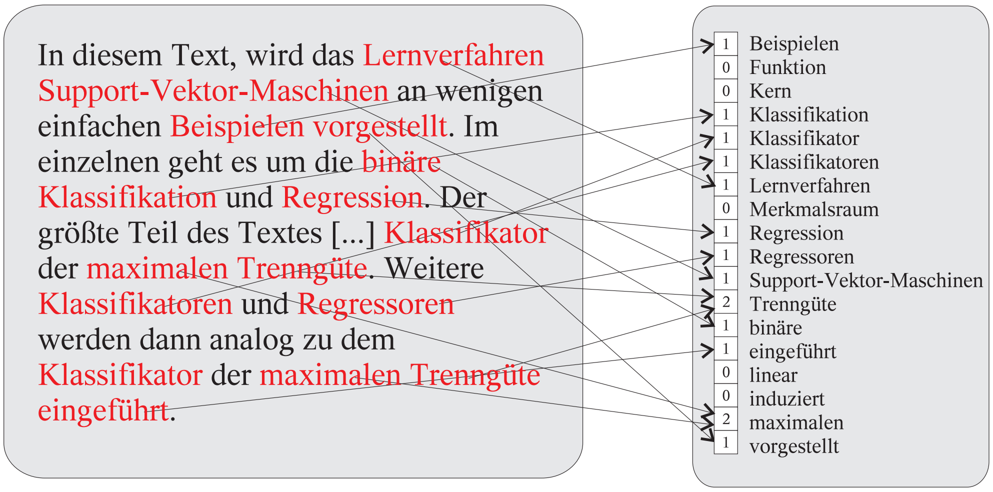

Abbildung 7: Attribute-Wert-Darstellung eines Dokumentes

Jedes Dokument wird so in eine Vektordarstellung $\vec{d_i} = (TF_1, TF_2, ..., TF_n)^T$ überführt, wobei TF_j der Term-Frequency $TF(w_j, d)$ entspricht. Auf diese Weise erhalten ähnliche Dokumente auch ähnliche Vektoren. Für die weitere Vorgehensweise werden zunächst die Begriffe Term-Frequency $TF(w_j, d)$ und Document-Frequency $DF(w_j)$ definiert.

Definition 3.1 (Term-Frequency $TF(w_j, d)$)
Unter der Term-Frequency $TF(w_j, d)$ versteht man die absolute Häufigkeit des Wortes w_j im Dokument d.

Definition 3.2 (Document-Frequency $DF(w_j)$)
Unter der Document-Frequency $DF(w_j)$ versteht man die absolute Häufigkeit des Wortes w_j in der Menge von Dokumenten $\{d_1, ...d_N\}$.

Bei der Berechnung der Vektoren sollen die einzelnen Wörter unterschiedlich stark gewichtet werden. Dabei sollen Wörter, die häufig und in vielen Dokumenten vorkommen schwächer ins Gewicht fallen als Wörter, die nur selten auftreten. Aus diesem Grund wird für jedes

Wort, das in der Lernmenge vorkommt die inverse document frequency $IDF(w_i)$ berechnet [Joa97]:

$$IDF(w_i) = \log(\frac{|D|}{DF(w_i)}) \quad (11)$$

Dabei bezeichnet $|D|$ die Anzahl der Dokumente. Mit den erzeugten Gewichten der einzelnen Wörter werden nun die Einträge in der Vektordarstellung aller Dokumente mutipliziert:

$$d^{(i)} = TF(w_i, d) \cdot IDF(w_i) \quad (12)$$

Abschließend werden die Vektoren noch auf eine Länge von 1 normiert, um von der unterschiedlichen Dokumentenlänge zu abstrahieren.

3.4 Bewertungsfunktionen

Um die Ergebnisse des Verfahres bewerten zu können, muss ein Maß für die Güte des Verfahrens definiert werden. Dabei muss sowohl die Korrektheit der Vorschläge als auch die Menge der richtigen Vorschläge betrachtet werden.

Mit Hilfe der Abbildung 8 sollen Funktionen für diese beiden Maße hergeleitet werden. Um Aussagen über die Korrektheit und Menge der richtigen Vorschläge treffen zu können, müssen alle Dokumente zunächst manuell klassifiziert werden. Anschließend werden alle Dokumente, die klassifiziert werden sollen, in zwei Mengen unterteilt. Zur Menge a gehören die Dokumente, die die Kriterien der Klasse, die untersucht werden soll, erfüllen. Alle anderen Dokumente gehören zur Menge b.

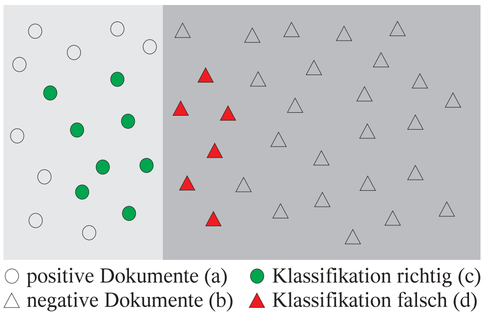

Abbildung 8: Ergebnis der Klassifikation für eine Klasse

Das Ergebnis der automatischen Klassifizierung ist in der Abbildung 8 dargestellt. Die ausgefüllten Kreise (Menge c) entsprechen den Dokumenten, die zu der überprüften Klasse gehören und bei der automatischen Klassifikation in diese Klasse eingeteilt wurden. Die ausgefüllten Dreiecke (Menge d) sind die Dokumente, die bei der automatischen Klassifikation der entsprechenden Klasse zugeordnet wurden aber, eigentlich nicht zu der Klasse gehören. Mit Hilfe der vier definierten Mengen können die Formeln zur Berechnung der Korrektheit und der Menge der richtigen Vorschläge definiert werden.

Die Korrektheit der Vorschläge, die sogenannte Precision, läßt sich wie in Formel 13 gezeigt, aus dem Verhältnis der richtigen Vorschläge (c) und der Gesamtzahl ($c + d$) der Vorschläge bestimmen.

$$Precision_{Class} = \frac{c}{c + d} \tag{13}$$

Die Menge der richtigen Vorschläge, der sogenannte Recall, berechnet sich aus dem Verhältnis der richtig klassifizierten Dokumente (c) und der Menge der Dokumente (a), die zu der entsprechenden Klasse gehören. (siehe Formel 14)

$$Recall_{Class} = \frac{c}{a} \tag{14}$$

4 Implementierung

4.1 Beschreibung der Softwarearchitektur im Projekt

4.1.1 Java2-Enterprise-Edition

Das Internet hat in den letzten Jahren stark an Bedeutung gewonnen. Inzwischen werden im Internet nicht nur Informationen angeboten, sondern auch komplexe Geschäftsprozesse abgewickelt. Aufgrund der rasanten Entwicklung auf diesem Gebiet haben die Entwickler immer weniger Zeit, eine zeitgemäße Applikation zu entwickeln. Um der Nachfrage gerecht zu werden, musste ein Standard geschaffen werden, um Applikationen möglichst einfach entwickeln zu können. Dabei muss auf komplexe Dinge wie Persistenz, Transaktionen und die Sicherheit geachtet werden. Diese Aufgabe gestaltet sich nicht einfach, da ein entsprechender Server alle Konfigurationsarbeiten für die Applikation übernehmen und die Entwicklung beschleunigen soll. Dabei muss ein entsprechender Applikationsserver die Skalierbarkeit und maximale Leistung gewährleisten.

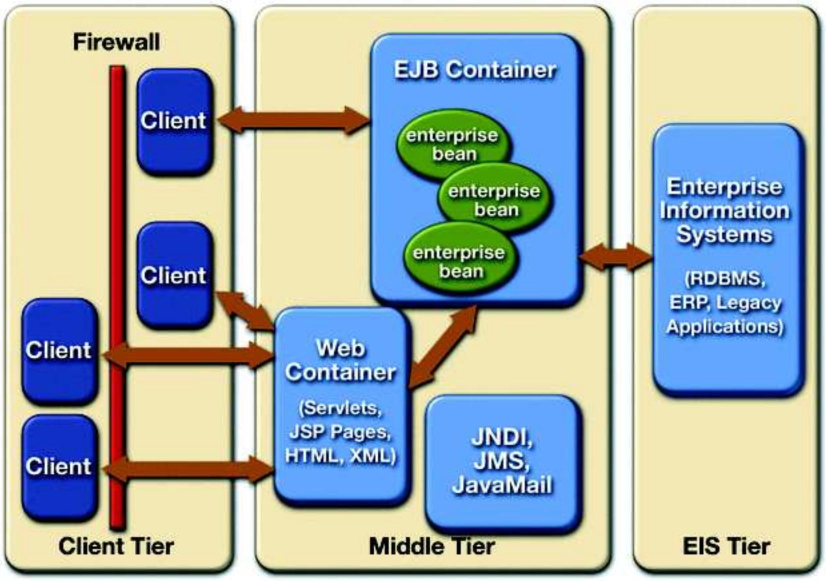

Abbildung 9: J2EE-Architekturüberblick

Eine Lösung stellte Sun Microsystems im Jahre 2000 mit der Java 2 Enterprise Edition (J2EE [10]) vor. J2EE stellt keinen Ersatz der Java 2 Standard Edition (J2SE [11]) sondern eine Erweiterung dar. Die Erweiterung beinhaltet Technologien, um verteilte Anwendungen zu realisieren. Dabei ist zu beachten, dass Sun Mircosystems mit J2EE nur eine Spezifikation

[11] `http://java.sun.com/j2se/`

der Schnittstellen vorgelegt hat. Die eigentliche Implementierung erfolgt durch kommerzielle Anbieter. Da alle Anbieter die gleiche Spezifikation implementieren, ist ein Austausch von Komponenten unterschiedlicher Anbieter theoretisch möglich. Zur Entwicklung und zum Testen bietet Sun Microsystems eine Referenzimplementierung, in der jedoch mit Absicht keinerlei Optimierungen bezüglich Leistung oder Skalierbarkeit enthalten sind [Sun03b].

Schicht- und Containerarchitektur

J2EE ist laut Spezifikation eine 3-Tier-Architektur, die sich wie in Abbildung 9 gezeigt in die Bereiche Client Tier, Middle Tier und Enterprise Information System (EIS) Tier aufteilt und auf Containern basiert. Um eine einfache Erweiterbarkeit und Wartbarkeit zu gewährleisten werden bei einer 3-Tier-Architektur, die Daten *(data layer)* von der Logik *(business layer)* und Darstellung *(presentation layer)* getrennt. Diese Aufteilung lässt sich jedoch noch beliebig verveinern. In der Regel wird eine Anwendung in die wesentlichen Bestandteile Benutzerschnittstelle *(user interface)*, Präsentationslogik *(presentation logic)*, Applikationslogik *(business logic)* und die Daten *(data layer)* aufgeteilt.

Als Laufzeitumgebung kommen auf der mittleren Ebene sogenannte *Container* zum Einsatz. Sie bieten bestimmte Dienste an, die Komponenten auf jeder J2EE-konformen Plattform vorraussetzen können. Im Middle Tier gibt es den *Web-Container*, der Servlets [12] und Java-Server-Pages [13]als Schnittstelle für die Verarbeitung von Client-Anfragen enthält, sowie den EJB-Container, der die Laufzeitumgebung für Enterprise-Java-Beans darstellt. In den die Logik der Applikation gekapselt ist. Neben diesen beiden Server-seitigen Containern gibt es noch den *Applet Container*, der Applets enthält und in der Regel in einen Browser integriert ist, und den *Application Client Container*, der eigenständige Java-Applikation beinhaltet.

Komponenten, Dienste und Kommunikation

J2EE umfasst eine Reihe von einzelnen Technologien, die in die Gruppen Komponenten, Services und Kommunikation unterteilt werden können. In den Komponenten wird die Logik der Applikation gekapselt, dabei unterscheidet J2EE zwei Komponenten:

- Web-Komponenten
  - Servlets
    Servlets sind Java-Programme, die die HTTP-Anfrage verarbeiten und eine entsprechende HTML-Seite generieren und als Antwort an den aufrufenden Client zurückschicken.
  - Java Server Pages (JSP)
    Java Server Pages sind HTML-Seiten mit eingebetten Java-Code zur dynamischen Generierung von Inhalt.

[12] `http://java.sun.com/products/servlet/`
[13] `http://java.sun.com/products/jsp/`

- Enterprise Java Beans
  Mit Enterprise Java Beans können Komponenten erstellt werden, die die eigentliche Geschäftslogik enthalten. Dabei werden drei Arten von EJB's unterschieden:
  - Entity EJB
    Entity EJB sind Objekte, die die Daten der EIS-Ebene (Geschäftsobjekte) dauerhaft repräsentieren.
  - Session EJB
    Session EJB realisieren geschäftliche Aktivität bzw. Prozesse. Es gibt zwei Arten von Session EJB:
    * „Stateless Session EJB“ halten ihren Zustand nur für die Dauer eines Methodenaufrufs.
    * „Stateful Session EJB“ halten ihren Zustand über mehrere Methodenaufrufe hinweg. Dabei ist eine Zuordnung der EJB-Instanz zum aufrufenden Client notwendig.
  - Message-driven EJB
    Message-driven EJB sind zustandslose Komponenten für die Verarbeitung von asynchronen Meldungen. Sie nehmen JMS-Messages (siehe Seite 28) an und verarbeiten diese Meldungen.

Dienste werden von den Server-seitigen Containern zur Verfügung gestellt. Auf diese Weise wird die Applikationslogik in der Komponententechnologie gekapselt. Folgende Services stehen zur Verfügung:

- JDBC
  Mit Hilfe der „Java Database Connectivity“ kann aus der Applikation heraus auf relationale Datenbanken zugegriffen werden.
- JAXP
  Das „Java API for XML Parsing“ ermöglicht die Verarbeitung von XML-Daten.
- JNDI
  Das „Java Naming and Directory Interface“ ermöglicht das Auffinden von Ressourcen.
- JAAS
  Der „Java Authentication und Authorization Service“ ermöglicht die Kontrolle über den Zugriff auf sensible Komponenten.
- JTA
  Das „Java Transaction API“ bietet eine umfangreiche Kontrolle über die Verwaltung von Transaktionen.

Folgende Technologien können bei der Kommunikation zwischen Client und Server oder auch zwischen verschiedenen Servern zum Einsatz kommen:

- HTTP
  Das zustandslose Hypertext-Transfer-Protocol (HTTP) wird vom Client verwendet, um Requests an den Server zu schicken.

- RMI
  Das Java Framework RMI (Remote Method Invocation) dient der Kommunikation von Objekten, die nicht in der selben virtuellen Maschine existieren.

- RMI-IIOP
  RMI-IIOP (RMI - Internet Inter ORB Protocol) ist eine Reimplementierung von RMI, die das CORBA kompatible IIOP als Protokoll verwendet.

- JMS
  Der Java Messaging Service (JMS) ermöglicht es, portable und nachrichtenbasierte Applikationen in Java zu erstellen.

- JavaMail
  JavaMail ermöglich das Empfangen und Versenden von E-Mails.

- XML
  Die eXtensible Markup Language (XML) dient zum Austausch von Daten.

Interaktionsstrukturen

Ein J2EE-System hat im Allgemeinen den folgenden Aufbau: Auf Client-Ebene werden bevorzugt Web-Browser, wo nötig auch Stand-Alone-Clients eingesetzt. Auf der mittleren Ebene wird die Geschäftslogik durch Komponenten in Web- und EJB-Containern realisiert. Auf EIS-Ebene kommen beispielsweise relationale Datenbanksysteme zum Einsatz. Es zeigt sich, dass die Halbwertszeit der Software mit steigender Entfernung von der EIS-Ebene rasch abnimmt. Aus diesem Grund empfiehlt sich auf Client-Ebene die Verwendung von Web-Browsern anstelle eigenständiger Applikationen, soweit dies möglich ist. Während eine Aktualisierung der Rendering-Engine der Browser nur selten notwendig sein dürfte, liegt im Falle von Stand-Alone-Clients naturgemäß mehr Funktionalität in der Client-Ebene, so dass diese von Änderungen am Gesamtsystem häufiger betroffen sein wird. In der Praxis kann die Systemarchitektur je nach Anforderung verschiedene Formen annehmen.

