Entwicklung eines dezentralen aktiven Dokumentenspeichersystems für digitale Bibliotheken

Diplomarbeit

eingereicht bei

Prof. Dr. O. Drobnik
Professur für Telematik und verteilte Systeme
Johann Wolfgang Goethe - Universität
Frankfurt am Main

von

Hans Matzen
Henry-Budge-Str. 58
60320 Frankfurt/Main

Hiermit versichere ich, Hans Matzen, daß ich die vorliegende Arbeit selbständig verfaßt, keine anderen, als die angegebenen Hilfsmittel verwendet und die Stellen, die anderen Werken dem Wortlaut nach entnommen sind, mit Quellenangaben kenntlich gemacht habe.

Frankfurt/Main, den 28. Januar 1998 ____________________

(Hans Matzen)

Vorwort

Diese Arbeit entstand im Rahmen des Projekts „Digitale Bibliothek Frankfurt" der Professur für Telematik und verteilte Systeme und des Hochschulrechenzentrums der Universität Frankfurt/Main. An dieser Stelle möchte ich mich bei den Menschen bedanken, die diese Arbeit ermöglicht haben. Vielen Dank an Herrn Professor O. Drobnik, ohne den es dieses Projekt nicht geben würde. Ich möchte auch Martin Hess, Andreas Heckwolf und Andreas Schessner für die Gespräche und die technische Unterstützung danken.

Mein ganz besonderer Dank geht an Christian Mönch, der sich immer Zeit für lange Diskussionen genommen hat und mir mit Rat und Tat zur Seite stand.

Frankfurt/Main, im Januar 1998

Hans Matzen

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

In den letzten Jahren haben digitale Bibliotheken zunehmend an Bedeutung gewonnen. Digitale Bibliotheken ermöglichen es große Mengen digitaler Dokumente der verschiedensten Medientypen (z.B. Texte, Videofilme, Musik, Bilder, usw.) zum Zugriff durch eine Vielzahl von Benutzern bereitzustellen. Weiterhin erlauben digitale Bibliotheken die Nutzung neuer digitaler Medien, sowie die Konservierung antiquarischer Medien durch Digitalisierung.

Für den Austausch, die Beschreibung und die Archivierung digitaler Dokumente hat sich die „Standard Generalized Markup Language“ (SGML) als geeignet erwiesen. Aus diesem Grund findet SGML in letzter Zeit vermehrte Aufmerksamkeit bei der Realisierung digitaler Bibliotheken.

Mit zunehmender Komplexität und Verfügbarkeit multimedialer Dokumente, stoßen auch leistungsfähige Dokumentenspeichersysteme zum Einsatz in digitalen Bibliotheken auf verstärktes Interesse.

Dokumentenspeichersysteme dienen der Aufbewahrung und dem Zugriff auf die in der Bibliothek verwalteten multimedialen Dokumente. Dabei kommt den Speichersystemen die wichtige Aufgabe der typgerechten Bereitstellung der Dokumente zu. Zusätzlich müssen Speichersysteme Eigenschaften aufweisen, die es erlauben eine große Anzahl von Zugriffen auf umfangreichen Datenbeständen auszuführen.

Ziel dieser Arbeit ist es, ein Dokumentenspeichersystem für den Einsatz in verteilten digitalen Bibliotheken zu entwickeln. Das Speichersystem soll multimediale Dokumente aufnehmen und den effektiven Zugriff darauf gewährleisten. Besondere Aufmerksamkeit wird dabei der Verwaltung von SGML-Dokumenten gewidmet.

Das Speichersystem soll an Anzahl und Umfang der zu speichernden Dokumente angepaßt werden können und in vorhandene und zukünftige Infrastrukturen verteilter digitaler Bibliotheken integrierbar sein.

Das Speichersystem wird Mechanismen vorsehen um Dokumente dezentral zu verarbeiten und Funktionalitäten bereitstellen, die aktive Speicher unterstützen. In dieser Arbeit werden Speicher als aktiv bezeichnet, wenn sie die Übertragung und Speicherung von Dokumenten ermöglichen und es erlauben auf den Dokumenten definierte Operationen auszuführen.

Zusammenfassung der Ziele:

- Speicherung multimedialer Dokumente und Bereitstellung effektiver Zugriffsmöglichkeiten auf die Dokumente
- Bereitstellung von Operationen zur Unterstützung des Konzepts aktiver Speicher
- Skalierbarkeit
- Flexibilität bezüglich der Anbindung an vorhandene und zukünftige Gesamtsysteme

Zusätzlich sollen die getroffenen Entwurfsentscheidungen durch Realisierung eines Prototyps evaluiert werden.

In Kapitel 2 erfolgt eine Einführung in das Projekt „Digitale Bibliothek Frankfurt“ der Universität Frankfurt. Zuvor werden grundlegende Konzepte, wie die „Standard Generalized Markup Language“ (SGML), mobiler Code und allgemeine Probleme bei der Realisierung digitaler Bibliotheken besprochen.

Kapitel 3 beschäftigt sich mit dem Entwurf des Dokumentenspeichersystems als verteiltes System. Die verschiedenen Komponenten, die im System miteinander interagieren, werden vorgestellt und die Anordnung der Komponenten unter Berücksichtigung bekannter Architekturen hergeleitet.

Naturgemäß kommt den Komponenten, die für die Speicherung der Dokumente verantwortlich sind, ein besondere Rolle zu. In Kapitel 4 wird ein aktiver Speicher entwickelt, der Funktionalitäten zur Verwaltung von multimedialen Dokumenten aufweist und spezielle Eigenschaften zur Verarbeitung von SGML-Dokumenten bereitstellt.

Der Realisierung eines Prototyps, nebst Evaluation ist Kapitel 5 gewidmet. Die Implementierung der verschiedenen Komponenten des Dokumentenspeichersystems und ihr Zusammenwirken werden hier beschrieben.

Kapitel 6 schließlich faßt die Arbeit und erzielte Ergebnisse zusammen und gibt einen Überblick über verschiedene Möglichkeiten das Speichersystem weiterzuentwickeln.

Kapitel 2

Grundlagen

Dieses Kapitel dient der Begriffsbildung und der Darstellung der notwendigen Grundlagen. Zunächst wird ein kurzer Überblick über die „Standard Generalized Markup Language“ (SGML), über verschiedene Verfahren SGML-Dokumente zu verarbeiten und über mobilen Code gegeben. Weiterhin wird das Projekt „Digitale Bibliothek Frankfurt“ der Universität Frankfurt ausführlich beschrieben. Abschließend werden Probleme erläutert, die bei der Realisierung von digitalen Bibliotheken und Speichersystemen auftreten.

2.1 Geschichte und Grundlagen von SGML

SGML besteht seit 1986 als internationaler ISO-Standard 8879 „Information Processing - Text and office systems - Standard Generalized Markup Language“ (SGML, [ISO86]). Die Grundidee stammt von Dr. Ch. F. Goldfarb (IBM), der 1969 die „Document Composition Facility General Markup Language“ (DCF GML) entwickelt hat. DCF GML ist eine Makrosprache für ein Formatiersystem und daher vergleichbar mit Formaten wie NROFF/TROFF ([Sch90]), TeX([Knu84]) oder LaTeX([Lam86]).

DCF GML wurde später von dem Standardisierungsgremium ISO/IEC JTC1/-SC18/WG8 zu SGML verallgemeinert und um die Möglichkeit zur Festlegung einer Grammatik für die jeweilige Dokumentstruktur erweitert. Dadurch unterscheidet sich SGML von jeder anderen Dokumentbeschreibungssprache. Die Grammatik für den Aufbau eines Dokuments nennt sich Document Type Definition (DTD) oder im Deutschen Dokumenttypfestlegung. Die deutschen SGML-Fachbegriffe sind in DIN 28879 ([DIN91]) übersetzt worden, werden aber hier nur der Vollständigkeit halber erwähnt, da sie eher ungebräuchlich sind. In der DTD werden der hierarchische Aufbau, sowie die Reihenfolge der

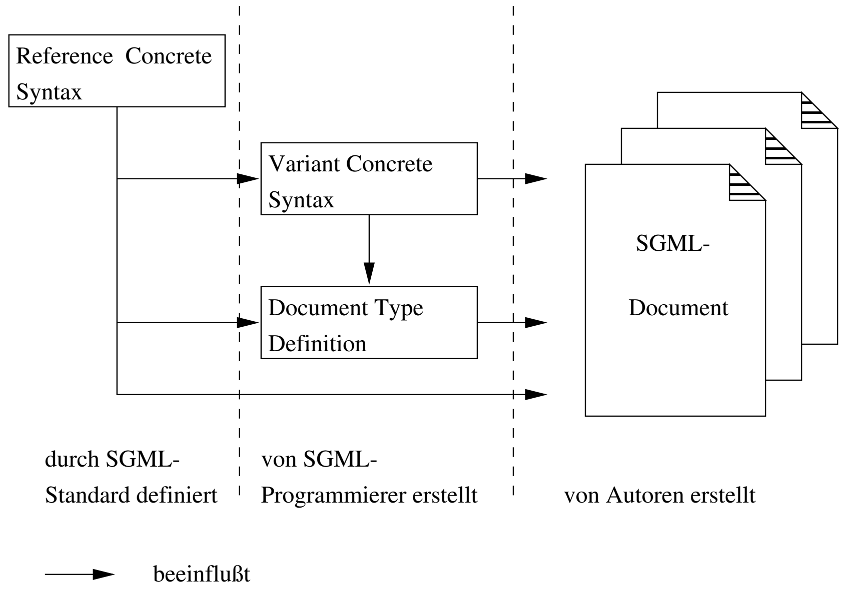


Abbildung 2.1: Aufbau von SGML-Dokumenten

Dokumentelemente festgelegt, durch sie werden die Tags für die daraus zu erstellenden Dokumente definiert. Den einzelnen Elementen können in der DTD Attribute zugeordnet werden, welche die Eigenschaften der Elemente beschreiben.

Im SGML-Standard ISO 8879 sind bestimmte Regeln zum Aufbau von DTD's und von den aus DTD's gebildeten Dokumenten festgeschrieben. Zudem werden in der „Reference Concrete Syntax" (dt. Bezugslexikalik) die Zeichen der SGML-Syntax im Standard vorgegeben. Es ist aber durchaus möglich diese anzupassen, in einer sogenannten „Variant Concrete Syntax" (dt. variante Lexikalik). Abbildung 2.1 zeigt die Zusammenhänge.

2.1.1 Merkmale von SGML

Der ISO-Standard definiert SGML als „eine Sprache für Dokumentenrepräsentation, welche Markup formalisiert und von System- und Verarbeitungsmöglichkeiten löst". SGML ermöglicht es, große Datenmengen soft- und hardwareunabhängig auszutauschen. Das Konzept der DTD läßt es zu, die Korrektheit jedes einzelnen Textelements innerhalb eines Dokuments zu überprüfen. Weiterhin sieht SGML Mechanismen vor, beliebige Informationstypen in ein Dokument einzubinden. SGML beschreibt nicht das Aussehen (Layout) des Dokuments, sondern seinen strukturellen Aufbau.

Durch die folgenden drei Punkte unterscheidet sich SGML von anderen Dokumentbeschreibungssprachen:

- Verwendung deskriptiven Markups
- Dokumenttypunterstützung
- Datenportabilität

Beim deskriptiven Markup werden die Tags (der Markup Code) dazu benutzt, die Teile des Dokuments strukturell zu klassifizieren. Im Gegensatz zum prozeduralen Markup werden die an die Tags gebundenen Operationen streng vom Markup getrennt. Durch deskriptiven Markup wird jedem Teil des Dokuments lediglich eine Semantik zugeordnet, das prozedurale Markup ordnet jedem Tag eine festvorgegebene Operation zu. Der Vorteil deskriptiven Markups ist, daß mehrere verschiedene Operationen auf das gleiche Dokument aufsetzen können und die jeweils gewünschten Dokumentteile bearbeitet werden. Ein Programm kann ein Inhaltsverzeichnis des Dokuments erstellen, ein anderes das Dokument in einer bestimmten Schriftart formatieren. Beispiele für prozedurales Markup sind alle gängigen Textverarbeitungsdateiformate, wie Word for Windows™ oder Word-Perfect™ Dateien. Deskriptives Markup findet man in den verschiedenen Darstellungsweisen von HTML-Seiten (Hyper Text Markup Language, [Tol96]) durch unterschiedliche HTML-Betrachter (Web-Browser) wieder.

SGML ermöglicht es, Dokumente zu klassifizieren und stellt dazu das Konzept der Document Type Definition (DTD) zur Verfügung. Der Typ eines Dokuments wird dabei durch seine wesentlichen Abschnitte und ihre Struktur formell definiert. Die Kodierung erfolgt nach dem ASCII-Standard ([ANS86]). Ein Buch könnte z.B. aus einem Titel, einem Autor und mehreren Kapiteln aufgebaut sein, wobei jedes Kapitel mehrere Unterkapitel bzw. Abschnitte enthalten kann. Diese Klassifizierung ermöglicht es, Dokumente auf ihre Zugehörigkeit bzw. Korrektheit zu einer, durch eine DTD definierte Dokumentenklasse zu prüfen. Weiterhin ist es möglich Operationen bzw. Algorithmen zu entwerfen, die auf einer ganzen Klasse von Dokumenten operieren können. Da die Algorithmen Kenntnis von der genauen Struktur der Dokumente haben, können sie effektiver und effizienter realisiert werden.