Abbildung 10 zeigt eine Mehr-Ebenen-Applikation. Der Browser als Client hat nur die Aufgabe, die direkte Benutzerinteraktion abzuwickeln. Dafür kommuniziert er, bevorzugt per HTTP, mit dem Web-Container in der mittleren Ebene. Die Web-Komponenten im Web-Container realisieren ausschließlich die internen Teile der Präsentationslogik. Sie nehmen HTTP-Requests des Browsers entgegen und beantworten diese mit dynamisch erstellten HTML-Seiten. Die eigentliche Geschäftslogik steckt in Enterprise JavaBeans, die im EJB-Container ablaufen. Sie steuern das gesamte System und interagieren dabei mit dem Web-Container einerseits und der Datenbank andererseits. So werden Datenzugriff und Interaktion mit dem Benutzer weitestgehend entkoppelt. Mehr-Ebenen-Applikationen weisen maximale

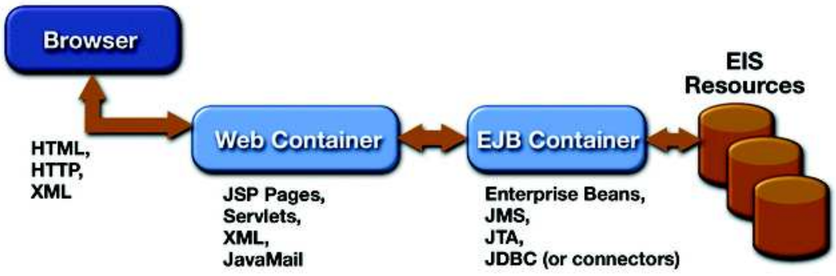

Abbildung 10: Mehr-Ebenen-Applikation

Mächtigkeit und Skalierbarkeit auf, sind jedoch entsprechend aufwändig umzusetzen.

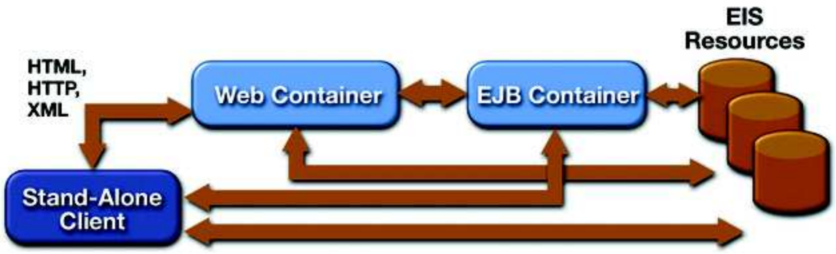

Abbildung 11: Standalone-Client-Architektur

Wenn auf Client-Ebene eine größere Funktionalität benötigt wird, kann eine Stand-Alone-Applikation eingestzt werden. Hieraus ergeben sich, wie in Abbildung 11 gezeigt, mehrere mögliche Systemstrukturen. Der Client kann unter Umgehung von Web-Komponenten direkt mit dem EJB-Container kommunizieren, normalerweise mit Hilfe von Remote-Method-Invocation (RMI). In diesem Fall übernimmt der Client die Aufgabe der Präsentation selbständig, die Geschäftslogik verbleibt jedoch auf der mittleren Ebene. Analog kann auch der EJB-Container umgangen werden. Dies ist der Fall, wenn der Client in der Lage ist, dynamische Web-Inhalte zu verarbeiten. Der Web-Container bietet dem Client dann Zugang zu diesen Inhalten, die auf der EIS-Ebene gelagert werden. Auch bei dieser Variante trägt die Client-Ebene die Präsentationslogik. Die Geschäftslogik kann je nach Komplexität im Web-Container oder auf der EIS-Ebene umgesetzt werden. Schließlich kann der Client auch direkt mit den Datenbanken der EIS-Ebene kommunizieren. Hierbei werden Präsentations- und Geschäftslogik, also die gesamte Funktionalität der mittleren Ebene, auf Client-Ebene untergebracht [Sun03a] [Hus01] [Deß02] [J2E02].

4.1.2 Übersicht über den J2EE-basierenden Implementierungsansatz

Zum Austausch der Daten zwischen den einzelnen Komponenten (z.B. Redaktionssystem) und der zentralen Datenverwaltungskomponente im Projekt META-AKAD wurde zunächt

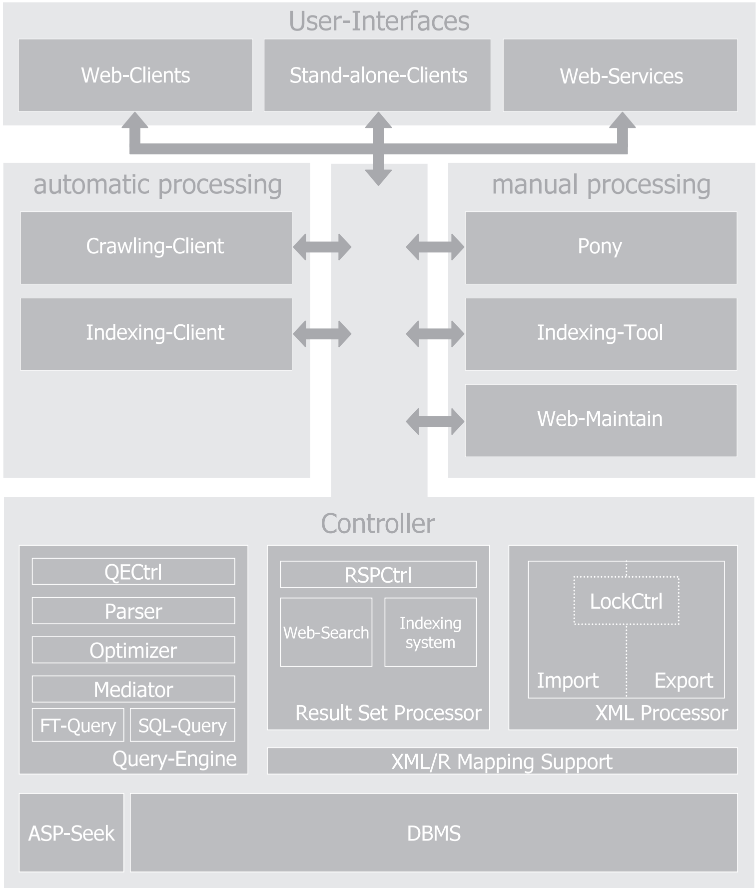

Abbildung 12: Systemarchitektur

die HTTP-Erweiterung WebDAV (Web-based Distributed Authoring and Versioning [14]) als Protokoll bzw. Schnittstelle gewählt. Dies hatte den Vorteil, schon sehr früh mit der Entwicklung einzelner Komponenten beginnen zu können, die auf der Funktionalität der Datenverwaltungskomponente aufbauen, ohne dass diese bereits vollständig realisiert sein muss.

Nachdem die HTTP-Erweiterung WebDAV als Schnittstelle zu der Datenverwaltungskomponente im Projekt META-AKAD den gewachsenen Ansprüchen nicht mehr gerecht wurde, wurde die Re-Implementierung mittels J2EE begonnen und dabei auf sinnvolle Modularität geachtet [FKL+02]. Abbildung 12 zeigt die Systemarchitektur, in der die verschiedenen Module dargestellt sind. Auf der Client-Seite existieren die Web-basierte Suche (Web-Search), ein Redaktionssystem zum manuellen Verarbeiten der Daten (Indexing-Tool) sowie der automatische Klassifizierungs- und Beschlagwortungs-Client (Indexing-Client). Die mittlere Schicht, in der mit EJBs die eigentliche Anwendungslogik der Datenverwaltungskomponente enthalten ist, gliedert sich in die drei Bereiche Anfrageverarbeitung (Query-Engine), Ergebnisverarbeitung (Result-Set-Processor) und XML-Dokumenten-Management (XML-

[14] http://www.webdav.org

Processor).

Bei der Anfrageverarbeitung kann die Suche über beliebige META-AKAD-Attribute sowie über den Volltext der Dokumente durchgeführt werden. Außerdem ist es möglich, statistische Daten über das Anfrageergebnis abzufragen. Zu diesen statistischen Daten gehören zum Beispiel die Anzahl der Treffer sowie die häufigsten Schlagworte und Klassifikationen. Bei der Darstellung des Anfrageergebnisses werden verschiedene Sortierkriterien unterstützt. Die Anfrage erfolgt auf zwei verschiendenen Datenquellen mit unterschiedlichem Datenmodell. Eine Datenquelle wird durch ein Volltextanfragesystem bereitgestellt. Die zweite Datenquelle ist ein relationales Datenbanksystem, in dem die Metadaten gespeichert sind. Durch einen sogenannten Mediator wird die Anfrage zerlegt und die beiden Ergebnismengen kombiniert. Da dieser Schritt sehr kostenintensiv ist und das Anfrageergebnis in unterschiedlichen Sortierungen benötigt wird, wird das Anfragergebnis vollständig materialisiert.

Die Anfrageverarbeitungskomponente konvertiert die interne Anfragerepräsentation, den sogenannten Query-Tree (siehe Abbildung 14), der im Rahmen eines Praktikums entwickelt wurde [FFW02], in SQL-Anfragen und Volltextsuchanfragen, führt diese aus, kombiniert das Ergebnis und speichert es in temporären Tabellen.

Die Arbeit der Ergebnisverwaltungskomponente (ResultSet-Processor) nutzt das materialisierte Anfrageergebnis als Grundlage für die Beantwortung der statistischen Anfragen. Diese Anfragen und die Sortierung der Anfrageergebnisse nach den verschiedenen Kriterien erfolgt ohne die wiederholte Ausführung der gesamten Anfrage.

Das Dokumenten-Management erfolgt durch den XML-Processor. Dieser kümmert sich um den Import und Export der XML-Daten. Außerdem ist er für die Sperrverwaltung bei der dokumentenorientierten Verarbeitung verantwortlich.

4.2 Beschreibung des entwickelten Indexierungsclients

Der Arbeitsablauf des Indexierungsclient lässt sich, wie in Abbildung 13 dargestellt, in vier Abschnitte einteilen: Datenimport, Vorbereitung der Titeldaten, Klassifikation und Datenexport. In den folgenden Abschnitten wird der genaue Ablauf der einzelnen Arbeitsschritte vorgestellt. Der Datenimport und Datenexport wird dabei einzeln betrachtet. Die Vorbereitung der Titeldaten sowie die eigentliche Klassifikation werden zusammen betrachtet, da es während der Entwicklung des Verfahrens Veränderungen an diesen beiden Teilen gab, die jeweils den Ablauf im anderen Teil beeinflusst haben.
Zum Beginn der Implementierung wurde überlegt, ob die Indexierungskomponente nur den Dokumenttitel oder auch den Dokumentvolltext verarbeiten soll. Die Verarbeitung des Dokumentvolltextes hätte zur Folge, dass Dokumente, die einen weniger aussagekräftigen Titel haben, besser klassifiziert werden könnten. Zunächst wurden in der Implementierungsphase beide Ansätze parallel verfolgt. Es stellte sich jedoch heraus, das mit der Integration des

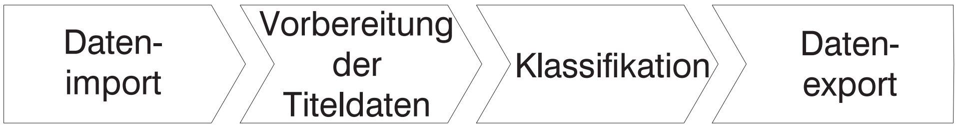

Abbildung 13: Arbeitsablauf

Volltextes eine Reihe von Problemen verbunden sind:

- Datenvolumen
  Bei der Einbeziehung der Volltexte in den Analyseprozess müssten alle relevanten Dokumente aus dem Internet heruntergeladen und (zumindest temporär) zwischengespeichert werden. Dabei würde eine recht große Datenmenge anfallen. Allein die 673 Testdokumente, die während der Entwicklungsphase genutzt wurden, hatten zusammen ein Volumen von mehr als 10 GB.

- Konvertierung der Dokumente in Texte
  Um die Volltexte in den Analyseprozess integrieren zu können, muss das entsprechende Dokument erst in eine reine Textform konvertiert werden. Für die Dokumentformate PS (PostScript), PDF (Portable Document Format) und HTML (HyperText Markup Language) gibt es frei verfügbare und kostenlose Konvertierungsprogramme. Es gibt jedoch auch Dokumentformate, die sich nicht so leicht in Textform konvertieren lassen (z.B. Microsoft Word). Auch die Konvertierung von PS und PDF Dokumenten läuft nicht immer problemlos ab. Einige Dokumente enthalten nach der Konvertierung keine sinnvollen Textfragmente mehr. In der Testphase kam es auch vor, dass die entsprechenden Konvertierungsprogramme nicht terminierten.

- Gefährdung der Homogenität der Datenbasis
  Durch die Einbeziehung des Volltexts könnte die Homogenität der im Verfahren verwendeten Daten nicht mehr gewährleistet sein. Im Projekt werden die verschiedensten Dokumenttypen gesammelt. Zum Beispiel ist die verwendete Wortmenge in einem Übungsblatt eine andere, als in einem Vorlesungsskript, das sich mit dem gleichen Thema beschäftigt. Die Titel von Vorlesungskripten und Übungen unterscheiden sich nicht so stark. Eine Einbeziehung des Volltextes in den Analyseprozess könnte das Ergebnis der automatischen Indexierung auch verschlechtern.

- Programmlaufzeit
  Da die Dokumente zunächst alle in eine Textform konvertiert werden müssen, dann eine wesentlich größere Datenmenge in Vektoren konvertiert werden muss und zuletzt auch das Support-Vector-Machine Verfahren aufgrund der höheren Vektorraumdimension eine längere Laufzeit benötigt, hätte die Integration der Volltexte eine wesentlich längere Programmlaufzeit zur Folge. So könnte sich der Indexierungsvorgang, der mit den derzeitigen Testdaten weniger als eine Minute dauert, über mehrere Stunden erstrecken.

Aufgrund dieser Probleme und möglicher Gefahren wurde der Ansatz nicht weiterverfolgt. Wenn zu den einzelnen Dokumenten eine textuelle Zusammenfassung vorliegen würde, könnte mit einer entsprechenden Integration der Zusammenfassung in den Analyseprozess sicherlich die Qualität des Indexierungsverfahren gesteigert werden, da die oben beschriebenen Probleme und Gefahren dabei nicht bestehen oder nicht so stark ins Gewicht fallen würden.

4.2.1 Datenimport

Bevor der Ablauf des Datenimports vorgestellt wird, sollte zunächst einmal geklärt werden, welche Daten von der Indexierungskomponente verarbeitet werden. Zur Klassifikation und Beschlagwortung mit Hilfe einer Support-Vektor-Maschine werden Dokumente benötigt, deren Klassifikationen und Schlagworte bereits intellektuell festgelegt wurden. Mit diesen Dokumenten lernt das Verfahren, welche Dokumentart welche Klassifikationen und welches Schlagwort zugeordnet bekommt. Aus diesem Grund wird diese Menge auch Lernmenge genannt. Tabelle 2 zeigt eine Auswahl von Datensätzen mit Titel, Klassifikation sowie Schlagwörtern.

Titel	Klassifikationen	Schlagwörter
Codierungstheorie und Kryptographie	SK 170 SK 880	Codierungstheorie Krytologie
Numerik partieller Differentialgleichungen	SK 540 SK 920	Numerische Verfahren partielle Differentialgleichung
Vorlesung Stochastik	QH 440 SK 800 SK 840	Statistik Stochastik Wahrscheinlichkeitstheorie

Tabelle 2: Beispieldaten

Die verwendete Lernmenge enthält ca. 700 Dokumente, die sich auf 100 Klassen verteilen und 250 verschiedene Schlagwörter besitzen. Die Dokumentenmenge, mit der das Verfahren getestet wurde, umfasst ca. 180 Dokumente, die zunächst intellektuell erschlossen wurden, um die durch das Verfahren automatisch vergebenen Klassifikationen und Schlagwörter auf ihre Korrektheit zu überprüfen.

Die Lerndokumente sind ebenso wie die Dokumente, die noch keine Klassifikation und noch kein Schlagwort besitzen, in der zentralen META-AKAD-Datenverwaltungskomponente gespeichert und sind mit Hilfe des Verwaltungsattributs *learningresource.indexclient.learningset* als Dokumente der Lernmenge gekennzeichnet.