Ein grundlegendes Ziel bei der Standardisierung von SGML war, daß SGML-Dokumente unabhängig von Hard- und Software auf andere Computersysteme portierbar sein sollen. Die Portierung wird erheblich dadurch vereinfacht, daß das SGML-Dokument die korrespondierende DTD enthalten kann und sich somit selbst beschreibt. Dokumentenbeschreibung und Dokument bilden dann eine funktionale Einheit. Um diese Unabhängigkeit auf der Ebene von einzelnen Zeichen und Zeichenketten zu erlangen, stellt SGML einen Ersetzungsmechanismus bereit.

Die durch diesen Mechanismus definierten Zeichenketten heißen Entitäten. Entitäten werden beispielsweise eingesetzt, um die Zeichensätze verschiedener Computersysteme aufeinander abzubilden oder um eine einheitliche Nomenklatur in einer Menge von Dokumenten zu gewährleisten. Eine vollständige Erläuterung der Merkmale von SGML findet sich in [Gol88].

2.1.2 SGML in der Praxis

Dieser Abschnitt soll anhand eines kleinen Beispiels beschreiben, wie SGML in der Praxis benutzt wird. Im Besonderen soll auf die Zusammenhänge zwischen DTD und dem eigentlichen SGML-Dokument eingegangen werden.

```
<!DOCTYPE MEMO SYSTEM "memo.dtd">
<MEMO STATUS=PUBLIC>
<TO>Romeo
<FROM>Julia
<BODY>
Dritter Aufzug 5.Szene
<P> Willst Du schon gehen? Der Tag ist ja noch fern.
Es war die Nachtigall und nicht die Lerche,
die eben jetzt dein banges Ohr durchdrang;
Sie singt des Nachts auf dem Granatbaum dort.
Glaub, Lieber, mir: es war die Nachtigall.</P>
</BODY>
<CLOSE> gez. William Shakespeare</CLOSE>
</MEMO>
```

Abbildung 2.2: Ein SGML-Dokument

Wie man in Abbildung 2.2 erkennt, die ein kleines SGML-Dokument darstellt, werden die Tags durch spitze Klammern gekennzeichnet (z.B. <FROM>). Die Elemente eines SGML-Dokuments beginnen immer mit einem sogenannten Starttag (z.B. <P>) und enden evtl. mit einem Endtag (z.B. </P>). Zwischen den korrespondierenden Start- und Endtags steht der eigentliche Inhalt des Elements. Die Benennung der Tags ist in diesem Fall selbsterklärend gewählt (z.B. <TO> bezeichnet den Adressaten), dies ist lediglich eine Konvention und nicht zwingend notwendig.

Wie unter Kapitel 2.1 beschrieben fußt jedes SGML-Dokument auf einer DTD. Eine einfache DTD für das in Abbildung 2.2 dargestellte SGML-Dokument zeigt Abbildung 2.3.

```
<!ENTITY  %DOCTYPE "MEMO"                               >
<!ELEMENT %DOCTYPE     - - ((TO & FROM),BODY,CLOSE?) >
<!ELEMENT TO            - 0 (#PCDATA)                 >
<!ELEMENT FROM          - 0 (#PCDATA)                 >
<!ELEMENT BODY          - - (P)*                      >
<!ELEMENT P             - 0 (#PCDATA)                 >
<!ELEMENT CLOSE         - 0 (#PCDATA)                 >
<!ATTLIST MEMO  STATUS (CONFIDENTIAL|PUBLIC) PUBLIC  >
<!ATTLIST P     ID     ID                    #IMPLIED>
```

Abbildung 2.3: Eine einfache Document Type Definition

Die in der DTD deklarierten Markup-Komponenten beschreiben alle Elemente, Attribute und Entitäten und ihre strukturelle Zusammensetzung in einem SGML-Dokument, das auf dieser DTD basiert.

Das reservierte Tag !ENTITY deklariert Entitäten. Entitäten stellen Konstanten dar, die innerhalb der Dokumentinstanz ersetzt werden (z.B. %DOCTYPE wird durch den String MEMO ersetzt).

Elemente werden durch das reservierte Tag !ELEMENT deklariert. Für jedes Element wird ein Name (z.B. MEMO), die Art der Begrenzung durch Start- und Endtags (z.B. - -, für beide Tags müssen vorkommen oder - 0, für das Starttag muß vorkommen, das Endtag kann vorkommen) und der Beschreibung des Inhalts des Tags (z.B. (#PCDATA) für Text) angegeben.

Das reservierte Tag !ATTLIST deklariert die Attribute, die ein Tag haben darf oder haben muß (z.B. hat das Tag MEMO ein Attribut STATUS, das den Wert CONFIDENTIAL oder PUBLIC annehmen kann, der Standardwert ist PUBLIC).

Eine umfassende Einführung in SGML würde den Rahmen dieser kurzen Einleitung sprengen, hierzu sei auf [Bry88] und [Rie95] verwiesen.

2.2 Speichern von SGML-Dokumenten

Das Speichern von SGML-Dokumenten kann in vielfältiger Weise geschehen. Im folgenden sollen einige repräsentative Verfahren erläutert werden.

Man speichert bzw. archiviert Dokumente, weil man zu einem späteren Zeitpunkt wieder darauf zugreifen möchte. Die Art des Zugriffs resultiert im wesentlichen aus der Art, auf die das Dokument gespeichert wurde. Um komfortabel auf SGML-Dokumente zugreifen zu können, benutzt man Kenntnisse über die Struktur des Dokuments bereits beim Speichern. Diese Kenntnisse werden entweder aus der DTD, aus dem SGML-Dokument selbst oder aus beiden gewonnen. In allen Fällen versucht man das Wissen über den strukturellen Aufbau der SGML-Dokumente zu nutzen, um später effizienter und effektiver darauf zugreifen zu können.

Die verschiedenen Methoden ein SGML-Dokument zu speichern bieten verschiedene Möglichkeiten auf den SGML-Dokumenten zu operieren. Im allgemeinen unterscheidet man zwischen Operationen, die Kenntnis über den strukturellen Aufbau des Dokuments haben (strukturorientierte Operationen) und solchen, die ohne jegliche Kenntnis über die Struktur operieren (strukturunabhängige Operationen). Erstere operieren meist effizienter auf den Dokumenten, während zweitere meist mit wesentlich weniger Aufwand zu realisieren sind.

Ein wichtiges Merkmal eines SGML-Dokuments ist der Strukturbaum. Der Strukturbaum stellt die Tag-Hierarchie des Dokuments dar. Er wird benutzt, um ein SGML-Dokument darzustellen oder nur bestimmte Abschnitte zu bearbeiten. Exemplarisch wird der Strukturbaum des in Abbildung 2.2 gezeigten SGML-Dokuments in Abbildung 2.4 dargestellt.

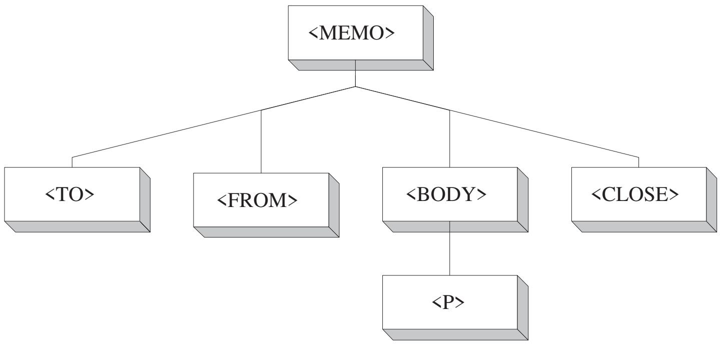


Abbildung 2.4: Der Strukturbaum des Beispiels

Der Strukturbaum beinhaltet nur den Markup des SGML-Dokuments, nicht aber die von den Tags eingeschlossenen Daten.

Grundsätzlich wird zwischen Markup und Daten unterschieden, mit Daten sind z.B. Texte, binäre Bilddaten, Tondaten, usw. gemeint. Unter einem Bitstrom versteht man eine Menge von Informationen, die bit- oder byteorientiert, z.B. als Datei auf einem Massenspeicher abgelegt sind. Mit Klassen sind die Dateien in einer Datenbank bezeichnet. Dieser Terminus kommt aus der Welt der objekt-orientierten Datenbanken (ODBMS), dort werden Datenstrukturen als Klassen bezeichnet.

2.2.1 Speichern als Bitstrom

Die einfachste Methode ein SGML-Dokument zu speichern, besteht darin, es als Bitstrom in einem Massenspeicher abzulegen. Bei der Speicherung selbst werden keine Eigenschaften des Dokuments berücksichtigt. Will man die so gespeicherten Dokumente durchsuchen oder verändern, so müssen sie sequentiell durchlaufen werden. Zur Unterstützung beim Durchsuchen großer Mengen auf diese Weise aufbewahrter Dokumente gibt es sogenannte SGML-Indizierer. Diese Programme erstellen vorab verschiedene Indizes (z.B. Hashtable, B-Tree), die es erlauben effizient strukturorientiert in den Dokumenten zu suchen. Alternativ stehen zur strukturunabhängigen Suche in Dokumenten sogenannte Volltextindizierer zur Verfügung. Diese ermöglichen die Suche nach Textmustern in einer Menge von Dokumenten, wobei es unerheblich ist, ob diese strukturiert sind oder nicht.

2.2.2 Ein objektorientierter Ansatz, VODAK

Ein objekt-orientierter Ansatz SGML-Dokumente zu speichern wird von der Gesellschaft für Mathematik und Datenverarbeitung (GMD) in Darmstadt im Rahmen des Projekts „VODAK“ verfolgt und ist in [BAN95] genau beschrieben. Das Projekt beschäftigt sich mit der Entwicklung eines objektorientierten Datenbanksystems (ODBMS) namens VODAK, das eigens für die Verwaltung von SGML-Dokumenten konzipiert wurde. In VODAK wird zu jeder DTD eine eigene Datenstruktur erstellt, die nur Dokumente enthält, die dieser DTD entsprechen. In Abbildung 2.5 wird ein Auszug aus einer solchen VODAK-Datenstruktur gezeigt. Für jeden SGML-Elementyp existiert eine Applikationsklasse in der Dokumententypschicht. Jede dieser Klassen ist von der Klasse *DOCUMENT_ELEMENT*, der dokumentunabhängigen Schicht abgeleitet. Zu jedem Element eines SGML-Dokuments existiert eine Instanz der dazugehörigen Applikationsklasse und eine Instanz vom Typ *DOCUMENT_ELEMENT*. In der dokumentunabhängigen Schicht finden sich Informationen über den Aufbau der Datenstruktur und Methoden um sich durch die Klassenhierarchie zu bewegen.

VODAK bildet also die Informationen aus der DTD auf eine Klassenhierarchie ab, das eigentliche SGML-Dokument wird dann durch Instanzen dieser Klassen beschrieben. Auf diese Weise ist es möglich, auf jedem einzelnen Elementtyp eines SGML-Dokuments Operationen zu definieren, die in Methoden der korrespondierenden Klasse realisiert werden. Ein SGML-Dokument kann man sich derart realisiert, als einen Baum vorstellen. Die Teilbäume des Baums sind Instanzen der verschiedenen Klassen der dokumenttypabhängigen Schicht. Die Methoden, um in diesem Baum suchen zu können, werden

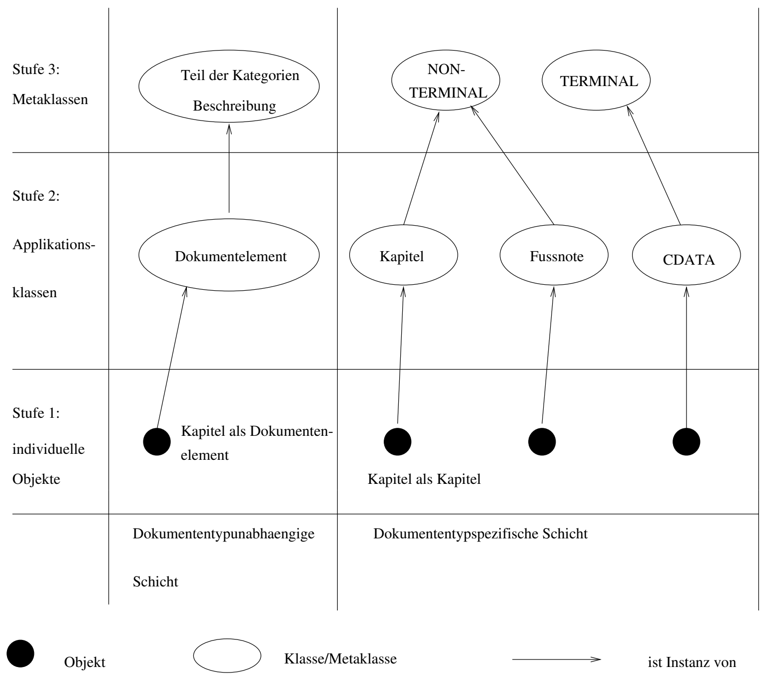


Abbildung 2.5: Das Klassenkonzept von VODAK

in der dokumentunabhängigen Schicht definiert und erlauben beispielsweise folgende Abfragen:

1. Finde alle Abschnitte deren Autor „Christian Mönch“ ist.
2. Finde das jeweils erste SGML-Element aller gespeicherten Dokumente.
3. Finde alle Tags (Applikationsklassen), die innerhalb des Tags „Kapitel“ auftreten können.

VODAK bietet uneingeschränkten Zugriff auf die gespeicherten SGML-Dokumente und die dazugehörigen DTD’s. Als Abfragesprache wurde die Vodak Query Language (VQL), für die Definition der Klassenhierarchie die Vodak Modelling Language (VML) entwickelt. Weiterführende Informationen findet man unter [KNS90] oder [BAH94].

2.2.3 Traditionelle Methoden für relationale Datenbanken

Die Struktur von SGML-Dokumenten bietet sich an, die SGML-Dokumente als Baumstruktur zu speichern. Als Beispiel soll wieder das in Abbildung 2.2 dargestellte Dokument dienen. Die in Abbildung 2.3 benutzte Darstellungsweise eignet sich nicht sonderlich gut zur Speicherung in einer Datenbank.

Man erkennt, daß jeder Knoten beliebig viele Kinder haben kann. Da man in Datenbankmanagementsystemen (DBMS) normalerweise keine dynamischen Variablen erzeugen kann, ist diese Darstellung nicht geeignet, um sie in eine Datenstruktur umzusetzen. Jeder Baum läßt sich so modifizieren, daß jeder Knoten maximal einen Vater, einen Sohn und einen Bruder hat. Die Idee dabei ist, daß der zweite Sohn eines Knotens immer der rechte Bruder des ersten Sohns des Knotens ist. Durchsucht man den Baum von einem Knoten aus, so findet man den ersten Sohn direkt von diesem Knoten aus, alle weiteren als Brüder des ersten.

Für das genannte Beispiel stellt sich das wie in Abbildung 2.6 gezeigt dar.

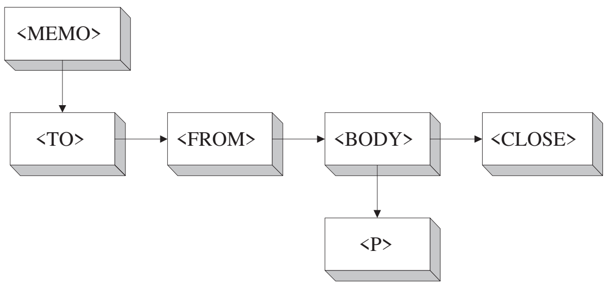


Abbildung 2.6: Vereinfachter Strukturbaum des Beispiels

Mit dieser Struktur kann man einen Baum viel leichter in einer Datenbank abspeichern. Jetzt hat jeder Knoten maximal zwei Verweise, die es sich zu merken gilt.

Eine Möglichkeit ist, den Strukturbaum in einer Klasse zu speichern und die Inhalte der Elemente (Daten) in einer anderen Klasse. Dabei können die Daten entweder direkt in die Datenbank eingebunden werden oder nur als Referenz auf Bitströme vermerkt sein. Zu jedem Knoten des Baums werden eine laufende Nummer, die Nummer des Vaters, des Sohnes und des Bruders gespeichert. Auf diese Art kann man den Baum bequem mittels Depth-First-Search ([Knu73]) traversieren und dabei die gewünschte Operation ausführen.

Eine weitere Möglichkeit ist, jede DTD einer Bewertung zu unterziehen und bestimmte Teile des Markups als Index-Tags auszuzeichnen. Die Index-Tags und deren beinhaltete Daten werden in einer Klasse vorgehalten, die restlichen Informationen in einer anderen. Dadurch verkleinert man die zu durchsuchende Datenmenge. Die Bewertung der DTD's muß allerdings manuell durchgeführt werden, was einen nicht unerheblichen Mehraufwand bedeutet.

2.3 Mobiler Code

Das Konzept „Mobilen Codes“ stellt einen Ansatz dar, konfigurierbare Dienste in Netzwerken anzubieten. Dabei unterscheidet man zwischen Diensterbringern und Dienstbenutzern. Der Diensterbringer bietet eine Leistung an, der Dienstbenutzer nimmt diese Leistung in Anspruch.

Wird mobiler Code zwischen zwei Instanzen eines Netzwerks eingesetzt, so sendet der Dienstbenutzer ein Programm bzw. eine Prozedur an den Diensterbringer. Der Diensterbringer führt das Programm innerhalb einer Ablaufumgebung aus und schickt entweder eventuelle Ergebnisse zurück oder aber der Rückmeldevorgang ist im Programm selbst integriert.

Diensterbringer, die keine Möglichkeit vorsehen, mittels mobilen Codes auf ihre Ressourcen zuzugreifen, legen fest auf welche Weise ein Dienstbenutzer den angebotenen Dienst verwenden muß. Durch Bereitstellung einer Ablaufumgebung und geeigneter Bibliotheken auf seiten des Diensterbringers erreicht man Flexibilität bzgl. der Art des Zugriffs. Der Dienstbenutzer bestimmt selbst wie er den angebotenen Dienst nutzt, er kann z.B. festlegen in welcher Form die Rückmeldungen erfolgen sollen. Zur Realisierung derartiger Diensterbringer eignen sich weitgehend plattformunabhängige Interpretersprachen wie Java, Tcl, Perl, Python, usw. Compilersprachen sind weniger geeignet, da sie im allgemeinen das Vorhandensein von bestimmten plattformabhängigen Bibliotheken voraussetzen und weniger portierbar sind.

Ein Problem beim Einsatz von mobilem Code ist die Sicherheit. Es muß gewährleistet werden, daß der vom Dienstbenutzer übermittelte mobile Code keine Möglichkeit hat mehr Privilegien zu erhalten, als ihm vom Diensterbringer erteilt wurden. Ein konkretes Beispiel wie mobiler Code in Netzwerken bzw. verteilten Systemen eingesetzt werden kann findet sich in [LDD95].

2.4 Eine Architektur für verteilte digitale Bibliotheken

Dieser Abschnitt beschäftigt sich mit einer Architektur für verteilte digitale Bibliotheken, die seit Januar 1997 an der Universität Frankfurt/Main an der Professur für Telematik und verteilte Systeme im Rahmen des Projektes „Digitale Bibliothek Frankfurt“ entwickelt wird.

Zunächst sollen die Ziele des Projektes umrissen werden, daran schließen sich Begriffsbildung und eine Erläuterung des derzeitigen Status des Projekts an.

2.4.1 Ziele des Projekts

Das Projekt „Digitale Bibliothek Frankfurt“ soll eine Infrastruktur für verteilte digitale Bibliotheken hervorbringen. Das resultierende Konzept wird es ermöglichen, die verschiedensten Arten von digitaler Information in verteilten Systemen zu verwalten und bereitzustellen.

Das Konzept sieht vor, auch neue Informationstypen mit geringem Aufwand zu integrieren. Desweiteren wird die Infrastruktur derart skalierbar sein, daß sie an zukünftige Anforderungen angepaßt werden kann. Eine detailierte Übersicht über Umfang und Ziele des Projekts liefert der Projektantrag [DK95].

2.4.2 Aufbau der Architektur

Eine digitale Bibliothek operiert auf digitalen Dokumenten. Ein digitales Dokument kann man sich als Menge digitaler Information vorstellen (z.B. ASCII Text, Bilder, Ton oder Videodaten). Um ein digitales Dokument innerhalb eines Gesamtsystems bearbeiten zu können, bedarf es einiger Verwaltungsinformationen. Es sollen mindestens folgende Daten zu jedem Dokument verfügbar sein:

- eine eindeutige Kennung des Dokuments innerhalb des Systems
- Autor, Titel und Eigentümer des Dokuments innerhalb des Systems

Auf den digitalen Dokumenten können verschiedene Operationen ausgeführt werden. Eine bestimmte Menge von Operationen, die sogenannten Basisoperationen, müssen auf jedem Dokument definiert sein. Basisoperationen sind das Speichern, das Durchsuchen oder das Präsentieren eines Dokuments. Alle zusätzlich definierten Operationen sind optional. Um ein digitales Dokument typtransparent verarbeiten zu können, werden jedem Dokument Operationen zugeordnet.

Die Operationen werden in sogenannten Dokumentenmethoden realisiert. Jedes Dokument kann eigene Dokumentenmethoden besitzen. Eine Dokumentenmethode oder dokumentspezifische Methode definiert eine bestimmte Operation auf einem bestimmten Dokument. Für jedes Dokument wird eine Menge von Dokumentenmethoden definiert. Dadurch erreicht man die gewünschte Unabhängigkeit vom Typ der im Dokument enthaltenen Information (Typtransparenz).

Die dem Gesamtsystem bekannten Informationen über ein Dokument sind:

- das Dokument
- die Dokumentenmethoden
- die Verwaltungsinformationen

Diese Informationen werden in einem Metadokument zusammengeführt. Für jedes digitale Dokument gibt es ein Metadokument, das alle bekannten Informationen über das Dokument beinhaltet. Metadokumente kann man sich als eine Art Behälter für die Dokument-Informationen vorstellen. Die Informationen werden entweder als Bezug (Referenz) oder direkt in das Metadokument eingebunden. Der Aufbau eines Metadokuments wird systemweit festgelegt, um einen einheitlichen Zugriff zu gewährleisten. Metadokumente werden in Form eines SGML-Dokuments realisiert, die zugehörige DTD findet sich in Anhang A. Alternativ können Metadokumente auch in anderer Form vorliegen, beispielsweise in reinem ASCII-Text. Abbildung 2.7 zeigt ein Beispiel für ein Metadokument.

```
<!DOCTYPE META SYSTEM "/usr/users2/DiBiF/Meta/meta.dtd">
<META id="test-000" version="0.1">
<HEADER>
    <INFO>Testmetadokument</INFO>
    <PROPERTY title="metadoc1.sgml" author="Hans Matzen"
              owner="Lehrstuhl Telematik"
              ident="TM001.HM1097"
              pubkey="mQBmAjFjzdkAAAECy">

    <METHOD name="Store"
            code="URL:http://www/~hans/tstore.py"
            mime="x-application/x-python">
    <METHOD name="Search"
            code="URL:http://www/~hans/tsearch.py"
            mime="x-application/x-python">
    <METHOD name="Present"
            code="URL:http://www/~hans/tpresent.py"
            mime="x-application/x-python">
</HEADER>
<DATA extern=URL:http://www/~hans/tdoku.sgml></DATA>
</META>
```

Abbildung 2.7: Beispiel für ein Metadokument

Ein Metadokument besteht aus einem Headerbereich und einem Databereich. Im Headerbereich finden sich die Verwaltungsinformationen und die Dokumentenmethoden, im Databereich steht das eigentliche Dokument. Von besonderem Interesse sind die Methoden-Tags. Außer dem Namen der Methode, werden der Ort in Form eines „Uniform Ressource Names“ (URN, [SM94]) und der „Multipurpose Internet Mail Extensions“-Typ (MIME, [BF93]) der Methode angegeben. Über den MIME-Typ kann eine ausführende Instanz ermitteln, in welcher Programmiersprache die Methode implementiert ist. Es ist möglich eine Methode in mehreren Programmiersprachen zu implementieren und damit eine höhere Verfügbarkeit zu erlangen. Die durch die Metadokumente beschriebenen digitalen Dokumente bringen ihre Funktionalität mit.

Die ausführenden Instanzen heißen Elemente. Unter Elementen versteht man einen oder eine Menge von Prozessen, die innerhalb des Systems eine bestimmte Aufgabe wahrnehmen. Es gibt verschiedene Ausprägungen von Elementen. Zum Beispiel:

- Katalogelemente, im Sinne eines bibliographischen Katalogs
- Präsentationselemente, die Dokumente zur Darstellung aufbereiten
- Speicherelemente, die der Speicherung und dem Zugriff auf Dokumente dienen

Die Elemente sind über eine definierte Schnittstelle an eine zum Gesamtsystem gehörende Infrastruktur verbunden, die Transportmechanismen bereitstellt, damit die Elemente miteinander kommunizieren können. Diese Schnittstelle definiert mindestens drei Kommandos. Ein Kommando dient der Übetragung von Dokumenten an das Element, eines der Ausführung von Dokumentenmethoden auf übertragenen Dokumenten und eines dem Abruf von Informationen über das Element und die von ihm bereitgestellten Funktionalitäten. Die Schnittstelle legt gleichzeitig die Basisfunktionalität aller Elemente fest.

Die Elemente stellen nur ihre Ressourcen zur Benutzung durch die Dokumentenmethoden zur Verfügung. In herkömmlichen Systemen findet man die Funktionalitäten in den Elementen realisiert. Damit sind die Möglichkeiten, Funktionalität auf Dokumentinhalte abzustimmen, beschränkt. Durch die Verbindung von Funktionalität und Dokument schafft man hier die Unabhängigkeit von den Dokumentinhalten und höchste Anpassungsmöglichkeiten an neue Dokumenttypen.

In Abhängigkeit von ihrer Funktionalität stellen die Elemente verschiedene Programmbibliotheken bereit, die von den Dokumentenmethoden benutzt werden, um auf die Ressourcen der Elemente zuzugreifen. Diese Bibliotheken werden nach ihrer Funktionalität gruppiert in sogenannten Paketen bereitgestellt. Jedes Paket realisiert eine Menge von Methoden (Operationen) zum Einsatz in den Dokumentenmethoden. Die bekannte Problematik beim Zusammenstellen von Operationen in Bibliotheken wächst durch den Anspruch an die Dokumentenmethoden, die Kenntnisse über elementspezifische Eigenschaften durch die Operationen zu verbergen. Das heißt, die Pakete stellen die Schnittstelle zwischen Methoden und Elementen dar.