Anfragen an die Datenverwaltungskomponente werden mit einer speziell entwickelten Anfragesprache durchgeführt. Eine Anfrage besteht aus einem Baum von Java-Objekten, im dem die Blätter Bedingungen für einzelne Metadatenfelder und die internen Knoten boolesche Verknüpfungen der Bedingungen enthalten.

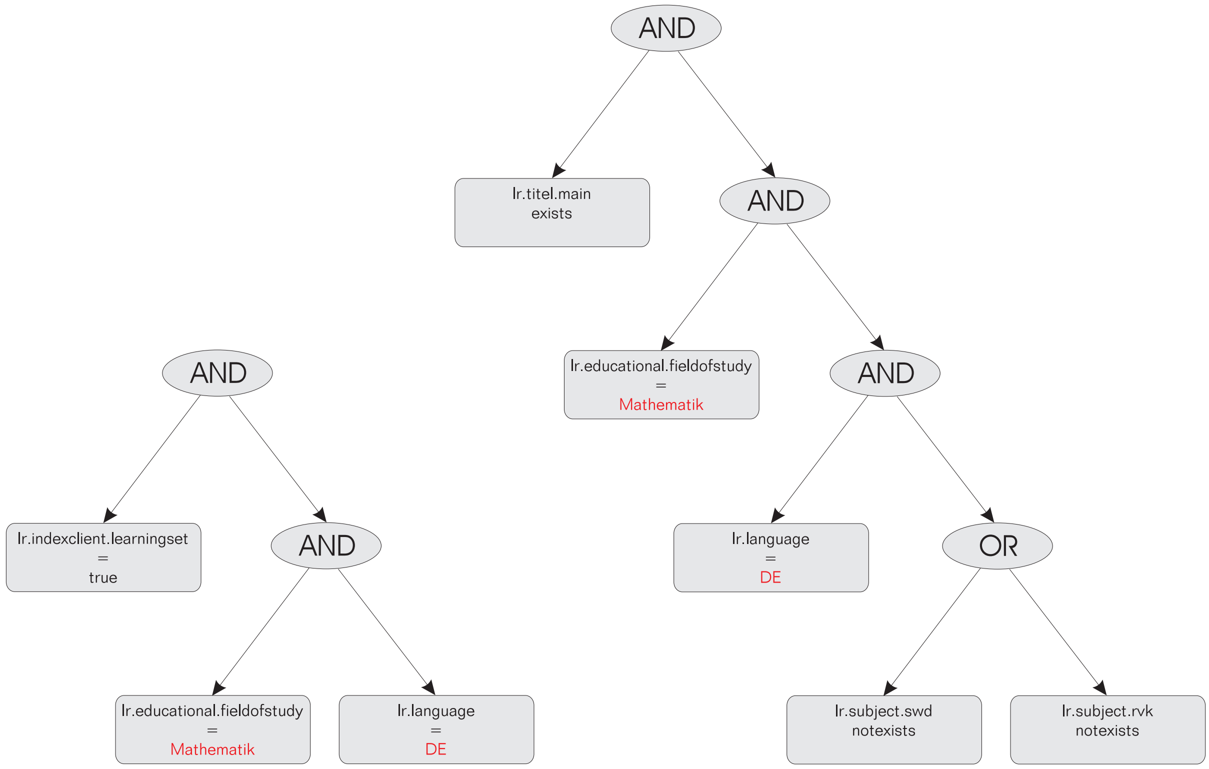

Abbildung 14: Anfragebäume

Der Import der Lernmenge und der Import der Dokumente, die klassifiziert und beschlagwortet werden sollen, unterscheidet sich nur in der Struktur des Anfragebaums. In Abbildung 14 sind die entsprechenden Anfragebäume für deutschsprachige Dokumente aus dem Fachgebiet Mathematik dargestellt. Der linke Anfragebaum zeigt die Anfrage, um die Dokumente der Lernmenge zu erhalten, die der deutschen Sprache und dem Fachbereich Mathematik angehören. Mit dem rechten Baum werden die deutschsprachigen Dokumente des Fachbereichs Mathematik angefragt, die keine Klassifikation und kein Schlagwort aber einen intellektuell kontrollierten Titel besitzen.

Ein Anfragebaum wird an die Query-Engine übergeben. Die Query-Engine verarbeitet die Anfrage und gibt ein sogenanntes Result-Set-Handle zurück. Mit diesem Result-Set-Handle kann beim Result-Set-Processor eine XML-Repräsentation des materialisierten Anfrageergebnisses angefordert werden. Die XML-Dokumente entsprechen der Dokumenttypdefinition die im Listing 1 dargestellt ist.

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT metabase (learningresource*)>
<!ELEMENT learningresource (title,subject*)>
<!ATTLIST learningresource guID CDATA #REQUIRED>
<!ELEMENT title  (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ATTLIST subject scheme   CDATA #REQUIRED
                  notation CDATA #IMPLIED
                  origin   CDATA #REQUIRED>
```

Listing 1: Dokumenttypdefinition für den XML-Import

Die XML-Dokumente werden durch das Data-Binding-Produkt Castor [15] und einem im Listing 3 (Seite 56) abgebildeten Mapping in eine entsprechende Java-Objektstruktur überführt. Durch die Tatsache, dass der Datenaustausch auf den XML-Dokumenten basiert, ist das Indexierungprogramm robuster gegenüber Veränderungen an der Datenverwaltungskomponente. Außerdem kann das Programm wesentlich leichter an den Einsatz in anderen Umfeldern angepasst werden.

Auf die hier beschriebene Art und Weise werden nacheinander die Lernmenge und die zu erschließende Dokumentmenge importiert.

4.2.2 Vorbereitung der Titeldaten und Klassifikation

Um Dokumente aufgrund ihres Titels klassifizieren zu können, müssen die Titel der Dokumente zunächst in Vektoren transformiert werden (siehe 3.3). Dabei wird zunächst den einzelnen Wörtern der Titel in der Lernmenge eine für das Wort eindeutige ID zugeordnet und die Häufigkeit des Wortes in allen Dokumenten gespeichert. Bei der Verarbeitung der einzelnen Wörter im Titel werden alle Buchstaben dieser Wörter in Kleinbuchstaben konvertiert. In diesem Verarbeitungsschritt wird für jedes Dokument der Lernmenge ein Vektor mit Tupeln von Wort-ID und Häufigkeit des Wortes im Titel angelegt. Dann wird mit Hilfe der Formel 11 die inverse Dokumenthäufigkeit der Wörter in den Titeln der Lernmenge berechnet. Für die Dokumente, die klassifiziert und beschlagwortet werden sollen, wird ebenfalls ein solcher Vektor mit Tupeln von Wort-ID und Häufigkeit gebildet. Dabei werden jedoch nur Wörter berücksichtigt, die auch in der Lernmenge vorkommen, da es nur mit diesen Wörtern möglich ist, Ähnlichkeiten zu Titeln der Lernmenge festzustellen. Die Vektoren, in denen die Häufigkeit der einzelnen Wörter festgehalten ist, werden dann mit der inversen Dokumenthäufigkeit des entsprechenden Wortes multipliziert und anschließend auf die

[15] http://www.castor.org

Länge 1 normiert.

Wie in der Tabelle 2 zu sehen ist, werden einem Titel nicht nur eine Klassifikation oder ein Schlagwort zugeordnet, sondern oft auch mehrere. Beobachtungen haben gezeigt, dass Klassifikationen oft in bestimmten Kombinationen auftauchen. Dies führte zu der Vermutung, dass es Zusammenhänge zwischen einzelnen Klassifikationen gibt. Aus diesem Grund wurde jede Kombination von Klassifikationen, die in der Lernmenge auftaucht, durch eine neue virtuelle Klasse ersetzt. Mit den unterschiedlichen Kombinationen von Schlagwörtern wurde analog verfahren.

Nun muss ein neu zu klassifizierendes Dokument nur noch einer solchen virtuellen Klasse zugeordnet werden. Diese Zuordnung soll wie bereits in Kapitel 3.2 beschrieben mit Hilfe einer Support-Vektor-Maschine durchgeführt werden.

Es gibt eine Reihe von frei verfügbaren Implementierungen einer Support-Vector-Machine in den Programmiersprachen C++ und Java. Die meisten Implementierungen unterstützen allerdings nur eine binäre Klassifikation, d.h. sie können entscheiden, ob ein Objekt zu einer Klasse gehört oder nicht. Für den oben beschriebenen Ansatz wird jedoch eine Implementierung benötigt, die in der Lage ist, mehrere Klassen zu erlernen und ein Objekt dann genau einer dieser Klassen zuzuordnen. Eine Implementierung in Java, die diesen Anforderungen gerecht wird ist die *libsvm* [16] von Chih-Chung Chang und Chih-Jen Lin.

Der erste Testlauf des Programms lieferte die Ergebnisse, die in den Tabelle 3 und 4 erfasst sind. Im nächsten Schritt wurde untersucht, ob die Zusammenfassung der unterschiedlichen Klassifikationen eines Dokumentes zu einer virtuellen Klasse sinnvoll ist. Aus diesem Grund wurde das Verfahren so geändert, dass zunächst eine Liste aller vergebener Klassifikationen und Schlagworte erstellt wurde. Für jede dieser Klassen und für jedes Schlagwort wurde ein entsprechendes Modell der Lernmenge erstellt, d.h. es wurde für jedes Dokument in der Lernmenge geschaut, ob das Dokument zu der entsprechenden Klasse gehört oder nicht. Auf diese Art und Weise wurde die Lernmenge in eine Positiv- und Negativmenge geteilt. Aus diesen beiden Mengen wurde dann ein Modell erstellt. Dann wurde für jedes Dokument der Menge, die klassifiziert werden soll, entschieden, ob dieses Dokument eher der Positiv- oder der Negativmenge angehört. Wenn das Dokument besser zur Positiv- als zur Negativmenge passt, wird die entsprechende Klassifikation zum Dokument hinzufügt.

Die Ergebnisse der Klassifizierung nach dem Modell der binären Entscheidung im Vergleich zum Modell der virtuellen Klassen sind in den Tabellen 5 und 6 dargestellt. Nach dem Vergleich der beiden Ergebnisse stellte sich die Frage, ob es sich bei den richtig oder falsch zu den einzelnen Klassifikationen und Schlagwörtern zugeordneten Dokumenten um dieselben Dokumente handelt oder nicht, d.h. wie wirkt sich eine Ausführung der beiden Verfahren nacheinander auf das Ergebnis aus. Das Ergebnis der Nacheinanderausführung der beiden

[16] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`

RVK	Lernmenge	Testmenge	richtig	falsch	Precision	Recall
SK 110	55	18	13	12	52.0	72.2
SK 130	27	2	0	1	0.0	0.0
SK 150	9	3	2	0	100.0	66.7
SK 170	16	4	3	0	100.0	75.0
SK 180	18	1	1	0	100.0	100.0
SK 200	30	11	10	2	83.3	90.9
SK 220	56	26	14	0	100.0	53.8
SK 230	7	5	2	0	100.0	40.0
SK 240	14	4	3	0	100.0	75.0
SK 260	11	1	0	0	0.0	0.0
SK 280	7	2	2	2	50.0	100.0
SK 340	4	1	0	0	0.0	0.0
SK 370	14	3	2	0	100.0	66.7
SK 380	32	10	5	1	83.3	50.0
SK 399	11	1	0	0	0.0	0.0
SK 400	93	56	32	0	100.0	57.1
SK 420	5	1	0	0	0.0	0.0
SK 430	11	2	0	0	0.0	0.0
SK 450	2	2	0	0	0.0	0.0
SK 470	1	2	0	0	0.0	0.0
SK 500	12	1	0	0	0.0	0.0
SK 520	17	5	2	0	100.0	40.0
SK 540	10	2	2	0	100.0	100.0
SK 600	10	5	5	0	100.0	100.0
SK 620	3	1	0	0	0.0	0.0
SK 700	19	3	2	0	100.0	66.7
SK 800	35	12	6	1	85.7	50.0
SK 820	9	2	2	0	100.0	100.0
SK 840	38	13	8	17	32.0	61.5
SK 860	2	1	0	0	0.0	0.0
SK 870	27	2	1	0	100.0	50.0
SK 880	5	2	2	0	100.0	100.0
SK 890	14	4	2	0	100.0	50.0
SK 900	53	8	8	2	80.0	100.0
SK 910	4	1	0	0	0.0	0.0
SK 920	12	2	1	0	100.0	50.0
SK 950	103	41	29	2	93.5	70.7
SK 970	3	1	0	0	0.0	0.0
SK 980	5	2	0	0	0.0	0.0
SM 700	13	6	2	0	100.0	33.3
SM 730	7	1	0	0	0.0	0.0
SH 500	2	2	0	0	0.0	0.0
ST 601	18	5	1	0	100.0	20.0
QH 440	11	5	5	1	83.3	100.0
UK 1200	1	1	0	0	0.0	0.0
Summe:	856	283	167	41	80.3	59.0

Tabelle 3: Klassifikationsergebnis: ”virtuelle Klasse”

Schlagwort	Lern-menge	Test-menge	richtig	falsch	Precision	Recall
Algebra	31	11	10	1	90.9	90.9
Algebraische Geometrie	4	2	1	0	100.0	50.0
Analysis	87	55	32	0	100.0	58.2
Analytische Geometrie	20	8	3	0	100.0	37.5
Computeralgebra	4	2	1	0	100.0	50.0
Differentialgeometrie	11	2	2	0	100.0	100.0
Differentialgleichung	11	2	0	0	0.0	0.0
Diskrete Mathematik	1	2	0	0	0.0	0.0
Elliptische Kurve	5	1	1	0	100.0	100.0
Fehlerrechnung	1	1	0	0	0.0	0.0
Funktionalanalysis	9	5	5	0	100.0	100.0
Funktionentheorie	20	4	2	0	100.0	50.0
Geometrie	5	1	1	2	33.3	100.0
Gewöhnliche Differentialgleichung	16	5	2	0	100.0	40.0
Graphentheorie	8	1	0	0	0.0	0.0
Höhere Mathematik	83	32	27	18	60.0	84.4
Integrationstheorie	4	2	0	0	0.0	0.0
Kombinatorik	6	1	1	0	100.0	100.0
Kommutative Algebra	3	2	2	0	100.0	100.0
Kryptologie	6	2	2	0	100.0	100.0
Lineare Algebra	54	25	15	0	100.0	60.0
Maple	4	1	0	0	0.0	0.0
Mathematik	29	0	0	7	0.0	0.0
Mathematikunterricht	5	6	1	0	100.0	16.7
Mathematische Logik	17	2	0	0	0.0	0.0
Mathematische Methode	12	1	0	0	0.0	0.0
Maßtheorie	3	1	0	0	0.0	0.0
Matlab	5	2	0	0	0.0	0.0
Mengenlehre	9	3	2	0	100.0	66.7
Numerische Mathematik	52	8	8	1	88.9	100.0
Numerisches Verfahren	13	1	1	0	100.0	100.0
Optimierung	22	2	1	0	100.0	50.0
Partielle Differentialgleichung	9	2	2	0	100.0	100.0
Spieltheorie	2	1	0	0	0.0	0.0
Statistik	32	10	8	3	72.7	80.0
Stochastik	18	4	4	1	80.0	100.0
Stochastischer Prozess	4	1	1	0	100.0	100.0
Topologie	7	2	2	2	50.0	100.0
Wahrscheinlichkeitstheorie	18	11	2	0	100.0	18.2
Wissenschaftliches Rechnen	4	1	1	0	100.0	100.0
Zahlentheorie	8	1	1	0	100.0	100.0
Summe:	662	226	141	35	80.1	62.4

Tabelle 4: Beschlagwortungsergebnis: ”virtuelle Klasse”

Verfahren im Vergleich zu den beiden einzelnen Verfahren ist in den Tabellen 7 und 8 abgebildet. Bei Betrachtung der Ergebnisse fällt auf, das die Zahl der richtig bzw. falsch zugeordneten Klassen bei der Kombination der beiden Verfahren nur an 3 Stellen größer ist als das Maximum aus den Verfahren, wenn sie alleine durchgeführt werden. Dies lässt stark vermuten, dass es sich bei den richtig klassifizierten Dokumenten um die gleichen Dokumente handelt. Aus diesem Grund ist die Kombination der beiden Verfahren nicht sinnvoll.

Das Verfahren der binären Entscheidung für jede Klasse ist dem Verfahren der virtuellen Klassen vorzuziehen, da es eine wesentlich höhere Precision-Werte aufweist. Durch die Steigerung der Precision sinkt jedoch der Recall. Da es in der zu entwickelnden Anwendung vor allem darauf ankommt, mit möglichst wenig manueller Nachbearbeitung eine Klassifikaton zu finden, soll die Steigerung der Precision im Vordergrund stehen.

Bei der Benutzung einer bestehenden Support-Vector-Machinen-Implementierung ist es nur sehr schwer möglich, etwas an dem Prinzip der Klassifizierung zu ändern, um die Ergebnisse zu verbessern. Deshalb konzentrierten sich die weiteren Bemühungen auf eine Veränderung der Eingangsvektoren. Die Eliminierung von Stoppwörtern, d.h. Wörter, die keine Aussagekraft über den Inhalt des Dokuments haben, hat nur eine unwesentliche Steigerung der Ergebnisse ergeben. Dieser nur schwache Erfolg lässt sich dadurch erklären, dass Stoppwörter häufig in Titeln auftauchen und bereits ihre Bedeutung für das Verfahren durch die Berechnung der inversen Dokumentenhäufigkeit verlieren.

Es gibt jedoch eine weitere Möglichkeit, um einen Titel besser durch einen Vektoren beschreiben zu können. Häufig tauchen in Titeln zusammengesetzte Wörter, sogenannte Komposita, in verschiedenen Zusammensetzungen auf. Da die verschieden zusammengesetzten Wörter in der Regel nicht so häufig auftreten wie ihre einzelnen Bestandteile ist ihre Wirkung bei der Klassifizierung und Beschlagwortung geringer, als wenn die zusammengesetzen Wörter in die einzelnen Bestandteile zerlegt wären.

Aus diesem Grund wurde nach einer Methode gesucht, um zusammengesetzte Wörter (Komposita) in ihre einzelnen Bestandteile zu zerlegen. Im Bereich des Information-Retrieval wird die sogenannte Kompositazerlegung mit Hilfe von morphologischen Werkzeugen durchgeführt. Diese Werkzeuge benötigen spezielle Lexika, die in der Regel auf das Vokabular der Anwendungsdomäne angepasst werden müssen. Die Zerlegung eines Wortes ist jedoch nur dann sinnvoll, wenn die einzelnen Bestandteile des Wortes bereits in der Menge aller Wörter, die in den Titeldaten vorkommen, enthalten sind, um deren Wirkung zu verstärken. Daher ist der Einsatz eines aufwändig anzupassenden morphologischen Werkzeuges nicht sinnvoll. Es gibt eine wesentlich einfachere Strategie um eine Menge von Wörtern in einzelne Bestandteile zu zerlegen. Bei dieser Methode findet jedoch keine kommplette Zerlegung der Komposita in ihre Einzelbestandteile statt. Dies ist in diesem Anwendungsfall auch nicht unbedingt notwendig.

Bei der Zerlegung der Wörter muss auf sehr effektive Weise untersucht werden, ob ein be-

	virtuelle Klasse				binäre Entscheidung			
RVK	richtig	falsch	Precision	Recall	richtig	falsch	Precision	Recall
SK 110	13	12	52.0	72.2	8	9	47.1	44.4
SK 130	0	1	0.0	0.0	0	0	0.0	0.0
SK 150	2	0	100.0	66.7	2	0	100.0	66.7
SK 170	3	0	100.0	75.0	3	0	100.0	75.0
SK 180	1	0	100.0	100.0	1	0	100.0	100.0
SK 200	10	2	83.3	90.9	10	1	90.9	90.9
SK 220	14	0	100.0	53.8	13	0	100.0	50.0
SK 230	2	0	100.0	40.0	2	0	100.0	40.0
SK 240	3	0	100.0	75.0	2	0	100.0	50.0
SK 260	0	0	0.0	0.0	0	0	0.0	0.0
SK 280	2	2	50.0	100.0	2	2	50.0	100.0
SK 340	0	0	0.0	0.0	0	0	0.0	0.0
SK 370	2	0	100.0	66.7	2	0	100.0	66.7
SK 380	5	1	83.3	50.0	5	1	83.3	50.0
SK 399	0	0	0.0	0.0	0	0	0.0	0.0
SK 400	32	0	100.0	57.1	29	0	100.0	51.8
SK 420	0	0	0.0	0.0	0	0	0.0	0.0
SK 430	0	0	0.0	0.0	0	0	0.0	0.0
SK 450	0	0	0.0	0.0	0	0	0.0	0.0
SK 470	0	0	0.0	0.0	0	0	0.0	0.0
SK 500	0	0	0.0	0.0	0	0	0.0	0.0
SK 520	2	0	100.0	40.0	2	0	100.0	40.0
SK 540	2	0	100.0	100.0	2	0	100.0	100.0
SK 600	5	0	100.0	100.0	5	0	100.0	100.0
SK 620	0	0	0.0	0.0	0	0	0.0	0.0
SK 700	2	0	100.0	66.7	2	0	100.0	66.7
SK 800	6	1	85.7	50.0	6	1	85.7	50.0
SK 820	2	0	100.0	100.0	2	0	100.0	100.0
SK 840	8	17	32.0	61.5	8	0	100.0	61.5
SK 860	0	0	0.0	0.0	0	0	0.0	0.0
SK 870	1	0	100.0	50.0	1	0	100.0	50.0
SK 880	2	0	100.0	100.0	2	0	100.0	100.0
SK 890	2	0	100.0	50.0	0	0	0.0	0.0
SK 900	8	2	80.0	100.0	7	1	87.5	87.5
SK 910	0	0	0.0	0.0	0	0	0.0	0.0
SK 920	1	0	100.0	50.0	0	0	0.0	0.0
SK 950	29	2	93.5	70.7	27	2	93.1	65.9
SK 970	0	0	0.0	0.0	0	0	0.0	0.0
SK 980	0	0	0.0	0.0	0	0	0.0	0.0
SM 700	2	0	100.0	33.3	0	0	0.0	0.0
SM 730	0	0	0.0	0.0	0	0	0.0	0.0
SH 500	0	0	0.0	0.0	0	0	0.0	0.0
ST 601	1	0	100.0	20.0	2	0	100.0	40.0
QH 440	5	1	83.3	100.0	4	1	89.0	80.0
UK 1200	0	0	0.0	0.0	0	0	0.0	0.0
Summe:	167	41	80.3	59.0	149	18	89.2	52.7

Tabelle 5: Klassifikationsergebnis: ”virtuelle Klasse” und ”binäre Entscheidung”

Schlagwort	virtuelle Klasse				binäre Entscheidung			
	richtig	falsch	Precision	Recall	richtig	falsch	Precision	Recall
Algebra	10	1	90.9	90.9	10	1	90.9	90.9
Algebraische Geometrie	1	0	100.0	50.0	0	0	0.0	0.0
Analysis	32	0	100.0	58.2	33	1	97.1	60.0
Analytische Geometrie	3	0	100.0	37.5	3	2	60.0	37.5
Computeralgebra	1	0	100.0	50.0	1	0	100.0	50.0
Differentialgeometrie	2	0	100.0	100.0	2	0	100.0	100.0
Differentialgleichung	0	0	0.0	0.0	0	0	0.0	0.0
Diskrete Mathematik	0	0	0.0	0.0	0	0	0.0	0.0
Elliptische Kurve	1	0	100.0	100.0	1	0	100.0	100.0
Fehlerrechnung	0	0	0.0	0.0	0	0	0.0	0.0
Funktionalanalysis	5	0	100.0	100.0	5	0	100.0	100.0
Funktionentheorie	2	0	100.0	50.0	2	0	100.0	50.0
Geometrie	1	2	33.3	100.0	0	0	0.0	0.0
Gewöhnliche Differentialgleichung	2	0	100.0	40.0	2	0	100.0	40.0
Graphentheorie	0	0	0.0	0.0	0	0	0.0	0.0
Höhere Mathematik	27	18	60.0	84.4	23	4	85.2	71.9
Integrationstheorie	0	0	0.0	0.0	0	0	0.0	0.0
Kombinatorik	1	0	100.0	100.0	1	0	100.0	100.0
Kommutative Algebra	2	0	100.0	100.0	2	0	100.0	100.0
Kryptologie	2	0	100.0	100.0	2	0	100.0	100.0
Lineare Algebra	15	0	100.0	60.0	13	0	100.0	52.0
Maple	0	0	0.0	0.0	0	0	0.0	0.0
Mathematik	0	7	0.0	0.0	0	0	0.0	0.0
Mathematikunterricht	1	0	100.0	16.7	0	0	0.0	0.0
Mathematische Logik	0	0	0.0	0.0	0	0	0.0	0.0
Mathematische Methode	0	0	0.0	0.0	0	1	0.0	0.0
Maßtheorie	0	0	0.0	0.0	0	0	0.0	0.0
Matlab	0	0	0.0	0.0	1	0	100.0	50.0
Mengenlehre	2	0	100.0	66.7	2	0	100.0	66.7
Numerische Mathematik	8	1	88.9	100.0	8	0	100.0	100.0
Numerisches Verfahren	1	0	100.0	100.0	0	0	0.0	0.0
Optimierung	1	0	100.0	50.0	1	0	100.0	50.0
Partielle Differentialgleichung	2	0	100.0	100.0	2	0	100.0	100.0
Physik	0	0	0.0	0.0	0	1	0.0	0.0
Spieltheorie	0	0	0.0	0.0	0	0	0.0	0.0
Statistik	8	3	72.7	80.0	8	0	100.0	80.0
Stochastik	4	1	80.0	100.0	4	2	66.7	100.0
Stochastischer Prozess	1	0	100.0	100.0	1	0	100.0	100.0
Topologie	2	2	50.0	100.0	2	2	50.0	100.0
Wahrscheinlichkeitstheorie	2	0	100.0	18.2	1	0	100.0	9.1
Wissenschaftliches Rechnen	1	0	100.0	100.0	1	0	100.0	100.0
Zahlentheorie	1	0	100.0	100.0	1	0	100.0	100.0
Summe:	141	35	80.1	62.4	132	14	90.4	58.4

Tabelle 6: Beschlagwortungsergebnis: ”virtuelle Klasse” und ”binäre Entscheidung”

RVK	virtuelle Klasse richtig	virtuelle Klasse falsch	binäre Entscheidung richtig	binäre Entscheidung falsch	Kombination richtig	Kombination falsch	Kombination Precision	Kombination Recall
SK 110	13	12	8	9	13	**13**	50.0	72.2
SK 130	0	1	0	0	0	1	0.0	0.0
SK 150	2	0	2	0	2	0	100.0	66.7
SK 170	3	0	3	0	3	0	100.0	75.0
SK 180	1	0	1	0	1	0	100.0	100.0
SK 200	10	2	10	1	10	2	83.3	90.9
SK 220	14	0	13	0	**15**	0	100.0	57.7
SK 230	2	0	2	0	2	0	100.0	40.0
SK 240	3	0	2	0	3	0	100.0	75.0
SK 260	0	0	0	0	0	0	0.0	0.0
SK 280	2	2	2	2	2	2	50.0	100.0
SK 340	0	0	0	0	0	0	0.0	0.0
SK 370	2	0	2	0	2	0	100.0	66.7
SK 380	5	1	5	1	5	1	83.3	50.0
SK 399	0	0	0	0	0	0	0.0	0.0
SK 400	32	0	29	0	32	0	100.0	57.1
SK 420	0	0	0	0	0	0	0.0	0.0
SK 430	0	0	0	0	0	0	0.0	0.0
SK 450	0	0	0	0	0	0	0.0	0.0
SK 470	0	0	0	0	0	0	0.0	0.0
SK 500	0	0	0	0	0	0	0.0	0.0
SK 520	2	0	2	0	2	0	100.0	40.0
SK 540	2	0	2	0	2	0	100.0	100.0
SK 600	5	0	5	0	5	0	100.0	100.0
SK 620	0	0	0	0	0	0	0.0	0.0
SK 700	2	0	2	0	2	0	100.0	66.7
SK 800	6	1	6	1	6	1	85.7	50.0
SK 820	2	0	2	0	2	0	100.0	100.0
SK 840	8	17	8	0	8	17	32.0	61.5
SK 860	0	0	0	0	0	0	0.0	0.0
SK 870	1	0	1	0	1	0	100.0	50.0
SK 880	2	0	2	0	2	0	100.0	100.0
SK 890	2	0	0	0	2	0	100.0	50.0
SK 900	8	2	7	1	8	2	80.0	100.0
SK 910	0	0	0	0	0	0	0.0	0.0
SK 920	1	0	0	0	1	0	100.0	50.0
SK 950	29	2	27	2	**31**	2	93.9	75.6
SK 970	0	0	0	0	0	0	0.0	0.0
SK 980	0	0	0	0	0	0	0.0	0.0
SM 700	2	0	0	0	2	0	100.0	33.3
SM 730	0	0	0	0	0	0	0.0	0.0
SH 500	0	0	0	0	0	0	0.0	0.0
ST 601	1	0	2	0	2	0	100.0	40.0
QH 440	5	1	4	1	5	1	83.3	100.0
UK 1200	0	0	0	0	0	0	0.0	0.0
Summe:	167	41	149	18	171	42	80.3	60.4

Tabelle 7: Klassifikation: Kombination der Verfahren

Schlagwort	virtuelle Klasse		binäre Entscheidung		Kombination			
	richtig	falsch	richtig	falsch	richtig	falsch	Precision	Recall
Algebra	10	1	10	1	10	1	90.9	90.9
Algebraische Geometrie	1	0	0	0	1	0	100.0	50.0
Analysis	32	0	33	1	33	1	97.1	60.0
Analytische Geometrie	3	0	3	2	3	2	60.0	37.5
Computeralgebra	1	0	1	0	1	0	100.0	50.0
Differentialgeometrie	2	0	2	0	2	0	100.0	100.0
Differentialgleichung	0	0	0	0	0	0	0.0	0.0
Diskrete Mathematik	0	0	0	0	0	0	0.0	0.0
Elliptische Kurve	1	0	1	0	1	0	100.0	100.0
Fehlerrechnung	0	0	0	0	0	0	0.0	0.0
Funktionalanalysis	5	0	5	0	5	0	100.0	100.0
Funktionentheorie	2	0	2	0	2	0	100.0	50.0
Geometrie	1	2	0	0	1	2	33.3	100.0
Gewöhnliche Differentialgleichung	2	0	2	0	2	0	100.0	40.0
Graphentheorie	0	0	0	0	0	0	0.0	0.0
Höhere Mathematik	27	18	23	4	29	18	61.7	90.6
Integrationstheorie	0	0	0	0	0	0	0.0	0.0
Kombinatorik	1	0	1	0	1	0	100.0	100.0
Kommutative Algebra	2	0	2	0	2	0	100.0	100.0
Kryptologie	2	0	2	0	2	0	100.0	100.0
Lineare Algebra	15	0	13	0	15	0	100.0	60.0
Maple	0	0	0	0	0	0	0.0	0.0
Mathematik	0	7	0	0	0	7	0.0	0.0
Mathematikunterricht	1	0	0	0	1	0	100.0	16.7
Mathematische Logik	0	0	0	0	0	0	0.0	0.0
Mathematische Methode	0	0	0	1	0	1	0.0	0.0
Maßtheorie	0	0	0	0	0	0	0.0	0.0
Matlab	0	0	1	0	1	0	100.0	50.0
Mengenlehre	2	0	2	0	2	0	100.0	66.7
Numerische Mathematik	8	1	8	0	8	1	88.9	100.0
Numerisches Verfahren	1	0	0	0	1	0	100.0	100.0
Optimierung	1	0	1	0	1	0	100.0	50.0
Partielle Differentialgleichung	2	0	2	0	2	0	100.0	100.0
Physik	0	0	0	1	0	1	0.0	0.0
Spieltheorie	0	0	0	0	0	0	0.0	0.0
Statistik	8	3	8	0	8	3	72.7	80.0
Stochastik	4	1	4	2	4	2	66.7	100.0
Stochastischer Prozess	1	0	1	0	1	0	100.0	100.0
Topologie	2	2	2	2	2	2	50.0	100.0
Wahrscheinlichkeitstheorie	2	0	1	0	2	0	100.0	18.2
Wissenschaftliches Rechnen	1	0	1	0	1	0	100.0	100.0
Zahlentheorie	1	0	1	0	1	0	100.0	100.0
Summe:	141	35	132	14	145	41	78.0	64.2

Tabelle 8: Beschlagwortungsergebnis: Kombination der Verfahren

stimmtes Teilwort ein gültiges und bekanntes Wort ist oder nicht. Aus diesem Grund wird zunächst eine Liste aller Wörter erstellt, mit der dann diese Überprüfung durchgeführt werden kann. Da eine lineare Suche über eine Liste aller Wörter zu langsam wäre, werden die Wörter in einer speziellen Baumdarstellung, nach Prinzip des sogenannten Fleigengewicht-Muster [Gam01], wie sie in Abbildung 15 darstellt ist, gespeichert. Dabei markiert ein Stern (*) ein gültiges Ende eines Wortes. Die Überprüfungen eines Teilwortes der Länge n benötigt so maximal $n + 1$ Zugriffe auf die Datenstruktur.

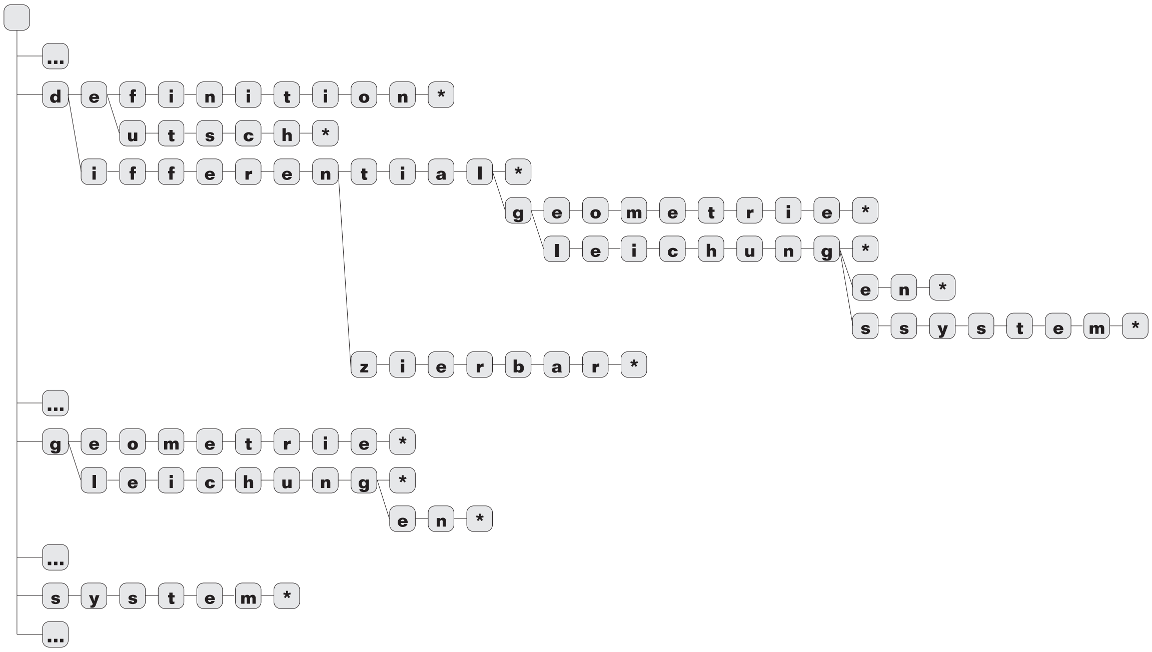

Abbildung 15: Baumstruktur zur Speicherung der Wörter

Einige zusammengesetze Wörter lassen sich nicht streng ohne Reste in die einzelnen Bestandteile zerlegen. Bei der Zerlegung müssen zwei Besonderheiten berücksichtig werden:

- Bei der Verbindung der Wörter *gleichung* und *system* zu dem Wort *gleichungssystem* wurde ein *s* als Verbindungsbuchstabe eingefügt. Bei der Zerlegung von Wörtern muss auch auf ein eventuelles Vorkommen von Verbindungsbuchstaben geachtet werden. In der deutschen Sprache gibt es die beiden Verbindungsbuchstaben *s* und *n*.

- Bei dem Versuch das Wort *differentialgleichungen* in die Bestandteile *differential* und *gleichung* zu zerlegen, fällt auf, dass *en* als Rest übrig bleibt. Es gibt bei der Zerlegung Reste, die mit recht hoher Wahrscheinlichkeit vermuten lassen, dass eine Zerlegung des Wortes in seine Bestandteile unter Vernachlässigung des Restes sinnvoll ist. Tabelle 9 enthält eine Liste mit gültigen Endungen und entsprechenden Beispielwörtern.

Tabelle 10 zeigt eine Reihe von zusammengesetzen Wörtern, die mit Hilfe dieses Verfahrens in die einzelnen Bestandteile zerlegt werden. Die Zerlegung von Wörtern durch Verwerfen

Endung	Beispiel	Endung	Beispiel
ant	musikant	igkeit	arbeitslosigkeit
bar	dankbar	ik	automatik
e	falle	ion	diskretion
ei	betrügerei	isch	alphabetisch
en	absichten	ischer	medizinischer
end	zielend	lich	absichtlich
em	negativerem	mittel	schlafmittel
er	abendkleider	n	adressen
ern	altern	oren	prozessoren
es	automatisches	s	autos
ien	materialien	se	verhältnisse
iert	differenziert	ster	kleinster
ierung	aktivierung	stoff	giftstoff
ig	vorzeitig	ung	kreuzung

Tabelle 9: gültige Endungen mit Beispielwörtern

Origianlwort	**Bestandteile**
differentialgeometrie	differential geometrie
funktionentheorie	funktionen theorie
zahlentheorie	zahlen theorie
übungen	übung
informatiker	informatik

Tabelle 10: Beispiele für die Wortzerlegung

bestimmter Reste hat ebenfalls zur Folge, dass einige Wörter auf ihre Grundform reduziert werden.

Die Ergebnisse des Verfahrens mit Stoppworteliminierung, Kompositazerlegung und Wortstammreduktion sind in den Tabellen 11 und 12 festgehalten. Dabei fällt auf, dass die Anzahl falsch vergebenen Klassifikationen und Schlagwörter abgenommen hat, die Anzahl der richtig vergebenen Klassifikationen gestiegen ist und die Anzahl der richtig vergebenen Schlagwörter konstant geblieben ist. Weitere Modifikationen des Verfahren zur Verbesserung der Ergebnisse wurden nicht unternommen.

RVK	ohne Kompositazerlegung				mit Kompositazerlegung			
	richtig	falsch	Precision	Recall	richtig	falsch	Precision	Recall
SK 110	8	9	47.1	44.4	8	8	50.0	44.4
SK 130	0	0	0.0	0.0	0	0	0.0	0.0
SK 150	2	0	100.0	66.7	2	0	100.0	66.7
SK 170	3	0	100.0	75.0	3	0	100.0	75.0
SK 180	1	0	100.0	100.0	1	0	100.0	100.0
SK 200	10	1	90.9	90.9	10	1	90.9	90.9
SK 220	13	0	100.0	50.0	14	0	100.0	53.8
SK 230	2	0	100.0	40.0	2	0	100.0	40.0
SK 240	2	0	100.0	50.0	2	0	100.0	50.0
SK 260	0	0	0.0	0.0	0	0	0.0	0.0
SK 280	2	2	50.0	100.0	2	2	50.0	100.0
SK 340	0	0	0.0	0.0	0	0	0.0	0.0
SK 370	2	0	100.0	66.7	2	0	100.0	66.7
SK 380	5	1	83.3	50.0	5	1	83.3	50.0
SK 399	0	0	0.0	0.0	0	0	0.0	0.0
SK 400	29	0	100.0	51.8	32	0	100.0	57.1
SK 420	0	0	0.0	0.0	0	0	0.0	0.0
SK 430	0	0	0.0	0.0	0	0	0.0	0.0
SK 450	0	0	0.0	0.0	0	0	0.0	0.0
SK 470	0	0	0.0	0.0	0	0	0.0	0.0
SK 500	0	0	0.0	0.0	0	0	0.0	0.0
SK 520	2	0	100.0	40.0	2	0	100.0	40.0
SK 540	2	0	100.0	100.0	2	0	100.0	100.0
SK 600	5	0	100.0	100.0	5	0	100.0	100.0
SK 620	0	0	0.0	0.0	0	0	0.0	0.0
SK 700	2	0	100.0	66.7	2	0	100.0	66.7
SK 800	6	1	85.7	50.0	6	1	85.7	50.0
SK 820	2	0	100.0	100.0	2	0	100.0	100.0
SK 840	8	0	100.0	61.5	8	0	100.0	61.5
SK 860	0	0	0.0	0.0	0	0	0.0	0.0
SK 870	1	0	100.0	50.0	1	0	100.0	50.0
SK 880	2	0	100.0	100.0	2	0	100.0	100.0
SK 890	0	0	0.0	0.0	0	0	0.0	0.0
SK 900	7	1	87.5	87.5	7	1	87.5	87.5
SK 910	0	0	0.0	0.0	0	0	0.0	0.0
SK 920	0	0	0.0	0.0	0	0	0.0	0.0
SK 950	27	2	93.1	65.9	27	2	93.1	65.9
SK 970	0	0	0.0	0.0	0	0	0.0	0.0
SK 980	0	0	0.0	0.0	0	0	0.0	0.0
SM 700	0	0	0.0	0.0	0	0	0.0	0.0
SM 730	0	0	0.0	0.0	0	0	0.0	0.0
SH 500	0	0	0.0	0.0	0	0	0.0	0.0
ST 601	2	0	100.0	40.0	2	0	100.0	40.0
QH 440	4	1	89.0	80.0	4	1	89.0	89.0
UK 1200	0	0	0.0	0.0	0	0	0.0	0.0
Summe:	149	18	89.2	52.7	153	17	90.0	54.1

Tabelle 11: Klassifikationsergebnis: ”binäre Entscheidung” ohne und mit Kompositazerlegung und Stoppworteliminierung

Schlagwort	ohne Kompositazerlegung				mit Kompositazerlegung			
	richtig	falsch	Precision	Recall	richtig	falsch	Precision	Recall
Algebra	10	1	90.9	90.9	10	1	90.9	90.9
Algebraische Geometrie	0	0	0.0	0.0	0	0	0.0	0.0
Analysis	33	1	97.1	60.0	33	1	97.1	60.0
Analytische Geometrie	3	2	60.0	37.5	3	2	60.0	37.5
Computeralgebra	1	0	100.0	50.0	1	0	100.0	50.0
Differentialgeometrie	2	0	100.0	100.0	2	0	100.0	100.0
Differentialgleichung	0	0	0.0	0.0	0	0	0.0	0.0
Diskrete Mathematik	0	0	0.0	0.0	0	0	0.0	0.0
Elliptische Kurve	1	0	100.0	100.0	1	0	100.0	100.0
Fehlerrechnung	0	0	0.0	0.0	0	0	0.0	0.0
Funktionalanalysis	5	0	100.0	100.0	5	0	100.0	100.0
Funktionentheorie	2	0	100.0	50.0	2	0	100.0	50.0
Geometrie	0	0	0.0	0.0	0	0	0.0	0.0
Gewöhnliche Differentialgleichung	2	0	100.0	40.0	2	0	100.0	40.0
Graphentheorie	0	0	0.0	0.0	0	0	0.0	0.0
Höhere Mathematik	23	4	85.2	71.9	23	4	85.2	71.9
Integrationstheorie	0	0	0.0	0.0	0	0	0.0	0.0
Kombinatorik	1	0	100.0	100.0	1	0	100.0	100.0
Kommutative Algebra	2	0	100.0	100.0	2	0	100.0	100.0
Kryptologie	2	0	100.0	100.0	2	0	100.0	100.0
Lineare Algebra	13	0	100.0	52.0	13	0	100.0	52.0
Maple	0	0	0.0	0.0	0	0	0.0	0.0
Mathematik	0	0	0.0	0.0	0	0	0.0	0.0
Mathematikunterricht	0	0	0.0	0.0	0	0	0.0	0.0
Mathematische Logik	0	0	0.0	0.0	0	0	0.0	0.0
Mathematische Methode	0	1	0.0	0.0	0	0	0.0	0.0
Maßtheorie	0	0	0.0	0.0	0	0	0.0	0.0
Matlab	1	0	100.0	50.0	1	0	100.0	50.0
Mengenlehre	2	0	100.0	66.7	2	0	100.0	66.7
Numerische Mathematik	8	0	100.0	100.0	8	0	100.0	100.0
Numerisches Verfahren	0	0	0.0	0.0	0	0	0.0	0.0
Optimierung	1	0	100.0	50.0	1	0	100.0	50.0
Partielle Differentialgleichung	2	0	100.0	100.0	2	0	100.0	100.0
Physik	0	1	0.0	0.0	0	1	0.0	0.0
Spieltheorie	0	0	0.0	0.0	0	0	0.0	0.0
Statistik	8	0	100.0	80.0	8	0	100.0	80.0
Stochastik	4	2	66.7	100.0	4	2	66.7	100.0
Stochastischer Prozess	1	0	100.0	100.0	1	0	100.0	100.0
Topologie	2	2	50.0	100.0	2	2	50.0	100.0
Wahrscheinlichkeitstheorie	1	0	100.0	9.1	1	0	100.0	9.1
Wissenschaftliches Rechnen	1	0	100.0	100.0	1	0	100.0	100.0
Zahlentheorie	1	0	100.0	100.0	1	0	100.0	100.0
Summe:	132	14	132	14	132	13	91.0	58.4

Tabelle 12: Beschlagwortungsergebnis: ”binäre Entscheidung” ohne und mit Kompositazerlegung und Stoppworteliminierung

4.2.3 Datenexport

Nachdem die Datensätze klassifiziert und beschlagwortet worden sind, müssen die neuen Metadaten über die Datenverwaltungskomponente in die Datenbank eingetragen werden. Durch die dokumentenorientierte Verarbeitungsweise ist es für jeden Datensatz notwendig, dass das ganze XML-Dokument beim XML-Prozessor angefragt wird. Die neuen Metadaten müssen dann in das XML-Dokument eingetragen werden und das XML-Dokument muss dann wieder an den XML-Prozessor übergeben werden. Dabei müssen die vorgesehenen Sperrmechanismen benutzt werden, um Änderungsanomalien zu vermeiden.

Genauso wie beim Datenimport wird beim Datenexport das Data-Bindung-Produkt Castor verwendet. Dabei werden die Klassifikations- und Schlagworteinträge eines jeden Datensatzes mit dem in Listing 4 (Seite 57) definierten Mapping in eine Datei geschrieben. Dann wird beim XML-Prozessor das XML-Dokument des entsprechenden Datensatzes angefordert und für den Zugriff durch andere gesperrt. Die beiden XML-Dokumente werden dann mit Hilfe der Extensible Stylesheet Language Transformation (XSLT [17]) zusammengeführt. Dabei werden, wie in Abbildung 16 dargestellt, die beiden XML-Dokumente und ein Stylesheet durch den XSLT-Prozessor gelesen. Der XSLT-Prozessor verarbeitet die beiden Dokumente nach den im Stylesheet angegeben Regeln und gibt ein Ergebnisdokument aus. Das Stylesheet zum Mischen der beiden XML-Dokumente ist im Listing 2 (Seite 55) dargestellt. Dieses Ergebnisdokument wird dann an den XML-Prozessor geschickt und die Zugriffssperre aufgehoben.

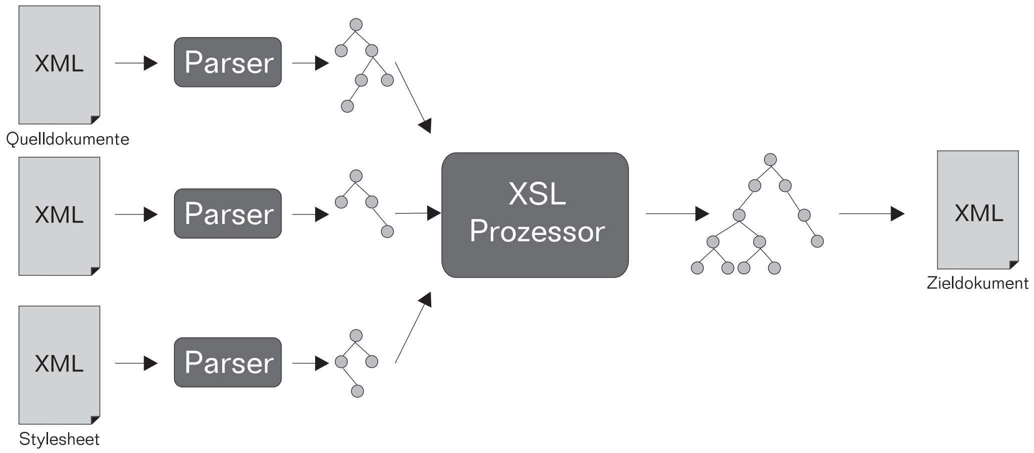

Abbildung 16: Ablauf der eXtensible Style Language Transformation

Alternativ könnte man beim Datenexport auf XSLT verzichten und die Modifikation der XML-Dokumente in Java realisieren. Dazu könnte man mit Hilfe des Klassengenerators von Castor Javaklassen generieren. Nachdem das XML-Dokument mit Castor in eine Struktur

[17] `http://www.w3.org/Style/XSL/`

von Java-Objekten überführt wurde, kann diese Objektstruktur durch entsprechende Methodenaufrufe manipuliert werden. Nach der Bearbeitung kann die Objektstruktur wieder in ein XML-Dokument überführt werden.

5 Bewertung

5.1 Grenzen automatischer Indexierung

Leider findet man nicht viele Aussagen über die Grenzen automatischer Indexierungsverfahren. Die Bewertung von Indexierungsergebnissen wird im Allgemeinen durch die Durchführung eines Retrievaltests vorgenommen. Die Ergebnisse der Retrievaltests, die in den einzelnen Projekten durchgeführt werden, sind jedoch nicht immer miteinander vergleichbar, da sie auf unterschiedlichen Datenbeständen und mit unterschiedlichen Anfragen an das Retrievalsystem durchgeführt werden.

Dennoch lassen sich einige grundlegende Bedingungen für eine erfolgreiche Indexierung festhalten. Um eine automatische Indexierung vornehmen zu können, müssen Teile des zu indexierenden Dokumentes in Textform vorliegen. Dabei kann es sich um den Volltext, ein Inhaltsverzeichnis, eine Zusammenfassung oder auch nur den Titel handeln. Das zur Verfügung stehende Textmaterial muss jedoch eine ausreichende Aussagekraft über den Inhalt des Dokumentes haben. Bei der Verwendung von Verfahren, die auf statistischen Analysen bereits indexierter Dokumente beruhen, müssen die Daten hinreichend homogen sein.

„Allgemein sind die Grenzen automatischer Erschließungsverfahren dort erreicht, wo die Intelligenz beginnt“ (Zitat Klaus Lepsky [Lep02]).

5.2 Abschließende Bewertung des implementierten Verfahrens

Im Rahmen der Implementierung wurden die einzelnen Verfahren nur nach der Anzahl der richtig oder falsch vergebenen Klassifikationen oder Schlagwörter betrachtet. Dabei konnten, wie bereits in den Tabellen 11 und 12 angegeben, gute Ergebnisse erzielt werden. Tabelle 13 beinhaltet nochmal eine Zusammenfassung der wichtigsten Ergebnisse. Die ersten Ergebnisse lieferte das Verfahren „virtuellen Klasse„. Im nächsten Schritt wurde überprüft, welches Ergebnis das Verfahren „binären Entscheidung“ liefert. Dabei fiel auf, dass beim Verfahren „binären Entscheidung“ die Precision um fast 10% höher war als beim Verfahren „virtuellen Klasse„. Im anschließenden Versuch, bei dem die beiden Verfahren kombiniert wurden, wurden die Ergebnisse wieder schlechter. Daher wurde nur noch das Verfahren „binären Entscheidung“ weiterentwickelt und eine Stoppworteliminierung und Kompositazerlegung vorgenommen. Diese Modifikation lieferte die besten Ergenisse mit einer Precision von über 90% und einem Recall zwischen 54% und 59%.

Aus Sicht der Bibliothek sind auch andere Zahlen zur Bewertung recht interessant. Da es sich um Dokumente handelt, die klassifiziert und beschlagwortet werden, sind Fragestellungen nach der Anzahl der Dokumente, für die keine Vorschläge generiert werden konnten, oder nach der Anzahl der Dokumente, die falsche Vorschläge enthalten, durchaus sinnvoll.

Die Datenmenge, mit der das Verfahren getestet wurde besteht leider nur aus 163 Doku-

Verfahren	Klassifikation Precision	Klassifikation Recall	Beschlagwortung Precision	Beschlagwortung Recall
Virtuelle Klasse	80.3%	59.0%	80.1%	62.4%
Binäre Entscheidung	89.2%	52.7%	90.4%	58.4%
Kombination der Verfahren	80.3%	60.4%	78.0%	64.2%
Binäre Entscheidung mit Stoppworteliminierung und Kompositazerlegung	90.0%	54.1%	91.0%	58.4%

Tabelle 13: Zusammenfassung der Ergebnisse

menten. Die Lernmenge, mit das Verfahren trainiert wurde enthält 693 Dokumente. Größere Datenmengen zur Evaluierung der Verfahren waren leider nicht verfügbar. Nach der Verarbeitung der Lernmenge bestand das bekannte Vokabular aus 486 verschiedenen Wörtern. Bei der Analyse der Testdokumente mussten 62 verschiedene Wörter aus den Titeln dieser Dokumente ignoriert werden, da diese Wörter im System nicht enthalten waren. Dies hat zur Folge, dass es 7 Dokumente gibt, die keine Wörter enthalten, auf deren Grundlage man eine Klassifikation oder Beschlagwortung vornehmen könnte.

Um eine Bewertung auf Dokumentebene durchzuführen sind folgende Fragen interessant:

- Für wie viele Dokumente konnte weder eine Klassifikation noch ein Schlagwort generiert werden?
- Wie viele Dokumente enthalten falsche Klassifikationen oder Schlagworte?
- Bei wie vielen Dokumenten entspricht die Menge der Klassifikationen und Schlagworte genau der Menge die intellektuell vergeben wurde?
- Bei wie vielen Dokumenten ist die Menge der Klassifikationen und Schlagworte eine Teilmenge der intellektuell vergebenen Klassifikationen und Schlagworte?

Eine Analyse der Ergebnisse auf Dokumentebene hat ergeben, dass bei 20 Dokumenten keine Klassifikation und auch kein Schlagwort generiert werden konnte. Bei 7 von den 20 Dokumenten ist dies, wie bereits erwähnt, auf fehlendes Vokabular in der Lernmenge zurückzuführen. Bei 19 Dokumenten enthielt die Menge der vorgeschlagenen Klassifikationen und Schlagworte auch Einträge, die intellektuell nicht vergeben worden sind. Dabei kann jedoch nicht immer ausgeschlossen werden, dass diese Vorschläge auch intellektuell vertretbar wären. Genau identisch sind die beiden Mengen bei 77 Dokumenten. Bei den verbleibenden 48 Dokumenten ist die automatisch vergebene Menge von Klassifikationen und Schlagworten eine Teilmenge der intellektuell vergebenen. Zusammenfassend läßt sich festhalten, dass das Verfahren in der Lage ist, mit einer Precision von 88% Dokumente zu klassifizieren und beschlagworten. Dabei konnten 78% aller Dokumente mit korrekten Metadaten versehen werden.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Im Rahmen dieser Arbeit, die sich in das Projekt META-AKAD eingliedert, wurden zunächst einige Verfahren, die sich mit automatischer Indexierung beschäftigen, vorgestellt. Dabei handelt es sich um Verfahren, die sowohl zum Vervollständigen von Metadaten als auch zur Generierung von Indexierungsinformation, die das Retrieval unterstützen sollen, genutzt werden. Danach wurde die Entwicklung des automatischen Indexierungsclients beschrieben und eine abschließende Bewertung durchgeführt. Dabei ist die Bewertung des Indexierungsclients zu einem positivem Ergebnis gekommen.

Diese Arbeit ist ein Bestandteil des Arbeitspakets 5 (Verbesserung der Qualität der Erschließung bei schlecht erschlossenem Material) im Projekt META-AKAD. Ziel dieses Arbeitspaketes ist, die Erschließung schlecht erschlossenen Materials maschinell zu unterstüzen, um diese Dokumente bei einer Suche finden zu können. Bereits zum Zeitpunkt des Projektantrags war allen Beteiligten klar, dass bei der automatischen Vergabe von Metadaten Fehler nicht zu vermeiden sind. Daher wurde eine intellektuelle Nachbearbeitung der automatisch produzierten Metadaten a priori vorgesehen, um die gewünschte Qualität der Metadaten sicher zu stellen [Gei00].

Das in dieser Arbeit entstandene Programm ist sicherlich in der Lage, die Ansprüche, die an eine automatische Klassifizierungs- und Beschlagwortungskomponente im Rahmen von Arbeitspaket 5 gestellt werden, zu erfüllen.

Die entwickelte Indexierungskomponente ist nicht nur in der Lage, Klassifikationen nach RVK und Schlagwörter nach SWD zu vergeben, sondern es werden beliebige Klassifikationen und beliebiges Vokabular unterstützt. Während der Entwicklung wurden die internen Datenstrukturen so geändert, dass unterschiedliche Schemata verarbeitet werden können, ohne dass Modifikationen am Code notwendig sind.

6.2 Ausblick

Ob der hier vorgestellte Ansatz ähnlich gute Ergebnisse auch in den anderen im Projekt bearbeiteten Fachgebieten erzielt, muss noch untersucht werden. Diese Untersuchung konnte im Rahmen dieser Arbeit nicht durchgeführt werden, da für eine entsprechende Evaluierung eine entsprechend große Dokumentmenge, die intellektuell klassifiziert und beschlagwortet wurde, notwendig ist. In dieser noch durchzuführenden Evaluierung muss ebenfalls überprüft werden, ob die verwendeten Algorithmen zur Konvertierung der Titel in entsprechende Vektoren auch für Dokumente anderer Sprachen die gewünschten Ergebnisse erzielen, oder ob sprachspezifische Anpassungen erforderlich sind.

Eine mögliche Erweiterung des Verfahrens ist die Integration eines Thesaurus, um ähnliche

Titel weiter angleichen zu können und somit die Streuung der Klassen zu verkleinern. Als Thesaurus für die Verarbeitung deutschsprachiger Dokumente könnte der in der Schlagwortnormdatei integrierte Thesaurus verwendet werden. Eine weitere Möglichkeit, deren Auswirkungen auf das Verfahren untersucht werden könnten, ist das automatische Hinzufügen des entsprechenden Schlagworts zu einem Dokument, falls bei der Titelanalyse ein Synonym durch das entsprechende Schlagwort ersetzt wird.

Außerdem wird es interessant sein, zu beobachten, inwieweit eine Vergrößerung der Lernmenge zur Steigerung des Recall beitragen kann. Aus diesem Grund wurde im Indexierungsclient die Möglichkeit geschaffen, eine Evaluierung der Lernmenge auf Grundlage der Dokumente in der Datenbank durchzuführen.

7 Anhang

7.1 Listings

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
   <xsl:output method="xml"/>

   <!-- Datei laden die gemischt werden soll -->
   <xsl:variable name="doc" select="document('export.xml')"/>

   <xsl:template match="/">
      <metabase>
         <xsl:for-each select="metabase/learningresource">

            <xsl:variable name="guID">
               <xsl:value-of select="@guID"/>
            </xsl:variable>

            <!-- Kinderknoten kopieren -->
            <learningresource>
               <xsl:copy-of select="child::*"/>

               <!-- Eintraege aus der Datei kopieren die zur    -->
               <!-- Learningresource mit aktuellen \$guID passen -->
               <xsl:copy-of select="\$doc/metabase/learningresource
                                     [@guID=\$guID]/child::*" />
            </learningresource>
         </xsl:for-each>
      </metabase>
   </xsl:template>
</xsl:stylesheet>
```