Auf diese Weise können auf jedem Dokument typgerechte Operationen definiert werden. Soll eine Operation innerhalb eines Elements auf einem Dokument ausgeführt werden, so muß das Element keine Kenntnis über den Inhalt des Dokuments haben.

Jedes Element stellt mindestens eine Ablaufumgebung für Dokumentenmethoden bereit. Dokumentenmethoden werden ausschließlich innerhalb dieser Ablaufumgebungen der Elemente ausgeführt. Ist ein Metadokument zu einem Element übertragen worden, kann durch Senden einer sogenannten Aktivierung das Element dazu aufgefordert werden, eine bestimmte Dokumentmethode auf dem zugeordneten Dokument auszuführen. Dokumente werden entweder in den Metadokumenten oder aber über eine von den Methoden aufzubauenden separaten Netzverbindung übertragen. Zwischen Elementen werden lediglich Metadokumente, Aktivierungen und Rück- bzw. Fehlermeldungen übertragen. Abbildung 2.8 verdeutlicht ein Szenario dieser Art.

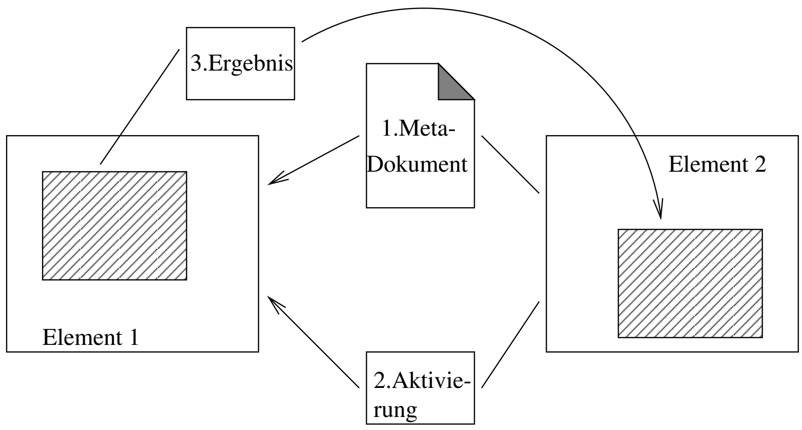


Abbildung 2.8: Kommunikation zwischen Elementen

Abschließend sei bemerkt, daß sich diese Architektur noch in der Entwicklung befindet und die hier erläuterten Sachverhalte keinesfalls einen endgültigen Stand repräsentieren. Eine ausführliche Beschreibung der Architektur ist in [MD98] gegeben.

2.5 Aspekte der Realisierung digitaler Bibliotheken

Dieser Abschnitt gibt einen kurzen Überblick über die Unterschiede zwischen konventionellen und digitalen Bibliotheken, sowie über die Probleme, die bei der Entwicklung digitaler Bibliotheken auftreten.

Die Bestandteile einer Bibliothek lassen sich in folgendes grobes Klassenschema unterteilen:

- Informationen (z.B. Bücher, Zeitschriften, Filme, multimediale digitale Dokumente)
- Metainformationen (z.B. Büchereikataloge, dynamische Indizes)
- Vorgänge bzw. Operationen (z.B. Ausleihen eines Buches, Volltextsuche)

Möchte man Informationen, die zuvor in einer konventionellen Bibliothek gelagert waren, in einer digitalen Bibliothek aufbewahren, so müssen diese Informationen in digitale Informationen übersetzt werden. Die Problematik einer solchen Übersetzung ist vielfältig. So können z.B. die Seiten eines Buches mittels eines Scanners digitalisiert und durch eine Texterkennungssoftware in ASCII-Text umgewandelt werden, das äußere Erscheinungsbild (Ledereinband, Goldschnitt) eines Buchs hingegen kann nicht digitalisiert werden. Die Übersetzung bibliographischer Metadaten ist schwierig, weil Metadaten und ihre Funktionalität meist von ihren physikalischen Gegebenheiten beeinflußt werden. So beinhaltet z.B. die räumliche Anordnung der Bücher in den Regalen einer konventionellen Bibliothek eine Semantik, die in digitalen Bibliotheken schwer nachzubilden ist. Die meisten Vorgänge in konventionellen Bibliotheken werden heute noch von Menschen ausgeführt (z.B. Beratung bei der Suche eines Buchs), das Problem der Übersetzung dieser Vorgänge resultiert daraus, daß sie meist informell und äußerst komplex sind. In [NFL$^+$95] findet sich ein detailiertere Beschreibung der angesprochenen Probleme.

Aber auch neue digitale Bestandteile werden in digitalen Bibliotheken verwaltet. Dazu gehören beispielsweise digitale multimediale Bücher. Digitale Metainformationen, wie z.B. Volltextindizes zur Suche in einer großen Menge von Informationen oder persönliche Anmerkungen, die der Leser eines Buches angegeben hat, bieten die Möglichkeit wesentlich detailliertere Metainformationen zu den Informationen vorzuhalten.

Bei der Realisierung digitaler Vorgänge oder Operationen gilt es zu beachten, daß diese evtl. große Mengen von Informationen und Metainformationen verarbeiten müssen, um ihre Aufgabe wahrzunehmen. Zu diesen Operationen gehören Volltextsuche, Einsatz von mobilen Agenten zur Informationsgewinnung und individuelle Darstellung der Informationen.

In Tabelle 2.1 werden die verschiedenen Bestandteile einer digitalen Bibliothek nach den genannten Kriterien zusammengefaßt.

	Informationen	**Metainformationen**	**Vorgänge/ Operationen**
Übersetzungen aus konventionellen Bibliotheken	Bücher, Zeitschriften, Filme	Büchereikatalog, räumliche Aufteilung	Beratung bei der Suche, Ausleihen eines Buchs
Zusätzliche Bestandteile digitaler Bibliotheken	multimediales digitales Dokument, Computerprogramme	dynamische Indizes, Anmerkungen zu Veröffentlichungen	Volltextsuche, individuelle Darstellung, Suche durch mobile Agenten

Tabelle 2.1: Bestandteile digitaler Bibliotheken

Eine digitale Bibliothek hat zum einen die Aufgabe Bestandteile einer konventionellen Bibliothek auf eine digitale Welt abzubilden, zum anderen neue, ursprünglich digitale Bestandteile aufzubewahren. In digitalen Bibliotheken sind Informationen und Metainformationen vollständig digitalisiert, während in konventionellen Bibliotheken Metainformationen nur teilweise (Kataloge) und Informationen meist gar nicht digitalisiert sind.

Da durch die Art und Weise wie Dokumente gespeichert werden, auch die Zugriffsmöglichkeiten definiert werden, trifft man besonders bei der Entwicklung von Speichern bzw. Speichersystemen auf die beschriebenenen Probleme. Die Komplexität multimedialer Dokumente, sowie die Größe der dabei entstehenden Datenmengen erfordern neue Konzepte zur Speicherung und den Zugriff darauf. Eine Architektur, die Vorteile verteilter Systeme nutzt, um diesen Anforderungen zu genügen, ist in [Bir95] beschrieben.

Teile der Funktionalitäten, die zum Zugriff auf in Speichersystemen befindliche Dokumente benötigt werden, werden von Massenspeichern (z.B. Bitstromspeicher), andere Teile von Datenbankmanagementsystemen (DBMS) bereitgestellt. DBMS besitzen eine Abfragesprache (z.B. die Structured Query Language, [ISO92]), die es ermöglicht auf die Merkmale der Dokumente zuzugreifen. Der Zugriff erfordert jedoch ein bestimmtes Wissen über die vorhandenen Merkmale.

Um auch multimediale Dokumente in einem DBMS verwalten zu können, bedarf es weiterer neuer Funktionalitäten. Beispielsweise muß es möglich sein, gespeicherte Audiodaten in verschiedenen Formen zur Verfügung zu stellen. Während das Anhören von Audiodaten lediglich einen rein sequentiellen Zugriff erfordert, bedarf die Bearbeitung (Schneiden) von Audiodaten auch Möglichkeiten zum Vor- und Zurückspulen.

Kapitel 3

Entwurf des Speichersystems

Dieses Kapitel beschäftigt sich mit der Entwicklung einer Architektur für ein Dokumentenspeichersystem zum Einsatz in verteilten digitalen Bibliotheken, das die in Kapitel 1 beschriebenen Ziele verwirklicht. Zunächst werden die Anforderungen an ein solches Speichersystem dargestellt. Es folgt die Entwicklung der grundlegenden Architektur des Speichersystems. Die getroffenen Entwurfsentscheidungen basieren auf den zuvor formulierten Anforderungen an ein solches Speichersystem. Anschließend soll zu jeder Komponente ein detailierter Überblick über Funktionalität und Integration in das Speichersystem gegeben werden.

3.1 Anforderungen

Die Anforderungen an das Dokumentenspeichersystem werden im folgenden aus den in Kapitel 1 angegebenen Zielen dieser Arbeit abgeleitet.

Das Dokumentenspeichersystem soll unabhängig von Anzahl und Größe der Dokumente einsetzbar sein, deshalb wird eine Verteilung innerhalb des Speichersystems vorgenommen. Es handelt sich hierbei um eine Verteilung im Sinne eines verteilten Systems. Das verteilte System soll konfigurierbar und steuerbar sein, um einen möglichst flexiblen Lastverbund zu erhalten.

Um sicherzustellen, daß das Speichersystem an vorhandene und zukünftige Gesamtsysteme digitaler Bibliotheken angebunden werden kann, soll eine weitgehende Modularisierung dafür Sorge tragen, daß das Speichersystem mit geringem Aufwand an neue Gegebenheiten (z.B. Veränderung des Übertragungsprotokolls zur Infrastruktur) angepaßt werden kann.

Zusammenfassung der Anforderungen:

- Verteilung innerhalb des Dokumentenspeichersystems (Lastverbund)
- Konfigurierbarkeit des verteilten Systems zwecks Steuerung des Lastverbundes
- Modularer Aufbau zwecks Anbindung an Gesamtsysteme digitaler Bibliotheken

3.2 Das Speichersystem, ein verteiltes System

Dieser Abschnitt entwickelt die Architektur des Speichersystems. Der grundlegende Aufbau des Speichersystems unterscheidet zunächst zwischen drei verschiedenen Arten von Komponenten, die im Speichersystem Verwendung finden können.

- Schnittstellenkomponenten
- Speicherkomponenten
- Vermittlerkomponenten

Unter Schnittstellenkomponenten versteht man Komponenten, die die Kommunikation mit der Infrastruktur bzw. dem Gesamtsystem durchführen. Ein Speichersystem kann mehrere Schnittstellenkomponenten besitzen. Diese können alle mit dem gleichen Gesamtsystem interagieren oder es ermöglichen, das Speichersystem an mehrere digitale Bibliotheken parallel zu verbinden. Über die Schnittstellenkomponenten erhält das Speichersystem die auszuführenden Aufträge. Die Ergebnisse eines Auftrages werden dann ebenfalls durch die Schnittstellenkomponente an den Auftraggeber geschickt.

Die Speicherkomponenten haben die Aufgabe Informationen aufzubewahren und diese Informationen auf Anfrage wieder zur Verfügung zu stellen. Weiterhin stellt ein Speicher eine oder mehrere Ablaufumgebungen und die dazugehörigen Programmbibliotheken für Dokumentenmethoden zur Verfügung, die es ermöglichen, die in Kapitel 2.4.2 erwähnten dokumentspezifischen Operationen speicherseitig auszuführen. Einige Operationen müssen speicherseitig ausgeführt werden, wie zum Beispiel der Vorgang des Speicherns, andere wie das Durchsuchen eines Dokuments können speicherseitig ausgeführt werden und wieder andere können nicht speicherseitig ausgeführt werden, beispielsweise die Darstellung (Präsentation) eines Dokuments.

Prinzipiell können Speicher für jede spezielle Art von Information (Videospeicher, Audiospeicher, u.s.w.) in das Speichersystem integriert werden. In Kapitel 4 wird ein aktiver Speicher behandelt, der zum einen exemplarisch in das Speichersystem eingebunden wird, zum anderen auf die speziellen Bedürfnisse von SGML-Dokumenten ausgerichtet ist.

Vermittlerkomponenten sorgen für die Verteilung der Aufträge auf die Speicher und für die Beantwortung jedes eingegangenen Auftrags. Weiterhin überwacht eine Vermittlerkomponente die Verbindungen zu Schnittstellen- und Speicherkomponenten. Vermittlerkomponenten tragen damit wesentlich zur konsistenten Verarbeitung der Auträge bei.

Die verschiedenen Komponententypen kommunizieren über definierte Schnittstellen unter Verwendung einer zuverlässigen Netzverbindung (z.B. TCP/IP).

Die verschiedenen Arten das Speichersystem als verteiltes System zu konzipieren unterscheiden sich durch die Ausprägung bzw. das Vorhandensein der Vermittlerkomponenten.

Ausgehend von der Aufteilung der Funktionalitäten auf die unterschiedlichen Komponententypen werden in den folgenden Abschnitten drei verschiedene Ansätze untersucht. Zum einen wird das klassische Client-Server-Paradigma (Two-Tier-Architektur) betrachtet, zum anderen zwei Ausprägungen der Three-Tier-Architektur. Die Ansätze werden beschrieben, auf Verwendbarkeit im vorliegenden Falle untersucht und die Auswahlentscheidung dargestellt.