Listing 2: XSLT Stylesheet

```
<?xml version="1.0"?>
<!DOCTYPE mapping PUBLIC
 "-//EXOLAB/Castor Object Mapping DTD Version 1.0//EN"
 "http://www.chrweber.de/mapping.dtd">
<mapping>
   <description>
      Mapping des XML-Files des ResultSetProcessor
   </description>
   <class name="indexclient.data.MetaAkadVectorContainer">
     <map-to xml="metabase"/>
     <field name="metaAkadVector"
            type="indexclient.data.MetaAkadDocument"
            collection="vector">
       <bind-xml name="learningresource" node="element"/>
     </field>
   </class>
   <class name="indexclient.data.MetaAkadDocument">
      <map-to xml="learningresource"/>
      <field name="id" type="java.lang.String">
         <bind-xml name="guID" node="attribute"/>
      </field>
      <field name="title" type="java.lang.String">
         <bind-xml name="title" node="element"/>
      </field>
      <field name="subjectVector"
             type="indexclient.data.MetaAkadSubject"
             collection="vector">
         <bind-xml name="subject" node="element"/>
      </field>
   </class>
   <class name ="indexclient.data.MetaAkadSubject">
      <map-to xml="subject"/>
      <field name="scheme" type="java.lang.String">
         <bind-xml name="scheme" node="attribute"/>
      </field>
      <field name="notation" type="java.lang.String">
         <bind-xml name="notation" node="attribute"/>
      </field>
      <field name="origin" type="java.lang.String">
         <bind-xml name="origin" node="attribute"/>
      </field>
      <field name="content" type="java.lang.String">
         <bind-xml name="subject" node="text"/>
      </field>
   </class>
</mapping>
```

Listing 3: Castor Import-Mapping

```
<?xml version="1.0"?>
<!DOCTYPE mapping PUBLIC "-//EXOLAB/Castor Object Mapping DTD Version 1.0//EN"
                         "http://www.chrweber.de/mapping.dtd">

<mapping>
  <description>Mapping des XML-Files des ResultSetProcessor</description>

  <class name="indexclient.data.MetaAkadVectorContainer">
    <map-to xml="metabase"/>

    <field name="metaAkadVector"
           type="indexclient.data.MetaAkadDocument" collection="vector">
      <bind-xml name="learningresource" node="element"/>
    </field>
  </class>

  <class name="indexclient.data.MetaAkadDocument">
    <map-to xml="learningresource"/>

    <field name="id" type="java.lang.String">
      <bind-xml name="guID" node="attribute"/>
    </field>
    <field name="subjectVector"
           type="indexclient.data.MetaAkadSubject" collection="vector">
      <bind-xml name="subject" node="element"/>
    </field>
  </class>

  <class name ="indexclient.data.MetaAkadSubject">
    <map-to xml="subject"/>

    <field name="scheme" type="java.lang.String">
      <bind-xml name="scheme" node="attribute"/>
    </field>
    <field name="notation" type="java.lang.String">
      <bind-xml name="notation" node="attribute"/>
    </field>
    <field name="origin" type="java.lang.String">
      <bind-xml name="origin" node="attribute"/>
    </field>
    <field name="content" type="java.lang.String">
      <bind-xml name="subject" node="text"/>
    </field>
  </class>

</mapping>
```

Listing 4: Exportmapping für Castor

7.2 Package indexclient.data

7.2.1 CLASS DataEvaluation

DECLARATION

public class DataEvaluation
extends java.lang.Object

CONSTRUCTORS

- *DataEvaluation*
  `public` **DataEvaluation**`( )`

METHODS

- *calculatePreRec*
  `public void` **calculatePreRec**`( java.util.Vector` **learnDocs**`, java.util.Vector` **testDocs**`, java.util.Vector` **subjectVector**`, java.lang.String` **scheme** `)`

7.2.2 CLASS DataExport

DECLARATION

public class DataExport
extends java.lang.Object

FIELDS

- private Learningresource learningresource
- private LockCtrl lockCtrl

CONSTRUCTORS

- *DataExport*
  `public` **DataExport**`( )`

METHODS

- *exportDocuments*
  `public void` **exportDocuments**`( java.util.Vector` **documents** `)`
- *exportDocumentToDB*
  `private void` **exportDocumentToDB**`( java.lang.String` **guid** `)`
- *exportDocumentToFile*
  `private void` **exportDocumentToFile**`(`
  `indexclient.data.MetaAkadDocument` **document,**
  `indexclient.data.Mapping` **mapping** `)`
- *learningResourceJNDILookup*
  `private static`
  `metabase.xmlprocessor.ejb.learningresource.Learningresource`
  learningResourceJNDILookup`( )`
- *lockCtrlJNDILookup*
  `private static`
  `metabase.xmlprocessor.ejb.learningresource.LockCtrl`
  lockCtrlJNDILookup`( )`