3.2.1 Die Two-Tier-Architektur (klass. Client-Server-Paradigma)

Die Two-Tier-Architektur setzt voraus, daß die Schnittstellenkomponente (Client) weiß an welche(n) Speicher (Server) sie sich wenden muß, wenn sie einen Auftrag ausführen lassen möchte. Bei dieser Architektur sind die Vermittlerfunktionalitäten auf Client und Server verteilt, so daß keine gesonderte Vermittlerkomponente eingesetzt wird.

Will man einen Speicher hinzufügen, so muß man entweder die Adresse des neuen Speichers allen Schnittstellenkomponenten mitteilen oder eine zusätzliche Schnittstellenkomponente starten, die dessen Adresse kennt. Die Adresse der neuen Schnittstellenkomponente muß dann allerdings publiziert werden, damit von außen Aufträge an die Schnittstellenkomponente geschickt werden können. Um allen Schnittstellenkomponenten die Adresse eines neuen Speichers mitzuteilen, muß entweder der Speicher alle Schnittstellenkomponenten kennen oder diese müssen manuell umkonfiguriert werden.

Möchte man einen Speicher aus dem System ausgliedern, so ist ein ähnlich komplizierter Vorgang durchzuführen, nur werden die Adressen des Speichers diesmal aus der Konfiguration der Schnittstellenkomponenten gelöscht oder die korrespondierende Schnittstellenkomponente wird ebenfalls ausgegliedert und dies wieder publiziert. Das Speichersystem muß also ganz oder teilweise angehalten werden, um neue Speicher ein- oder auszugliedern. Desweiteren müssen Mechanismen vorgesehen werden, die es ermöglichen Aufträge zu serialisieren und nacheinander auszuführen. Diese Mechanismen können entweder in die Schnittstellenkomponente oder in den Speicher integriert werden. Abbildung 3.1 veranschaulicht das Szenario.

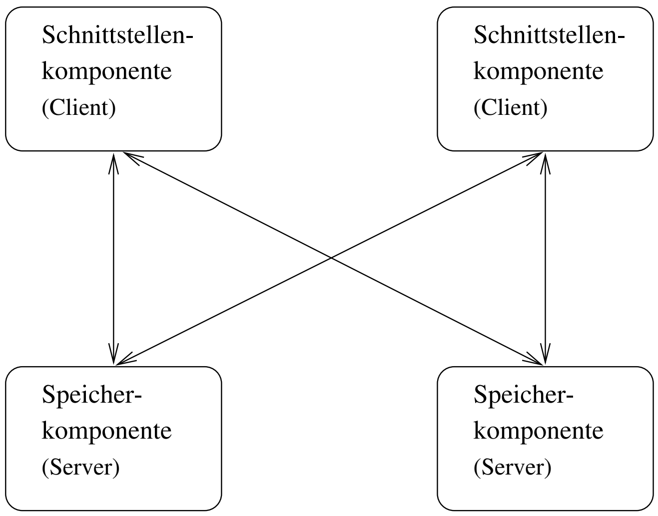


Abbildung 3.1: Client-Server-Ansatz

3.2.2 Die Three-Tier-Architektur mit Trader

Bei der Three-Tier-Architektur mit Trader setzt man eine Vermittlerkomponente (Trader) ein, die auf Anfragen einer Schnittstellenkomponente (Client), dieser mitteilt wo ein Speicher (Server) zu finden ist.

Die Schnittstellenkomponente fragt zuerst beim Trader nach der Adresse eines Speichers und wendet sich dann direkt an den Speicher. Durch diese Vermittlungsinstanz ist es möglich zur Laufzeit neue Speicher in das Speichersystem einzubinden oder auszugliedern. In Abbildung 3.2 ist diese Anordnung schematisch dargestellt.

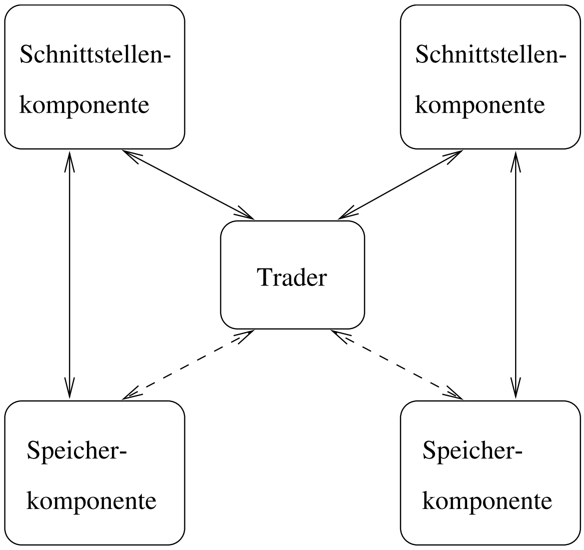


Abbildung 3.2: Three-Tier mit Trader-Ansatz

Die Speicher melden sich beim Trader an bzw. ab, wenn sie gestartet oder terminiert werden. Zusätzlich muß der Speicher in der Lage sein, während der Ausführung eines Auftrags neue Aufträge entgegenzunehmen, eingehende Aufträge zu sammeln und nacheinander abzuarbeiten, da weder der Trader noch die Schnittstellenkomponente wissen, ob der Speicher im Leerlauf ist und auf eingehende Aufträge wartet. Dieses Problem kann behoben werden, wenn man zwischen Trader und allen Speichern eine Art Steuerverbindung aufbaut, über die der Speicher dem Trader mitteilt, in welchem Zustand er sich befindet. Über diese Steuerleitung können die Speicher auch gesteuert werden (z.B. Anhalten eines Speichers).

Zu inkonsistenten Zuständen des Speichersystems kommt es dann, wenn die Datenübertragungszeiten zwischen Schnittstellenkomponente und Speicherkomponente stark von den Laufzeiten zwischen Speicherkomponente und Trader abweichen.

Ist die Datenübertragungszeit zwischen Speicher und Trader wesentlich größer als die zwischen Schnittstellenkomponente und Trader, dann kann es passieren, daß der Trader gleichzeitig mehrere Aufträge an einen Speicher vermittelt, weil die Nachricht an den Trader über die Zustandsveränderung des Speichers noch nicht vom Trader empfangen wurde. Analog kann es zu einer Situation kommen, in der der Trader glaubt alle Speicher wären beschäftigt, obwohl diese ihre Aufträge bereits abgearbeitet haben. Im umgekehrten Fall tritt eine ähnliche Situation ein, da sich gleichzeitig mehrere Aufträge auf dem Weg zum Speicher befinden können.

Eine gleichmäßige Auslastung des Systems kann dann nicht mehr gewährleistet werden. Sieht man entsprechende Rückmeldemechanismen vor, können auch diese Probleme gelöst werden.

3.2.3 Die Three-Tier-Architektur mit Broker

Die Three-Tier-Architektur mit Broker verwendet eine Vermittlerinstanz (Broker), die man als zentralen Makler für Aufträge verstehen kann. Abbildung 3.3 veranschaulicht den Aufbau dieses Ansatzes.

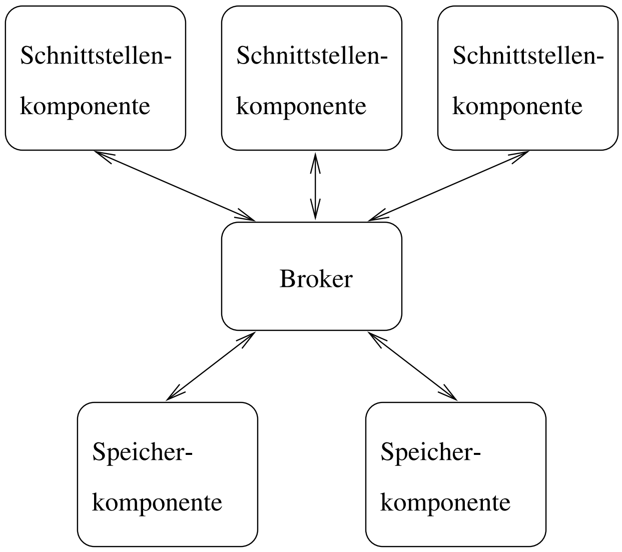


Abbildung 3.3: Three-Tier mit Broker-Ansatz

Die Schnittstellenkomponenten (Clients) kennen die Adresse des Brokers, ebenso die Speicher (Server). Die Speicher melden sich beim Broker an und ab, wenn sie ein- oder ausgegliedert werden. Eine Schnittstellenkomponente übermittelt einen Auftrag an den Broker, dieser prüft welche Speicher verfügbar sind und leitet den Auftrag an diesen Speicher weiter. Der Broker wartet auf die Antwort vom Speicher und sendet sie an die Schnittstellenkomponente.

Will man einen Speicher eingliedern, dann muß der Speicher nur die Adresse des Brokers kennen, er meldet sich an und ist ab diesem Zeitpunkt Bestandteil des Speichersystems. Das Ausgliedern eines Speichers kann direkt durch den Broker veranlaßt werden oder durch ein Signal an den Speicher erfolgen, der sich nach Ausführung des evtl. laufenden Auftrags beim Broker abmeldet. Der Broker kann seinerseits ein Kommando an den Speicher senden, der sich dann beendet.

Der Broker ist also immer über den Zustand aller Speicher informiert und kann diese steuern. Auch wenn ein Speicher „abstürzt" wird dies vom Broker bemerkt (Timeout, Lost-Connection) und eine Fehlermeldung an die Schnittstellenkomponente gesendet oder sogar der Auftrag an einen anderen Speicher geschickt.

Zu Performanceproblemen kann es kommen, wenn die Aufträge große Datenmengen beinhalten und der Broker zu lange braucht um sie an die Speicher „durchzureichen". Zu Überlastungen der Speicher kann es nicht kommen, da der Broker den nächsten Auftrag erst an den Speicher weiterleitet, wenn der Speicher verfügbar ist. Bei diesem Aufbau benötigt man deshalb keine Mechanismen zur Serialisierung der Aufträge in den Schnittstellenkomponenten oder den Speichern. Sollten einmal mehr Aufträge an den Broker übergeben werden als Speicher verfügbar sind, können diese entweder vom Broker zwischengespeichert werden bis ein Speicher frei wird oder der Broker gibt eine entsprechende Fehlermeldung an die Schnittstellenkomponente zurück.

3.2.4 Entscheidung für die Three-Tier-Architektur mit Broker

Für das zu entwerfende Speichersystem ist es von nachhaltiger Wichtigkeit, daß sowohl die Anpassung der Schnittstellenkomponenten, wie der Speicher mit geringem Aufwand durchführbar ist. Gleichzeitig sollen die Speicher als Lastverbund fungieren und auf Grund dessen eine gleichmäßige Auslastung gewährleistet sein.

Ein leistungsfähiger Lastverbund zeichnet sich dadurch aus, daß man ihn zur Laufzeit an quantitative Bedürfnisse anpassen kann, eine gleichmäßige Verteilung gewährleistet ist und man den „status quo" des Lastverbundes und damit der einzelnen Speicher jederzeit ermitteln kann. Dabei bedeutet gleichmäßige Verteilung, daß die einzelnen Speicher gleichermaßen mit Aufträgen versorgt werden.

Architektur *Leistungsmerkmal*	**Two-Tier**	**Three-Tier mit Trader**	**Three-Tier mit Broker**
Client transparentes Ein- und Ausgliedern von Speichern	nein	ja	ja
kontrollierte Ausführung der Aufträge	nein	bedingt (nur durch zusätzlichen Aufwand zu realisieren)	ja
gleichmäßige Verteilung	nein	bedingt (nur durch zusätzlichen Aufwand zu realisieren)	ja

Tabelle 3.1: Leistungsmerkmale der Architekturen

In Tabelle 3.1 sind die Leistungsmerkmale der verschiedenen Ansätze zusammengefaßt. Es ist ersichtlich, daß nur der Three-Tier-Ansatz mit Broker alle Merkmale aufweißt.

Hält man die Funktionalität der Schnittstellenkomponenten und Speicher so klein wie möglich, so vereinfacht dies evtl. vorzunehmende Anpassungen an das Gesamtsystem einer digitalen Bibliothek.

Architektur *Funktionalität*	**Two-Tier**	**Three-Tier mit Trader**	**Three-Tier mit Broker**
Serialisierung von Aufträgen	Schnittstelle Speicher	Speicher	Broker
Überwachen der Verbindung zum Speicher	Schnittstelle	Schnittstelle	Broker
Verteilung der Aufträge	Schnittstelle	Trader	Broker

Tabelle 3.2: Aufteilung der Funktionalitäten

Die Aufteilung der Funktionalitäten ist in Tabelle 3.2 dargestellt. Die geringste Funktionalität fällt diesen Komponenten bei der Three-Tier-Architektur mit Broker zu.

Von den untersuchten Architekturen ist also die Three-Tier Architektur-mit Broker am besten zur Realisierung eines verteilten Dokumentenspeichersystems geeignet, daß den genannten Anforderungen genügt.

Das für das Speichersystem zugrundeliegende Konzept sieht vor, daß die Dokumente in Form von Metadokumenten an das Speichersystem übergeben werden. Die Metadokumente selbst referenzieren die Operationen und beinhalten oder referenzieren den Inhalt des Dokuments. Da Metadokumente nur beim Vorgang des Speicherns durch den Broker verarbeitet werden müssen und nicht etwa bei der Verteilung von anderen Aufträgen, ist ein Leistungsengpass nur zu erwarten, wenn zu viele Speicheraufträge zur selben Zeit an das Speichersystem gestellt werden. In der Regel wird das Speichersystem aber in Relation zum Gesamtauftragsvolumen nur verhältnismäßig wenige Speicheraufträge durchzuführen haben, die Anzahl von Abfragen und Suchaufträgen wird um ein vielfaches höher liegen. Aus diesem Grund ist nicht zu erwarten, daß der Broker einen Engpaß bildet, da die durch ihn durchschnittlich vermittelten Datenmengen im Verhältnis zum Inhalt eines Dokuments eher gering sind.

Der Nachteil dieser Architektur ist, setzt man nur einen Broker ein, ist das System bei Ausfall des Brokers nicht mehr funktionsfähig. Setzt man mehrere Broker ein, so müssen Mechanismen in Schnittstellenkomponenten und Speicher implementiert werden, die dafür Sorge tragen, daß automatisch ein anderer Broker kontaktiert wird. Auf Seiten der Schnittstellenkomponente läßt sich dies einfach dadurch erzielen, daß jede Schnittstellenkomponente mehrere Broker kennt und bei Ausfall des primären Brokers automatisch einen anderen Broker anspricht, sowie für die laufenden Aufträge eine Fehlermeldung an den Auftraggeber schickt, damit dieser den Auftrag erneut stellt.

Die Speicher können sich bei Ausfall des primären Brokers nicht mehr bei diesem abmelden und können nicht feststellen, daß der Broker nicht mehr verfügbar ist. Hier ist eine Art „Puls“ von Vorteil, d.h. der Speicher schickt in bestimmten Zeitintervallen eine kurze Nachricht an den Broker und wartet auf eine Antwort. Antwortet der Broker nicht innerhalb einer bestimmten Zeitspanne, so versucht sich der Speicher bei einem anderen Broker anzumelden. Auch hier muß der Speicher mehrere Broker kennen.

Alternativ kann ein Protokoll definiert werden, das es den Brokern ermöglicht untereinander zu kommunizieren. Werden die Speicherlisten ständig ausgetauscht und ein „Puls“ unter den Brokern realisiert, dann ist bei Ausfall eines Brokers die Übernahme der Speicher- und Schnittstellenkomponenten durch einen anderen Broker möglich. Allerdings bedarf es hierzu eines nicht geringen Realisierungsaufwands.

3.3 Eigenschaften des Dokumentenspeichersystems

In dieser Arbeit wird ein Konzept vorgestellt, daß die Three-Tier-Architektur mit Broker verwendet. Das Konzept sieht vor, daß mehrere Schnittstellenkomponenten und mehrere Speicherkomponenten, sowie ein Broker eingesetzt werden.

Bei Ausfall des Brokers erkennen dies die Schnittstellenkomponenten und senden entsprechende Fehlermeldungen an die auftraggebenden Instanzen der digitalen Bibliothek, die den Auftrag gegebenenfalls neu einreichen können.

Um das Speichersystem steuern zu können kommt zusätzlich eine Managementkomponente zum Einsatz, die es erlaubt das System über eine entsprechende Benutzeroberfläche zu konfigurieren.

Managementkomponenten dienen dazu die Ressourcen des Speichersystems (die Speicher) zu beobachten und zu steuern. Im wesentlichen findet man hier Komponenten mit folgenden Funktionalitäten:

- Monitorkomponenten (Watch)
- Administrationskomponenten (Maintenance)

Die Monitorkomponenten kommunizieren über definierte Schnittstellen mit den Administrations- und Vermittlerobjekten und stellen dem Benutzer Informationen über den Zustand des Systems (z.B. wieviele Speicher im System vorhanden sind) zur Verfügung. Die Administrationskomponenten bieten einem Administrator die Möglichkeit das System zu steuern (z.B. Anhalten eines Speichers). In dieser Arbeit werden Monitor- und Administrationskomponente in einem Prozeß vereint.

Aus den getroffenen Entwurfsentscheidungen ergibt sich der in Abbildung 3.4 dargestellte Aufbau für das Dokumentenspeichersystem.

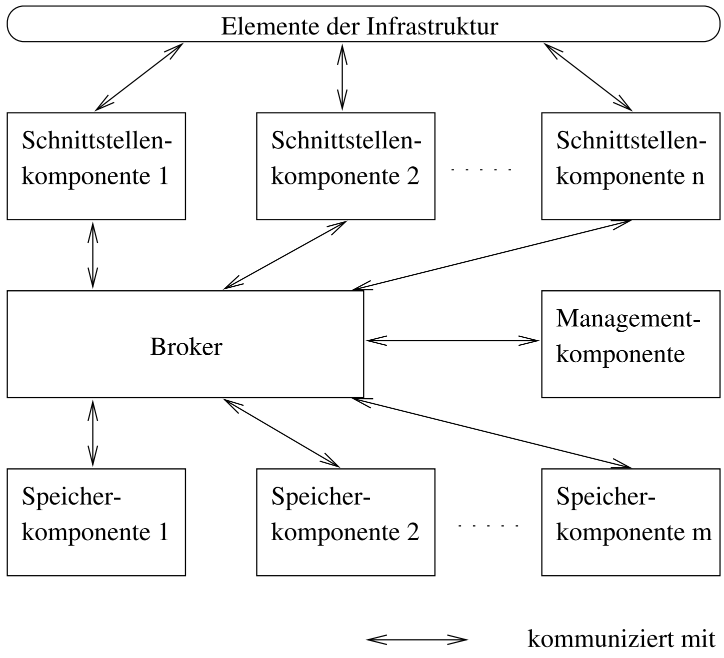


Abbildung 3.4: Aufbau des Speichersystems

Der Broker unterhält Verbindungen zu den Schnittstellenkomponenten, den Speichern und zu einer Managemenkomponente. Über die Mangementkomponente wird das gesamte Dokumentenspeichersystem gesteuert. Das heißt, der Broker vermittelt nicht nur die Aufträge zwischen Schnittstellen und Speichern, sondern auch die Befehle der Managementkomponente. Die Managementkomponente steuert die Zustände der Speicher und kann auf die Verteilung der Aufträge Einfluß nehmen. Desweiteren zeigt sie die aktuelle Konfiguration des Speichersystems in geeigneter Form an.

3.4 Funktionsweise des Speichersystems

Die folgenden Abschnitte beschäftigen sich mit der Funktionsweise der Komponenten des Speichersystems (Schnittstellenkomponente, Broker, Managementkomponente, Speicher) und deren Zusammenspiel. Weiterhin wird ein Überblick über Art und Funktionalität des zu verwendenden Übertragungsprotokolls gegeben.

3.4.1 Das Übertragungsprotokoll des Speichersystems

Innerhalb des Speichersystems wird ein Übertragungsprotokoll verwendet, das dem „Hyper Text Transfer Protocol" (HTTP, [BLFFN95]) ähnlich, ASCII-basiert und zustandslos ist. Die eingesetzten Dienstprimitive können in zwei Klassen unterteilt werden, Speicherkommandos und Steuerkommandos. Die Menge der Speicherkommandos beinhaltet alle Befehle, die von den Speichern ausgeführt werden können. Folgende Grundfunktionalitäten müssen bereitgestellt werden:

- Speichern eines Dokuments
- Aktivieren einer Methode auf einem Dokument
- Informationen über die Programmbibliothek für Dokumentenmethoden des Speichers anfordern
- Rückmeldungen und Fehlermeldungen

In der Klasse der Steuerkommandos sollen mindestens folgende Funktionen unterstützt werden:

- Anmelden eines Speichers, einer Managementkomponente beim Broker
- Abmelden eines Speichers, einer Managementkomponente beim Broker
- Abfrage der aktuellen Konfiguration des Brokers durch die Managementkomponente
- Anhalten eines Speichers durch den Broker (ausgelöst durch Managementkomponente)
- Fortsetzen eines Speichers durch den Broker (ausgelöst durch Managementkomponente)
- Ausgliedern eines Speichers durch den Broker (ausgelöst durch Managementkomponente)
- Zuordnen eines Auftrags zu einem bestimmten Speicher durch den Broker (ausgelöst durch Managementkomponente)
- Fehlermeldungen

3.4.2 Die Schnittstellenkomponente

Schnittstellenkomponenten stellen das Bindeglied zwischen dem Speichersystem und dem Gesamtsystem dar. Die Schnittstellenkomponente dient weiterhin dazu, die eingehenden Nachrichten zu filtern und zu konvertieren, d.h.

die Kommandos des Gesamtsystems werden in Kommandos des Speichersystems übersetzt und evtl. vorzunehmende länderspezifische Konvertierungen vorgenommen (z.B. Umlaute). Es ist zu bemerken, daß nicht die eigentlichen Dokumente gefiltert und damit verändert werden, sondern die eingegangenen Aufträge. In einem Speichersystem kann es mehrere Schnittstellenkomponenten geben. In Abbildung 3.5 werden unterschiedliche Möglichkeiten der Einbindung skizziert.

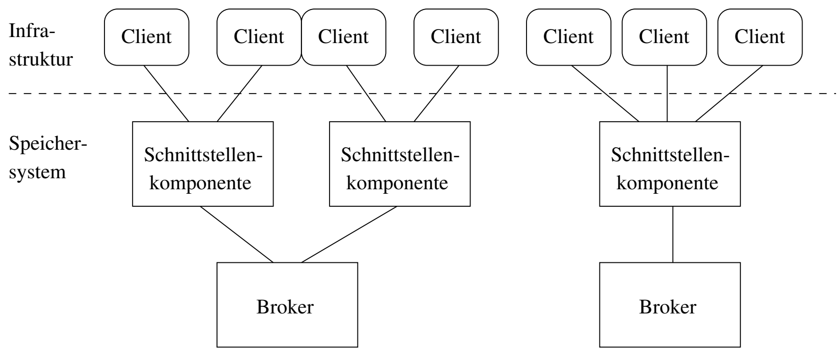


Abbildung 3.5: Einbindung der Schnittstellenkomponente

Die Schnittstellenkomponente wartet unter einer dem Gesamtsystem bekannten Adresse auf Aufträge aus der Infrastruktur (Client). Eine Schnittstellenkomponente kann gleichzeitig mehrere Aufträge aus der Infrastruktur verarbeiten.

Für jeden Auftrag unterhält die Schnittstellenkomponente zwei Netzverbindungen, eine zur auftraggebenden Instanz und eine zum Broker. Das hat den Vorteil, daß man keine Auftragskennung innerhalb des Speichersystems vergeben muß, um ein Ergebnis, einem Auftrag und dem Auftraggeber zuzuordnen. Die Zuordnung erfolgt durch einen verbindungsorientierten Dienst und ist damit eindeutig.

Empfängt die Schnittstellenkomponente einen Auftrag, so baut sie eine Netzverbindung zu einem Broker auf und leitet den Auftrag an diesen weiter. Die Schnittstellenkomponente wartet jetzt bis Resultate oder eine Fehlermeldung vom Broker empfangen werden und sendet diese an den Auftraggeber. Nachdem die Nachricht vom Broker empfangen wurde, sei es eine Fehlermeldung oder ein Ergebnis, wird die Verbindung zum Broker abgebaut.

Sollte die Schnittstellenkomponente keine Verbindung zum Broker aufbauen können, sendet sie eine entsprechende Fehlermeldung an den Auftraggeber.

Damit der Zugang zum Dokumentenspeichersystem selbst bei Ausfall des Brokers bestehen bleibt, werden Schnittstellenkomponenten nicht angehalten.

Aus diesem Grund sind keine Steuerungsfunktionalitäten für diesen Komponententyp im Übertragungsprotokoll vorgesehen. Hiermit wird sichergestellt, daß eine auftraggebende Instanz immer eine Antwort auf einen Auftrag erhält und das ein Verbindungsaufbau zum Dokumentenspeichersystem ständig gewährleistet ist.

Wie und wann die Netzverbindung zur auftraggebenden Instanz des Gesamtsystems auf- bzw. abgebaut wird, hängt von den verwendeten Transportmechanismen der Infrastruktur ab. Dieser Teil der Funktionalität der Schnittstellenkomponente muß an das jeweilige Gesamtsystem angepaßt werden, weil verschiedene Infrastrukturen unterschiedliche Kommunikationsprotokolle zur Benutzung festlegen. Im Normalfall wird es sich hierbei um verbindungsorientierte TCP/IP-Verbindungen ([CK+81]) handeln. Eine Einführung in das TCP/IP-Protokoll bietet [Ste94]. Denkbar sind auch Anbindungen über IPX/SPX ([Nov92]) oder NetBeui/NetBios ([IBM88]) als Netzprotokoll. Darauf aufsetzend werden häufig HTTP-ähnliche Protokolle (Kap. 3.4.1) oder email-Protokolle (SMTP, [Pos82]) als Übertragungsprotokolle verwendet.

3.4.3 Der Broker

Broker haben die Aufgabe, die Aufträge, die sie durch die Schnittstellenkomponenten erhalten, auf die Speicher zur Ausführung zu verteilen und die Speicher zu steuern. Man kann Broker als zentrale Verwaltungs- und Steuerinstanz des Speichersystems verstehen.