7.2.3 CLASS DataImport

DECLARATION

public class DataImport
extends java.lang.Object

FIELDS

- public static final int LEARN
- public static final int CLASSIFY

- public static final int EVALUATE
- private QECtrl qeCtrl
- private RSPCtrl rspCtrl

CONSTRUCTORS

- *DataImport*
  `public` **DataImport** `( )`

METHODS

- *getQueryResult*
  `private metabase.queryengine.beans.queryresult.RSHandle` **getQueryResult** `( java.lang.String` **lang**`, java.lang.String` **course**`, int` **type** `)`
- *getXMLfromRSProcessor*
  `private java.io.Reader` **getXMLfromRSProcessor** `( metabase.queryengine.beans.queryresult.RSHandle` **rsHandle** `)`
- *importDocuments*
  `public java.util.Vector` **importDocuments** `( java.lang.String` **lang**`, java.lang.String` **course**`, int` **type** `)`
- *qeCtrlJNDILookup*
  `private static metabase.queryengine.ejb.qectrl.QECtrl` **qeCtrlJNDILookup** `( )`
- *rspCtrlJNDILookup*
  `private static metabase.resultprocessor.ejb.rspctrl.RSPCtrl` **rspCtrlJNDILookup** `( )`

7.2.4 CLASS **DataUnzip**

Klasse zum Entpacken von Zip-Archiven

DECLARATION

public class DataUnzip
extends java.lang.Object

CONSTRUCTORS

- *DataUnzip*
  `public` **DataUnzip**`( )`

METHODS

- *saveEntry*
  `private static void` **saveEntry**`( java.util.zip.ZipFile` **zf**, `java.util.zip.ZipEntry` **target** `)`
  - **Usage**
    * Speichert einen Eintrag aus einem ZIP-Archiv
  - **Parameters**
    * `zf` - Zipfile
    * `target` - ZipEntry

- *unzip*
  `public static void` **unzip**`( java.lang.String` **filename** `)`
  - **Usage**
    * Entpackt das ZIP-Archiv
  - **Parameters**
    * `filename` - Pfad der Datei die entpackt werden soll

7.2.5 CLASS **MetaAkadDocument**

DECLARATION

public class MetaAkadDocument
extends java.lang.Object

FIELDS

- private String id
- private String title
- private HashMap wordMap
- private Vector subjectVector

CONSTRUCTORS

- *MetaAkadDocument*
  `public` **MetaAkadDocument** `( )`

METHODS

- *addSubject*
  `public void` **addSubject** `(`
  `indexclient.data.MetaAkadSubject` **subject** `)`
- *getId*
  `public java.lang.String` **getId** `( )`
  - **Usage**
    - Getter for property id.
  - **Returns** - Value of property id.
- *getSubjectVector*
  `public java.util.Vector` **getSubjectVector** `( )`
  - **Usage**
    - Getter for property subject.
  - **Returns** - Value of property subject.
- *getTitle*
  `public java.lang.String` **getTitle** `( )`
  - **Usage**
    - Getter for property title.
  - **Returns** - Value of property title.

- *getWordMap*
  `public java.util.HashMap` **getWordMap**`( )`
  - **Usage**
    * Getter for property wordMap.
  - **Returns** - Value of property wordMap.

- *setId*
  `public void` **setId**`( java.lang.String` **id** `)`
  - **Usage**
    * Setter for property id.
  - **Parameters**
    * `id` - New value of property id.

- *setSubject*
  `public void` **setSubject**`( java.util.Vector` **subjectVector** `)`
  - **Usage**
    * Setter for property subject.
  - **Parameters**
    * `subject` - New value of property subject.

- *setTitle*
  `public void` **setTitle**`( java.lang.String` **title** `)`
  - **Usage**
    * Setter for property title.
  - **Parameters**
    * `title` - New value of property title.

- *setWordMap*
  `public void` **setWordMap**`( java.util.HashMap` **wordMap** `)`
  - **Usage**
    * Setter for property wordMap.
  - **Parameters**
    * `wordMap` - New value of property wordMap.

7.2.6 CLASS **MetaAkadSubject**

Klasse zur Speicherung aller Informationen eines Subjects (Schema, Notation, Inhalt, Herkunft)

DECLARATION

public class MetaAkadSubject
extends java.lang.Object

FIELDS

- private String scheme
- private String notation
- private String content
- private String origin

CONSTRUCTORS

- *MetaAkadSubject*
  `public` **MetaAkadSubject** `( )`
- *MetaAkadSubject*
  `public` **MetaAkadSubject** `(`
  `indexclient.data.MetaAkadSubject` **subject** `)`
- *MetaAkadSubject*
  `public` **MetaAkadSubject** `( java.lang.String` **scheme**`,`
  `java.lang.String` **content**`, java.lang.String` **origin** `)`
- *MetaAkadSubject*
  `public` **MetaAkadSubject** `( java.lang.String` **scheme**`,`
  `java.lang.String` **notation**`, java.lang.String` **content**`,`
  `java.lang.String` **origin** `)`

METHODS

- *equals*
  `public boolean` **equals** `( java.lang.Object` **sub** `)`
- *getContent*
  `public java.lang.String` **getContent** `( )`
  - **Usage**

    * Getter for property content.
  - **Returns** - Value of property content.

- *getNotation*
  `public java.lang.String` **getNotation**`( )`
  - **Usage**
    * Getter for property notation.
  - **Returns** - Value of property notation.

- *getOrigin*
  `public java.lang.String` **getOrigin**`( )`
  - **Usage**
    * Getter for property origin.
  - **Returns** - Value of property origin.

- *getScheme*
  `public java.lang.String` **getScheme**`( )`
  - **Usage**
    * Getter for property scheme.
  - **Returns** - Value of property scheme.

- *setContent*
  `public void` **setContent**`( java.lang.String` **content** `)`
  - **Usage**
    * Setter for property content.
  - **Parameters**
    * `content` - New value of property content.

- *setNotation*
  `public void` **setNotation**`( java.lang.String` **notation** `)`
  - **Usage**
    * Setter for property notation.
  - **Parameters**
    * `notation` - New value of property notation.

- *setOrigin*
  `public void` **setOrigin**`( java.lang.String` **origin** `)`
  - **Usage**

    * Setter for property origin.
  - **Parameters**
    * `origin` - New value of property origin.

- *setScheme*
  `public void` **setScheme** `( java.lang.String` **scheme** `)`

  - **Usage**
    * Setter for property scheme.
  - **Parameters**
    * `scheme` - New value of property scheme.

7.2.7 CLASS **MetaAkadVectorContainer**

Containerklasse für eine Vector von MetaAkadDokuments

DECLARATION

public class MetaAkadVectorContainer
extends java.lang.Object

FIELDS

- private Vector metaAkadVector

CONSTRUCTORS

- *MetaAkadVectorContainer*
  `public` **MetaAkadVectorContainer** `( )`

METHODS

- *getMetaAkadVector*
  `public java.util.Vector` **getMetaAkadVector** `( )`

  - **Usage**
    * Get-Methode für metaAkadVector.

– **Returns** - Liefert metaAkadVector zurück.

- *setMetaAkadVector*
  `public void` **setMetaAkadVector**`(`
  `java.util.Vector` **metaAkadVector** `)`
  – **Usage**
    * Set-Methode für metaAkadVector.
  – **Parameters**
    * `metaAkadVector` - neuer Wert für metaAkadVector.

7.3 Package indexclient

7.3.1 CLASS Indexing

Startklasse für den META-AKAD-Indexclient
Hilfe: –help
Installation –install URL
Klassifikation –run FACH SPRACHE

DECLARATION

public class Indexing
extends java.lang.Object

CONSTRUCTORS

- *Indexing*
  `public` **Indexing**`( )`

METHODS

- *installConfig*
  `private static void` **installConfig**`( java.lang.String` **zipurl** `)`
  – **Usage**
    * Installationsmethode für die Konfigurationsdaten
  – **Parameters**
    * `zipurl` - URL der Konfigurations-ZIP-Datei

- *main*
  `public static void` **main** `( java.lang.String[]` **args** `)`
  - **Usage**
    * Startmethode
  - **Parameters**
    * `args` - Komandozeilenparameter

- *runIndexingProcess*
  `private static void` **runIndexingProcess** `(`
  `java.util.Properties` **properties** `)`

7.4 Package indexclient.indexing

7.4.1 CLASS SubjectAnalyser

Klasse zur Analyse und Speicherung der in MetaAkadDocuments vorkommenden Klassifikationen und Schemata

DECLARATION

public class SubjectAnalyser
extends java.lang.Object

FIELDS

- private Vector schemeVector
- private Vector subjectVector

CONSTRUCTORS

- *SubjectAnalyser*
  `public` **SubjectAnalyser** `( )`

METHODS

- *analyse*
  `public void` **analyse**`(` `java.util.Vector` **documents** `)`
  - **Usage**
    - Analysiert die übergebenen Dokumente nach Klassifikationen und deren Schemata. Diese werden dann im schemeVector und subjectVector gespeichert
  - **Parameters**
    - `documents` - Vektor mit den zu untersuchenden Dokumenten

- *getSchemeCount*
  `public int` **getSchemeCount**`( )`
  - **Usage**
    - Get-Methode für die Anzahl der verschieden Schemata.
  - **Returns** - Anzahl der verschieden Schemata

- *getSchemeVector*
  `public java.util.Vector` **getSchemeVector**`( )`
  - **Usage**
    - Get-Methode für den Vektor mit allen Schemata.
  - **Returns** - Vektor mit allen Schemata.

- *getSubjectCount*
  `public int` **getSubjectCount**`( )`
  - **Usage**
    - Get-Methode für die Anzahl der verschieden Klassifikationen.
  - **Returns** - Anzahl der verschieden Klassifikationen.

- *getSubjectCountByScheme*
  `public int` **getSubjectCountByScheme**`(` `java.lang.String` **scheme** `)`
  - **Usage**
    - Get-Methode für die Anzahl aller Klassifiaktionen eines bestimten Schemas.
  - **Parameters**
    - `scheme` - Schema dessen Anzahl der Klassifikationen zurückgeliefert werden soll
  - **Returns** - Anzahl der gesuchten Klassifikationen

- *getSubjectVector*
  `public java.util.Vector` **getSubjectVector**`( )`
  - **Usage**
    * Get-Methode für den Vektor mit allen Klassifikationen.
  - **Returns** - Vector mit allen Klassifikationen.

- *getSubjectVectorByScheme*
  `public java.util.Vector` **getSubjectVectorByScheme**`(`
  `java.lang.String` **scheme** `)`
  - **Usage**
    * Get-Methode für alle Klassifikationen eines bestimten Schemas.
  - **Parameters**
    * `scheme` - Schema, dessen Klassifikationen zurückgeliefert werden sollen
  - **Returns** - Vektor mit den gesuchten Klassifikationen

7.4.2 CLASS **Svm**

Diese Klasse ist für den Auruf der Methoden, die von LIBSVM zur Verfügung gestellt werden, zuständig.

Die Daten werden in den entsprechenden Datenstrukturen abgelegt, bzw. manipuliert. Die Ergebnisse werden wieder in die Dokumentdatenstruktur gespeichert

DECLARATION

public class Svm **extends** java.lang.Object

FIELDS

- private svm_parameter svm_param
- private svm_problem my_svm_problem
- private svm_model my_svm_model

CONSTRUCTORS

- *Svm*
  `public` **Svm** `( )`
  - **Usage**
    - Erstellt eine neue Instanz von Svm und setzt Parameter, die für von den SVM-Methoden benötigt werden

METHODS

- *addProposal*
  `public int` **addProposal** `( java.util.Vector` **test_docs**`, indexclient.data.MetaAkadSubject` **subject** `)`
  - **Usage**
    - Methode zum Klassifizieren der Testdokumente. Die vorgeschlagenen Klassifikationen werden im Dokumentenvektor gespeichert
  - **Parameters**
    - `test_docs` - Vektor mit Testdokumenten
    - `category` - zu untersuchende Klassifikation

- *buildNewProblem*
  `public void` **buildNewProblem** `( java.util.Vector` **documents** `)`
  - **Usage**
    - Methode zum Setzen der Vektoren aus der Lernmenge, um daraus ein Modell berechnen zu lassen
  - **Parameters**
    - `documents` - Vektor mit MetaAkadDokumenten

- *getSvmNode*
  `public indexclient.libsvm.svm_node[]` **getSvmNode** `( indexclient.data.MetaAkadDocument` **doc** `)`
  - **Usage**
    - Methode zur Konvertierung des im MetaAkadDocument als HashMap gespeicherten Vektors in das von LIBSVM gebnötigte Format.
  - **Parameters**
    - `doc` - Das zu konvertierende MetaAkadDocument
  - **Returns** - das von LIBSVM benötigte Vektorformat

- *setParamGamma*
  `public void` **setParamGamma** `( double` **value** `)`
  - **Usage**
    - Set-Methode für den SVM-gamma Wert (Empfehlung: 1/#Merkmale)
  - **Parameters**
    - `value` - Gammawert

- *setProblemClasses*
  `public void` **setProblemClasses** `( java.util.Vector` **documents**`, indexclient.data.MetaAkadSubject` **subject** `)`
  - **Usage**
    - Methode zum Setzen der Positiv- und Negativbeipiele für die entsprechende Klassifikation und anschließende Modellbildung
  - **Parameters**
    - `documents` - Vektor mit Lerndokumente
    - `subject` - zu untersuchende Klassifikation

7.4.3 CLASS **Vectorisation**

Klasse zur Vektorisierung von Texten

DECLARATION

public class Vectorisation
extends java.lang.Object

FIELDS

- public static final int LEARN
- public static final int CLASSIFY
- private HashMap idCache
- private HashMap countCache
- private HashMap globalCountCache
- private HashMap idfCache

- private Vector stopwordVector
- private int learnDocumentCount
- private int wordCount

CONSTRUCTORS

- *Vectorisation*
  `public` **Vectorisation** `( )`

METHODS

- *addCountToCache*
  `private void` **addCountToCache** `( java.util.HashMap` **map** `)`
  - **Usage**
    * Addiert die Häufigkeiten in der HashMap zum CountCache
  - **Parameters**
    * `map` - HashMap

- *analyseString*
  `private java.util.HashMap` **analyseString** `(`
  `java.lang.String` **string**`, int` **type**`,`
  `int` **minWordLength**`, boolean` **stopwords**`,`
  `indexclient.indexing.WordTree` **wordTree** `)`
  - **Usage**
    * Zerlegt den übergebenen String und speichert die Wörter in einer HashMap
  - **Parameters**
    * `string` - String, der zerlegt werden soll
    * `type` - Modus LEARN oder CLASSIFY
    * `minWordLength` - Mindestlänge eines Wortes, damit es verwendet wird
    * `stopwords` - true, wenn Stopwörter eleminiert werden sollen
    * `wordTree` - WordTree, in dem die Wörter gespeichert werden

- *analyseVector*
  `private java.util.HashMap` **analyseVector** `(`
  `java.util.Vector` **words** `)`
  - **Usage**
    * Konvertiert einen Vector von Wörtern in eine HashMap von Wörtern und deren Häufigkeit

- **Parameters**
  - `words` - Vektor mit Wörtern
- **Returns** - HashMap mit Wörtern und deren Häufigkeit

- *calculateIDF*
  `private void` **calculateIDF**`( )`
  - **Usage**
    - Berechnet die Inverse Document Frequency

- *decomposeWords*
  `private java.util.HashMap` **decomposeWords**`( indexclient.indexing.WordTree` **tree**`, java.util.HashMap` **map**`, int` **minDecompositionLength** `)`
  - **Usage**
    - Zerlegt die in der HashMap übergebenen Wörter in Wörter, die im Wordtree vorhanden sind und gibt eine HashMap mit geänderten Häufigkeiten zurück
  - **Parameters**
    - `tree` - WordTree, der die möglichen Teilwörter enthält
    - `map` - HashMap mit den zu zerlegenden Wörtern
    - `int` - minDecompositionLength Mindestlänge eines Wortes, damit ein Zerlegungsversuch unternommen wird
  - **Returns** - HashMap mit den zerlegeten und nicht zerlegbaren Wörtern und die entsprechenden Häufigkeiten

---

- *fetchId*
  `private java.lang.Integer` **fetchId**`( java.lang.String` **word** `)`
  - **Usage**
    - Liefert eine eindeutige Nummer für das übergebene Wort zurück
  - **Parameters**
    - `word` - Wort dessen ID gesucht wird
  - **Returns** - ID des übergebenen Wortes

- *generateWordMap*
  `public void` **generateWordMap**`( java.util.Vector` **documents**`, int` **type**`, indexclient.indexing.WordTree` **wordTree**`, java.util.Properties` **prop** `)`
  - **Usage**
    - Erstellt die von der SVM benötigen Vektoren und speichert diese in den Dokumenten ab

– **Parameters**
  ∗ `documents` - Vektor mit MetaAkadDocuments
  ∗ `type` - Modus LEARN oder CLASSIFY
  ∗ `wordTree` - WordTree in dem alle relevanten Wörter gespeichert werden
  ∗ `prop` - Properties wie minWordLength, minDecompositionLength, stopword, decomposition, ...