Jeder Speicher meldet sich beim Broker an bzw. ab, wenn er gestartet oder beendet wird und teilt dem Broker die Adresse mit, unter der er auf Aufträge wartet. Der Broker unterhält seinerseits eine Liste aller angemeldeten Speicher und deren Zustände. Es handelt sich hierbei nicht um den Prozeßzustand der Speicher bzw. des Brokers, sondern um einen systeminternen Zustand der jeweiligen Komponente. Die Zustände der Komponenten werden vom Broker gespeichert und Aufträge dementsprechend verteilt. Ein Speicher befindet sich beispielsweise im Zustand „passive“, wenn er keinen Auftrag bearbeitet und er dem Broker als „passive“ bekannt ist. Der Speicher selbst hat keine Kenntnis davon, daß er „passiviert“ wurde, er empfängt lediglich keine weiteren Aufträge mehr. Mögliche Zustände aus der Sicht des Brokers sind:

- *not connected*; der Speicher wurde gestartet und hat sich noch nicht beim Broker angemeldet oder hat sich bereits wieder abgemeldet
- *active idle*; der Speicher darf Aufträge verarbeiten, ist aber zur Zeit im Leerlauf

- *active busy*; der Speicher darf Aufträge verarbeiten und arbeitet zur Zeit
- *passive*; der Speicher darf keine Aufträge erhalten, er ist gestoppt

Ein Zustandsdiagramm, das in gleicher Weise für den Broker und die Speicher gilt, ist in Abb. 3.6 dargestellt.

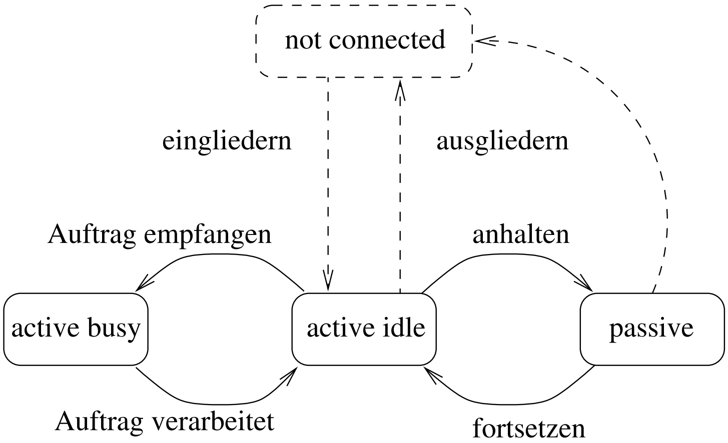


Abbildung 3.6: Zustandsdiagramm des Brokers und der Speicher

Der Zustand „not connected" wird angenommen, wenn die Komponenten noch nicht oder nicht mehr Bestandteil des verteilten Systems sind.

Wenn eine Schnittstellenkomponente eine Verbindung zum Broker aufbaut und einen Auftrag sendet, überprüft der Broker in der Speicherliste welcher Speicher verfügbar ist. Der Broker baut eine Verbindung zum Speicher auf, sendet den Auftrag an den Speicher und vermerkt in der Speicherliste, daß der Speicher jetzt arbeitet (*active busy*) in der Speicherliste.

Der Broker wartet auf Antwort vom Speicher, erhält er eine Antwort, dann wird die Speicherliste wiederum aktualisiert (*active idle*) und die Verbindung zum Speicher abgebaut. Sollte die Verbindung zum Speicher gestört werden oder wird innerhalb einer bestimmten Zeitspanne keine Antwort empfangen, dann sendet der Broker selbständig eine Fehlermeldung an die Schnittstellenkomponente.

Die Auswahl eines geeigneten Speichers stellt ein nicht triviales Problem dar. Die gleichmäßige Verteilung der Aufträge unter Berücksichtigung der unterschiedlichen Performance der einzelnen Speicher sollte empirisch erfolgen. Dazu sammelt der Broker Informationen über Ausführungszeiten und Auftragsmenge der einzelnen Speicher. Anhand dieser statistischen Daten läßt sich jedem Speicher ein Perfomance-Index zuordnen (z.B. durchschnittliche Verarbeitungszeit pro Auftrag). Der Broker sucht dann immer den Speicher zur Abwicklung eines Auftrags aus, der den besten Performanceindex hat und momentan keinen Auftrag verarbeitet. Eine detailierte Beschreibung verschiedener Verfahren zur Leistungsmessung in verteilten Systemen findet sich in [Heu94].

3.4.4 Das Monitor-/Steuerungswerkzeug

Um das System steuern zu können, besitzen die Broker eine Steuerverbindung zu einer Managementkomponente, die als Monitor-/Steuerungswerkzeug bezeichnet werden soll. Mit diesem Werkzeug kann man sich an einen Broker verbinden und die Konfiguration des Brokers und der daran verbundenen Speicher anzeigen lassen. Über das Monitor-/Steuerungswerkzeug kann der Zustand der Speicher, sowie des Brokers verändert werden. Darüber hinaus können Speicher aus dem System ausgegliedert werden und Aufträge manuell auf die Speicher verteilt werden.

Es werden folgende Operationen unterstützt:

- Anhalten, Fortsetzen und Ausgliedern von Speichern
- Abfrage von Zustandsinformationen der Speicher
- Anhalten und Fortsetzen des Brokers
- Manuelle Verteilung der Aufträge auf die Speicher

Die vom Monitor-/Steuerungswerkzeug initiierten Operationen bzgl. der Speicher werden dabei auf die in Abschnitt 3.4.1 beschriebenen Funktionen abgebildet. Die Operationen werden direkt durch den Broker verarbeitet. Das Monitor-/Steuerungswerkzeug verbindet sich über eine ausgezeichnete Netzverbindung an den Broker.

3.4.5 Der Speicher

Die Integration der Speicher in das Speichersystem erfolgt wiederum über einen verbindungsorientierten Dienst. Innerhalb eines Speichersystems können und sollen mehrere Speicher parallel eingesetzt werden, sie bilden einen Lastverbund. Alle Speicher im Lastverbund werden über das gleiche Netz- und Übertragungsprotokoll angesprochen und müssen dieses unterstützen.

Wird ein Speicher neu initiiert, dann schickt er eine Nachricht an den Broker und teilt diesem seine Existenz und die Adresse, unter der er auf Aufträge wartet, mit. Die Adresse des Brokers muß dem Speicher vorab bekannt sein.

Soll der Speicher terminiert werden, so schickt er zuvor ebenfalls eine Nachricht an den Broker, um sich abzumelden und um damit zu verhindern, daß weitere Aufträge an ihn vergeben werden.

Der Broker wendet sich an den Speicher, wenn ein Auftrag von diesem Speicher ausgeführt werden soll. Der Speicher empfängt einen Auftrag vom Broker, ermittelt welche Aktion (Methode) auszuführen ist und führt die Methode innerhalb einer geeigneten Ablaufumgebung für mobilen Code aus. Das Ergebnis des Methodenaufrufs wird an den Broker zurückgegeben. Ein solcher Ablauf ist in Abbildung 3.7 skizziert. Ein Speicher kann immer nur einen Auftrag gleichzeitig bearbeiten.

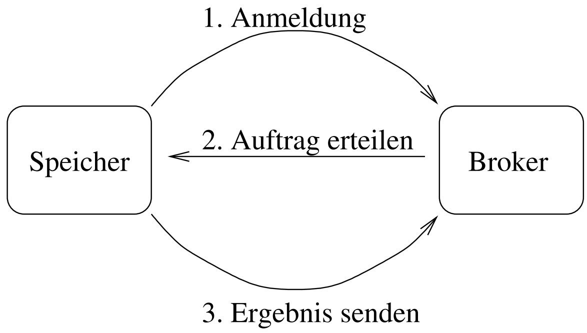


Abbildung 3.7: Auftragsausführung aus Sicht des Speichers

Bemerkenswert ist, daß alle in das Speichersystem eingebundenen Speicher auf demselben Datenbestand operieren. Trotzdem wird durch die Verteilung auf mehrere Speicher ein großer Leistungsgewinn erreicht, da die Speicher auf unterschiedlichen Teilmengen der Daten operieren und die eigentliche Verarbeitung der Daten innerhalb der Ablaufumgebungen der Speicher erfolgt (z.B. das Entpacken und Packen von Bilddateien). Dabei werden die Speicher im allgemeinen auf unterschiedliche Rechnerknoten verteilt.

Kapitel 4

Der aktive Speicher

Nachdem in Kapitel 3 die verschiedenen Komponenten entworfen und ihre Rollen im Dokumentenspeichersystem definiert wurden, beschäftigt sich dieses Kapitel mit den aktiven Speichern, welche die eigentliche Auftragsverarbeitung innerhalb des Speichersystems übernehmen.

Unter aktiven Speichern werden in dieser Arbeit Speicher verstanden, die es ermöglichen Dokumente zu übertragen, zu speichern und geeignete Mechanismen bereitstellen, um auf den Dokumenten definierte Operationen auszuführen. Weiterhin erlauben aktive Speicher die Übertragung und Speicherung der auf den Dokumenten definierten Operationen, die auch als Dokumentenmethoden bezeichnet werden.

4.1 Anforderungen

Die Anforderungen an den aktiven Speicher werden im folgenden aus den Zielen (Kapitel 1) dieser Arbeit abgeleitet.

Der aktive Speicher verwaltet multimediale Dokumente, die durch Metadokumente (s. Kapitel 2.4.2) beschrieben werden.

Zur Verwaltung multimedialer Dokumente (z.B. SGML-Dokumente mit eingebunden Programmen, Videosequenzen, Tonstücken, Bildern, Sprachaufnahmen) bedarf es medienspezifischer Operationen, die jedem einzelnen Dokument individuell zugeordnet werden können. Diese Operationen werden in Dokumentenmethoden realisiert und in Form von mobilem Code bereitgestellt, um das Dokument an mehreren Lokationen innerhalb der Infrastruktur bearbeiten zu können.

Dabei stellt der Speicher selbst keine kompletten Methoden zu Bearbeitung der Dokumente zur Verfügung (z.B. zum Speichern oder Durchsuchen eines Dokuments), sondern atomarere Operationen aus denen komplette Methoden zusammengestellt werden können. Diese atomaren Operationen werden in der Programmbibliothek für Dokumentenmethoden bereitgestellt.

Der Speicher soll es ermöglichen Dokumente sowohl in Form eines Bitstroms (Datei) als auch in einer Datenbank abzulegen. Aus diesem Grund sollen atomare Methoden zum Zugriff auf Bitstromspeicher, Datenbanken und evtl. weitere Speicherformen (record based, volume based, usw.) bereitgesstellt werden. Damit können sowohl strukturorientierte als auch strukturunabhängige Operationen auf den Dokumenten ausgeführt werden.

Das Speichern als Bitstrom hat im Gegensatz zur Speicherung in einer Datenbank den Vorteil, daß beliebige Dokumententypen abgelegt werden können, ohne neue Funktionalitäten bereitstellen zu müssen. Die Vorteile der in einer Datenbank gespeicherten Dokumente liegen darin, daß die Kenntnis über den strukturellen Aufbau der Dokumente benutzt werden kann, um komfortabler und effizienter darauf zu operieren.

Um sicherzustellen, daß der aktive Speicher an vorhandene und zukünftige Gesamtsysteme digitaler Bibliotheken angebunden werden kann, soll eine weitgehende Unabhängigkeit von den verwendeten Basisprodukten erreicht werden. Unter Basisprodukten versteht man beispielsweise Datenbankmanagementsysteme (DBMS), Bitstromspeicher oder die in Bitstromspeichern zur Suche verwendeten Volltextindizierer. Durch einen stark modularen Aufbau des Speichers kann man erreichen, daß bestimmte Teile des Speichers, wie zum Beispiel ein Datenbankmanagementsystem, ausgetauscht werden können. Das bedeutet, daß alle Einstellungen, die innerhalb der Speicher benötigt werden (z.B. Benennung der Datenstrukturen, Pfadnamen, etc.) um Dokumentenmethoden auszuführen bzw. um die Basisprodukte anzusprechen, dem Rest des Gesamtsystems bzw. der Infrastruktur verborgen bleiben.

Zusammenfassung der Anforderungen:

- Verarbeitung von Metadokumenten
- Bereitstellung atomarer Operationen zum Zugriff auf die Ressourcen (DBMS, Bitstromspeicher, Volltextindizerer) des Speichers
- Modularer Aufbau des Speichers zwecks Anbindung von Basisprodukten (DBMS, Volltextindizierer)

4.2 Eigenschaften des aktiven Speichers

In den nächsten Abschnitten wird die Architektur eines Speichers entworfen, der die genannten Anforderungen erfüllt.

Der Speicher stellt atomare Operationen zur Verarbeitung von SGML-Dokumenten bereit. Durch generische Schnittstellen zu den Basisprodukten (Bitstromspeicher, DBMS, Indizierer) ist es jedoch möglich Dokumente jeden anderen Typs ebenfalls zu verarbeiten. In Abhängigkeit vom Dokumententyp muß hierbei eventuell mit Effizienzeinbußen gerechnet werden.

Die Definition der atomaren Operationen ist überaus kritisch, weil sichergestellt werden muß, daß man aus einer Menge atomarer Operationen eine Menge von Methoden erhält, die so aufeinander abgestimmt sind, daß die gewünschte Funktionalität erreicht wird. Gleichzeitig sollen die atomaren Operationen so wenig wie möglich restriktiv sein und dem Autor einer Methode jeden Spielraum belassen.