- *getLearnDocumentCount*
  `public int` **getLearnDocumentCount**`( )`
  – **Usage**
    ∗ Get-Methode für die Anzahl der Lerndokumente.
  – **Returns** - Anzahl der Lerndokumente.

- *getStopWords*
  `public java.util.Vector` **getStopWords**`(`
  `java.lang.String` **sourceFile** `)`
  – **Usage**
    ∗ Liest Stoppwörter aus der übergeben Datei ein und liefert sie als Vektor zurück
  – **Parameters**
    ∗ `sourceFile` - Pfad der Datei, die eingelesen werden soll
  – **Returns** - Vector mit Stoppwörtern

- *getWordCount*
  `public int` **getWordCount**`( )`
  – **Usage**
    ∗ Get-Methode für die Anzahl der verschiedenen Wörter.
  – **Returns** - Anzahl der verschiedenen Wörter.

- *getWordIds*
  `private java.util.HashMap` **getWordIds**`(`
  `java.util.HashMap` **oldMap** `)`
  – **Usage**
    ∗ Konvertiert eine HashMap mit Wörtern in eine HashMap mit ID’s
  – **Parameters**
    ∗ `oldMap` - HashMap mit Wörtern
  – **Returns** - HashMap mit ID’s

- *norm*
  `private java.util.HashMap` **norm**`( java.util.HashMap` **map** `)`

- **Usage**
  - Normiert die übergebene HashMap auf die Länge 1
- **Parameters**
  - `zu` - normierende HashMap
- **Returns** - Normierte HashMap

- *replaceChars*
  `private java.lang.String` **replaceChars**`(`
  `java.lang.String` **word** `)`
  - **Usage**
    - Ersetzt bestimmte Zeichen im übergebenen Wort
  - **Parameters**
    - `word` - Wort in dem Zeichen ersetzt werden sollen
  - **Returns** - Wort in dem die Zeichen ersetzt wurden

7.4.4 CLASS **WordTree**

Klasse zur Speicherung von großen Wortmengen in einer Baumstruktur. Bietet die Möglichkeit zur Zerlegeung eines Wortes in Wörter, die in der Baumstruktur vorhanden sind

DECLARATION

public class WordTree
extends java.lang.Object

FIELDS

- private static Vector suffixVector
- private static Vector bondingVector
- private Hashtable table

CONSTRUCTORS

- *WordTree*
  `public` **WordTree**`( )`

- *WordTree*
  `public` **WordTree**`( java.util.Properties` **prop** `)`
  - **Usage**
    - Erzeugt eine neue Instanz eines WordTree
  - **Parameters**
    - `prop` - Properties decomposition, suffix, bonding & language

METHODS

- *addWord*
  `public void` **addWord**`( java.lang.String` **word** `)`
  - **Usage**
    - Fügt ein neues Wort dem Baum hinzu
  - **Parameters**
    - `Wort` - das hinzugefügt werden soll

- *containsSubWord*
  `public boolean` **containsSubWord**`( java.lang.String` **subword** `)`
  - **Usage**
    - Testet, ob der Baum ein Teilwort enthält
  - **Parameters**
    - `subword` - zu suchendes Teilwort
  - **Returns** - true, wenn das Teilwort im Baum vorhanden ist, sonst false

- *containsWord*
  `public boolean` **containsWord**`( java.lang.String` **word** `)`
  - **Usage**
    - Testet ob der Baum ein Wort als gültiges Wort enthält
  - **Parameters**
    - `subword` - zu suchendes Teilwort
  - **Returns** - true, wenn das Wort im Baum vorhanden ist, sonst false

- *decompose*
  `public java.util.Vector` **decompose**`( java.lang.String` **word** `)`
  - **Usage**
    - Methode zum Zerlegen des übergebenen Wortes in Wörter, die im Baum vorhanden sind, und eventuelle gültige Endungen

- **Parameters**
  - `word` - Zu zerlegendes Wort
- **Returns** - Vektor mit den gefunden Teilwörtern, NULL falls das Wort nicht zerlegbar ist

---

- *decompose*
  `private java.util.Vector` **decompose**`( java.lang.String` **word**`, boolean` **start** `)`
  - **Usage**
    - interne Methode zum Zerlegen von Wörtern
  - **Parameters**
    - `word` - das zu zerlegende Wort
    - `start` - muss beim ersten Aufruf true sein

- *getSubTree*
  `private indexclient.indexing.WordTree` **getSubTree**`( java.lang.String` **word** `)`
  - **Usage**
    - Liefert den Teilbaum zurück, der unterhalb des übergebenen Wortes hängt
  - **Parameters**
    - `word` - Wort, dessen Unterbaum zurückgeliefert werden soll
  - **Returns** - WordTree, der unterhalb des übergebenen Wortes hängt

---

- *getWordEndings*
  `public java.util.Vector` **getWordEndings**`( java.lang.String` **word** `)`
  - **Usage**
    - Liefert alle gültigen Endungen des übergebenen Wortes zurück
  - **Parameters**
    - `word` - Wort dessen gültigen Endungen gesucht werden sollen
  - **Returns** - Vektor der alle gültigen Endungen enthält

- *getWords*
  `public java.util.Vector` **getWords**`( java.lang.String` **prefix** `)`
  - **Usage**
    - Liefert alle Wörter des Baumes zurück, die mit dem übergebenen Prefix anfangen
  - **Parameters**
    - `prefix` - mit dem alle Wörter beginnen sollen

- **Returns** - Vektor mit allen Wörtern, die mit dem prefix beginnen

---

- *getWordsAsVector*
  `public java.util.Vector` **getWordsAsVector**`(`
  `java.lang.String` **sourceFile** `)`
  - **Usage**
    * Liest die angegebene Datei und liefert die enthaltenen Wörter in einem Vektor zurück
  - **Parameters**
    * `sourceFile` - Dateiname
  - **Returns** - Vektor mit den in der Datei enthaltenen Wörtern

---

- *print*
  `public void` **print**`( )`
  - **Usage**
    * Methode zum Anzeigen der derzeitigen Baumstruktur

- *print*
  `private void` **print**`( java.lang.String` **prefix** `)`
  - **Usage**
    * interne Hilfsmethode zum Anzeigen der derzeitigen Baumstruktur
  - **Parameters**
    * `prefix` - bereits erkanntes Wort

7.5 Bedienung des Indexierungsclients

Um den Indexierungsclient benutzen zu können, muss auf dem entsprechenden Rechner das `applicationlauncher.jar` des Applicationservers Orion [18] vorhanden sein. Der Aufruf erfolgt mit

```
java -jar applicationlauncher.jar
ormi://host:port/applicationname/indexclient.jar
user passord [Optionen]
```

Bevor mit der Klassifizierung und Beschlagwortung begonnen werden kann, muss ein ZIP-Archiv mit Konfigurationsdaten heruntergeladen und entpackt werden. Dies ist mit der Option `--install URL` möglich. Als URL ist dabei die URL des ZIP-Archives anzugeben (Bsp. `http://metaakad.ub.uni-kl.de/indexclient.zip`. Die Konfigurationsdateien werden in das Verzeichnis `metaakad_Indexclient` entpackt. In diesem Konfigurationsverzeichnis befinden sich folgende Dateien:

[18] `http://www.orionserver.com/`

Parameter	Bedeutung
stopword	Mit diesem Parameter kann festgelegt werden, ob eine Stoppworteliminierung durchgeführt werden soll.
decomposition	Mit diesem Parameter kann festgelegt werden, ob eine Kompositazerlegung durchgeführt werden soll.
minDecompositionLength	Gibt die Mindestlänge eines Wortes an, um eine Kompositazerlegung zu versuchen.
minWordLength	Gibt die Mindestlänge eines Wortes an, damit es in die Vektorisierung einbezogen wird.
stopwordDE	Gibt den Namen der Datei an, in der die Stoppwörter für die Sprache `DE` stehen. Das Sprachkürzel kann auch gegen andere Sprachen (z.B. `EN`) ausgetauscht werden.
suffixDE	Gibt den Namen der Datei an, in der die gültigen Endungen für die Sprache `DE` zu finden sind. Das Sprachkürzel kann auch gegen andere Sprachen (z.B. `EN`) ausgetauscht werden.
bondingDE	Gibt den Namen der Datei an, in der die Verbindungsbuchstaben für die Sprache `DE` abgelegt sind. Das Sprachkürzel kann auch gegen andere Sprachen (z.B. `EN`) ausgetauscht werden.

Tabelle 14: Parameter in der Datei `index.conf`

- `index.conf`
  Die Datei `index.conf` ist die zentrale Konfigurationsdatei. In Listing 5 ist eine Beispieldatei dargestellt. Die Bedeutung der Parameter, die in der Konfigurationsdatei eingestellt werden können, ist in Tabelle 14 erklärt.

- `importmapping.xml`
  Mapping für den XML-Import mit dem Data-Binding-Produkt Castor.
  (siehe Listing 3)

- `exportmapping.xml`
  Mapping für den XML-Export mit dem Data-Binding-Produkt Castor.
  (siehe Listing 4)

- `merge.xsl`
  XSLT-Stylesheet zum Kombinieren von XML-Dokumenten. (siehe Listing 2)

- `bondings.DE`
  Datei mit deutschen Verbindungsbuchstaben

- `suffix.DE`
  Datei mit gültigen deutschen Endungen

- `stopwords.DE`
  Datei mit deutschen Stoppwörtern

```
stopword = true
decomposition = true
minDecompositionLength = 4
minWordLength = 4
stopwordDE = stopwords.DE
suffixDE= suffix.DE
bondingDE = bondings.DE
```

Listing 5: index.conf

Der Klassifizierungs- und Beschlagwortungsvorgang wird mit der Option `-- run FACH SPRACHE` (z.B. `--run Mathematik DE`) gestartet. Um die derzeitige Lernmenge zu evaluieren, muss das Programm mit der Option `-- evaluate FACH SPRACHE` aufgerufen werden.

Literatur

[Cha02] Amin Chatti. Einsatz von Text-Mining-Algorithmen bei der Realisierung eines semi-automatischen Erschließungswerkzeugs im Projekt Meta-Akad. Projektarbeit, FB Informatik, Universität Kaiserslautern, Dezember 2002.

[Deß02] Stefan Deßloch. Web Services Support in Middleware Platforms (J2EE). Vorlesungsskript, FB Informatik, Universität Kaiserslautern, 2002. `http://wwwdvs.informatik.uni-kl.de/courses/WFWS/WS2002/Vorlesungsunterl%agen/Kapitel.05.pdf`.

[DPHS98] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the Seventh International Conference on Information and Knowledge Management*, pages 148–155, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE. `http://research.microsoft.com/~sdumais/cikm98.pdf`.

[FFW02] Marcus Flehmig, Stephan Fudeus, and Christian Weber. Realisierung einer Webbasierten Suchschnittstelle für die Verwendung in Meta-Akad. Interner Praktikumsbericht, FB Informatik, Universität Kaiserslautern, August 2002.

[FH02] Marcus Flehmig and Theo Härder. DB-orientierte Aufbereitung und Suchunterstützung für Lehr-/Lerndokumente. Projektbericht, FB Informatik, Universität Kaiserslautern, Dezember 2002. `http://wwwdvs.informatik.uni-kl.de/pubs/papers/FH02.INNO.pdf`.

[FKL+02] Marcus Flehmig, Helge Knüttel, Barbara Leiwesmeyer, Norbert Ritter, Martin Schmettow, and Gisela Weber. Metadatenzugang für akademisches Lehr- und Lernmaterial. Forschungsbericht, Universität Regensburg, Universität Kaiserslautern, Oktober 2002.

[Gam01] Erich Gamma. *Entwurfsmuster : Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2001.

[Gei00] Friedrich Geißelmann. Metadatenzuganz für akademisches Lehr- und Lernmaterial. Projektantrag, Universität Regensburg, Universität Kaiserslautern, November 2000.

[Hem01] Bastian Hemminger. Kerninduzierte Merkmalsräume. Seminararbeit, Universität Karlsruhe, 2001. `ftp://i11ftp.ira.uka.de/pub/neuro/lauer/SVMSem01_bhe.ps.gz`.

[Hic98] Thomas B. Hickey. *CORC - Cooperative Online Resource Catalog*. Online Computer Library Center, Inc., 1998. `http://www.oclc.org/research/publications/arr/1998/hickey/corc.htm`.

[Hus01] Martin Husemann. Java 2, Enterprise Edition Einführung und Überblick. Seminararbeit, FB Informatik, Universität Kaiserslautern, 2001. `http://wwwdvs.informatik.uni-kl.de/courses/seminar/SS2001/ausarbeitung5%.pdf`.

[J2E02] Java 2 Enterprise Edition (J2EE): Ein Architekturüberblick. Seminararbeit, Universität Osnabrück, August 2002. `http://bow.oec.uni-osnabrueck.de/lehre/aktuelles_semester/ws02_03/haupt%studium_bow/seminare/download/orga_wi/j2ee_2.pdf`.

[Joa97] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 143–151, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US. `http://www-ai.cs.uni-dortmund.de/DOKUMENTE/joachims_97a.pdf`.

[Joa98] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE. `http://www.cs.cornell.edu/People/tj/publications/joachims_97b.pdf`.

[Lep02] Klaus Lepsky. Automatisches Indexieren. Vorlesungsskript, Fachhochschule Köln, 2002. `http://www.kl.forschung-am-bau.de/Informatives/WFI-Ilmenau.pdf`.

[Luc01] Heinz-Dirk Luckhardt. Automatische und intellektuelle Indexierung. Vorlesungsskript, Universität des Saarlandes, 2001. `http://www.phil.uni-sb.de/fr/infowiss/papers/iwscript/exkurs.ind.html`.

[Lun97] Lund University Libraries. *DESIRE - Development of a European Service for Information on Research and Education*, November 1997. `http://www.lub.lu.se/desire/desireIindex.html`.

[Sch01a] Martin Schmettow. *Rahmenkonzept für die Benutzerschnittstelle zum Sammeln und Erschließen*, August 2001.

[Sch01b] Andreas Schröder. Support-Vektor-Maschinen. Seminararbeit, Universität Karlsruhe, 2001. `ftp://i11ftp.ira.uka.de/pub/neuro/lauer/SVMSem01_as.ps.gz`.

[Sun03a] Sun Microsystems, Inc. *Designing Enterprise Applications with the J2EE Platform*, 2003. `http://java.sun.com/blueprints/guidelines/designing_enterprise_applicat%ions/`.

[Sun03b] Sun Microsystems, Inc. *Java(TM) 2 Platform, Enterprise Edition v 1.4 - Documentation*, 2003. http://java.sun.com/j2ee/1.4/docs/.

[ULB99] ULB Düsseldorf. *KASCADE - KAtalogerweiterung durch SCanning und Automatische DokumentErschließung*, Juni 1999. http://www.uni-duesseldorf.de/ulb/kas_home.htm.

[ULB01] ULB Düsseldorf. *MILOS - Maschinelle Indexierung zur erweiterten Literaturerschließung in Online-Systemen*, Januar 2001. http://www.uni-duesseldorf.de/ulb/mil_home.htm.

[Uni99] Universität Osnabrück. *OSIRIS - Osnabrück Intelligent Research Information System*, Juni 1999. http://www.ub.uni-osnabrueck.de/osiris/index.html.