Die Zuordnung der Methoden zu einem Dokument erfolgt außerhalb des Speichers durch eine Instanz in der Infrastruktur (i. allg. durch den Autor des Dokuments selbst) und manifestiert sich in den Metadokumenten.

Die daraus resultierenden Vorteile sind offensichtlich. Die veröffentlichende Instanz hat volle Kontrolle über die Operationen, die auf dem Dokument ausgeführt werden können. Sie kann festlegen auf welche Art ein Dokument gespeichert, durchsucht oder präsentiert wird. Desweiteren können die Methoden einfach an neue Bedürfnisse angepaßt werden.

Zur Realisierung von Dokumentenmethoden werden verschiedene Pakete, die eine Grundfunktionalität des Speichers bereitstellen, hergeleitet und in der Programmbibliothek für Dokumentemethoden zusammengefaßt. Dabei wird eine für SGML-Dokumente verwendbare Menge von Operationen bzw. Methoden definiert. Weiterhin werden entsprechende Ablaufumgebungen zur Ausführung der Dokumentenmethoden zur Verfügung gestellt.

4.3 Grundlegender Aufbau und Funktionsweise

In diesem Abschnitt wird die Architektur des Speichers, sowie die Vorgehensweise bei der Auftragsverarbeitung dargestellt. Die einzelnen Module des Speichers lassen sich direkt aus den genannten Eigenschaften ableiten. Der Speicher setzt sich aus einer Schnittstelle zur Kommunikation mit dem Broker (Kapitel 3.4.3) bzw. der Infrastruktur, einer oder mehrerer Ablaufumgebungen und der Programmbibliothek zum Zugriff auf die Speicherressourcen zusammen. Die Programmbibliothek beinhaltet nach Funktionalitäten gebündelte Pakete, die ihrerseits auf die Schnittstellen zu den Basisprodukten aufsetzen.

In Kapitel 2.5 wurde auf die Komplexität multimedialer Dokumente und den Anspruch mehrere unterschiedliche Zugriffsmöglichkeiten bereitzustellen hingewiesen. Aus diesem Grund werden ein Datenbankmanagementsystem und ein Bitstromspeicher als Basisprodukte im Speicher eingesetzt. Um den Bitstromspeicher effizient durchsuchen zu können, wird zusätzlich ein Volltextindizierer eingebunden.

Bei der Auswahl der Basisprodukte ist zu beachten, daß mehrere Speicher auf dem gleichen Datenbestand operieren (Kapitel 3.4.5). Eine Datenbank muß in der Lage sein gleichzeitig mehrere Datenbank-Clients zu bedienen. Analog müssen Bitstromspeicher und die darauf operierenden Indizierer derart angesprochen werden können, daß parallele Zugriffe möglich sind. Beispielsweise kann ein Bitstromspeicher durch ein Dateisystem realisiert werden, das über das „Network Filesystem"-Protokoll ([Sun95]) oder das „Session Message Block"-Protokoll ([Mic89]) an mehrere Rechnerknoten verbunden ist.

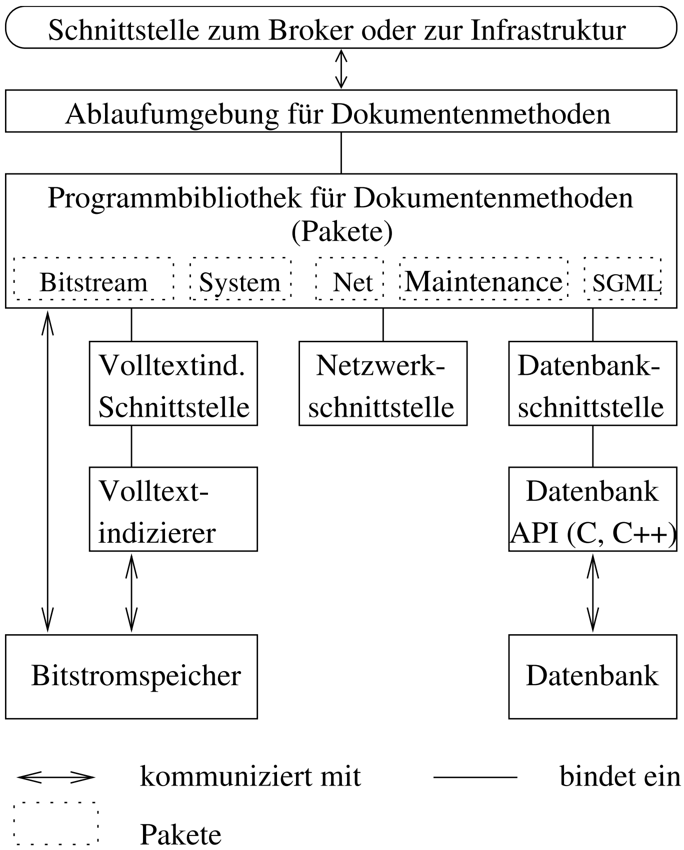


Abbildung 4.1: Aufbau des Speichers mit einer Ablaufumgebung

Der Aufbau des Speichers mit nur einer Ablaufumgebung ist in Abbildung 4.1 illustriert. Der Speicher empfängt über die Schnittstelle (zum Broker oder zur Infrastruktur) einen Auftrag, welcher von der Schnittstelle konvertiert bzw. gefiltert wird. Der Auftrag wird dann an die Ablaufumgebung für Dokumentenmethoden übergeben. In der Ablaufumgebung wird der Auftrag ausgewertet und das darin enthaltene Kommando, unter Zuhilfenahme der Programmbibliothek für Dokumentenmethoden (Pakete) und der darunter liegenden Schnittstellen zu den Basisprodukten ausgeführt. Eine entsprechende Rückmeldung über Erfolg oder Mißerfolg des ausgeführten Kommandos wird an die Schnittstelle übergeben. Die Schnittstelle sendet die Rückmeldung zurück an die Instanz, welche den Auftrag übermittelt hat.

Durch den Einsatz einer Schnittstelle im Speicher selbst, ist es möglich den in dieser Arbeit entwickelten Speicher auch unabhängig vom Dokumentenspeichersystem einzusetzen oder ihn innerhalb anderer Speichersysteme zu verwenden.

Der Speicher verarbeitet mindestens folgende Kommandos:

- Speichern, zum Speichern eines Dokuments
- Aktivieren, zum Ausführen einer Dokumentenmethode auf einem Dokument (wie in Kap. 2.4.2 beschrieben)
- Informieren, liefert Informationen über die Programmbibliothek für Dokumentenmethoden

Ein Speichervorgang erfordert gleichzeitig die Übermittlung des Metadokuments für das zu speichernde Dokument. Wurde das Metadokument übermittelt, so wird das enthaltene Dokument temporär zwischengespeichert und den Dokumentenmethoden der Zugriff erlaubt. Das Dokument wird dann durch die Aktivierung (s.u.) einer entsprechenden Dokumentenmethode im Speicher abgelegt. Jeder Speichervorgang erzeugt eine eindeutige Dokumentenkennung (Dokumenten-ID), die dem Dokument automatisch zugeordnet wird und der auftraggebenden Instanz nach erfolgreicher Beendigung des Vorgangs übermittelt wird.

Diese Kennung ist innerhalb des Dokumentenspeichersystems eindeutig. Innerhalb der Infrastruktur besitzt das Speichersystem wiederum eine eindeutige Kennung. Konkateniert man die beiden Kennungen durch einen Punkt (.), so erhält man eine systemweit eindeutige Kennung. Der Aufbau der Kennung und ein Beispiel sind im folgenden wiedergegeben.

```
Kennungsschema : <Speichersystemkennung>.<Dokumenten-ID>

Beispiel       : Aktiver_Speicher_1.700-s-853-de
```

Aktivierungen veranlassen den Speicher dazu, eine bestimmte Dokumentenmethode auf einem bestimmten Dokument auszuführen. Dabei werden die Dokumenten-ID und der Methodenname, sowie evtl. Methodenparameter zusammen mit dem Aktivierungskommando an den Speicher übertragen. Der Speicher führt die Methode auf dem Dokument aus und sendet die Resultate (z.B. Suchergebnisse) oder eine Fehlermeldung an den Auftraggeber.

Um sich über die von der Programmbibliothek bereitgestellten Pakete zu informieren, kann ein „Informieren"-Kommando an den Speicher übermittelt werden. Der Speicher durchsucht alle Pakete der Programmbibliothek und sendet die so ermittelten Informationen über die verfügbaren Methoden zurück.

4.4 Schnittstelle des Speichers

Im folgenden wird die Schnittstelle des aktiven Speichers zum Speichersystem bzw. zum Gesamtsystem beschrieben. Die Funktionalität der Schnittstelle beschränkt sich darauf, Nachrichten zu konvertieren bzw. zu filtern und die Routinen zur Kommunikation mit anderen Instanzen innerhalb des Speichersystems oder der Infrastruktur bereitzustellen (ähnlich der Schnittstellenkomponente, s. Kap. 3.4.2). Diese Routinen werden von der Ablaufumgebung des aktiven Speichers benutzt, um Nachrichten zu empfangen oder zu versenden.

Die Schnittstelle des Speichers verarbeitet eingehende Aufträge, die einem einfachen zustandslosen, HTTP-ähnlichen und ASCII-basiertem Protokoll entsprechen, wie es in Kapitel 3.4.1 beschrieben ist. Weiterhin übernimmt diese Schnittstelle evtl. vorzunehmende länderspezifische Konvertierungen bzw. Filterungen (z.B. Umwandlung von Umlauten) der eingehenden Aufträge. Zusätzlich wird eine Umsetzung (mapping) von Kommandos durchgeführt, z.B. wenn die eingehenden Aufträge andere Kommandos verwenden als der Speicher intern. Die Rückmeldungen über eine erfolgreiche Ausführung eines Auftrags oder entsprechende Fehlermeldungen werden ebenfalls über die Schnittstelle zurückgegeben. Die eingegangenen Aufträge werden dann an die Ablaufumgebung übergeben. Im allgemeinen wird die hier entworfene Schnittstelle durch eine Menge von Methoden oder Funktionen realisiert sein. Das Protokoll der Schnittstelle soll analog zu den beschriebenen Kommandos mindestens folgende eingehende Dienstprimitive unterstützen:

- Speichern
- Aktivieren
- Informieren

Die Menge der Kommandos kann um neue Befehle erweitert werden, damit kann der Speicher an neue Gegebenheiten angepaßt werden.

4.5 Die Ablaufumgebungen für Dokumentenmethoden

Dieser Abschnitt stellt die Funktionsweise der Ablaufumgebungen für Dokumentenmethoden dar und definiert, wie diese in den aktiven Speicher eingebunden werden.

Bei der Ausführung einer Dokumentenmethode werden Dokumenten-ID und Dokumentenmethode, sowie evtl. Parameter der Methode an eine Ablaufumgebung übergeben. Die Ablaufumgebung sorgt für die Einbettung der Dokumentenmethode in das lokale Umfeld des Speichers, beispielsweise durch Initialisierung bestimmter globaler Variablen. In dieser Umgebung wird dann die Dokumentenmethode ausgeführt. Entsprechende Mechanismen, sowohl für die Parameterübergabe, als auch für die Rückgabe von Ergebnissen oder Fehlermeldungen, werden ebenfalls in der Ablaufumgebung vorgesehen. Als Beispiel für die Werteübergabe soll hier das Umleiten der Standardbildschirmausgabe und der Mechanismus zur Paramterübergabe „call by value“ angeführt werden. Eine Übergabe von Werten durch „call by reference“ ist bei entfernt ausgeführten Methoden nicht möglich.

Der Speicher erkennt, wie bereits unter Kapitel 2.4.2 beschrieben, anhand des jeder Dokumentenmethode zugeordneten MIME-Typs, welcher Ablaufumgebung die auszuführende Dokumentenmethode übergeben werden muß. Wird ein bestimmter MIME-Type und damit eine bestimmte Programmiersprache nicht unterstützt, so wird eine Fehlermeldung generiert. Es werden verschiedene Ablaufumgebungen unterstützt. Deshalb ist es notwendig, daß die Ablaufumgebungen streng modular realisiert werden. Erleichtert wird dadurch die Einbindung neuer Ablaufumgebungen. Als geeignete Programmiersprachen werden, wie unter Kapitel 2.3 dargestellt, Sprachen wie beispielsweise Python, Java oder Tcl betrachtet. Interessant ist in diesem Zusammenhang, mehrere Ablaufumgebungen für die gleiche Programmiersprache bereitzustellen, die sich durch die Zugriffsmöglichkeit auf Systemressourcen unterscheiden. Auf diese Weise können verschiedenen Instanzen unterschiedliche Zugriffsrechte zugeteilt werden. Eine Instanz zur Speicheradministration (z.B. Warten der Datenbank) benötigt mehr Privilegien als eine Instanz, die im Speicher suchen möchte.

Vorstellbar ist hier beispielsweise eine Unterscheidung durch die Erlaubnis, Kindprozesse zu erzeugen oder Sockets zu benutzen. Voraussetzung für die

Unterstützung dieses Mechanismus ist die Möglichkeit den Interpreter oder Compiler der verwendeten Programmiersprache dahingehend konfigurieren zu können.

4.6 Repräsentation der Dokumente in der Datenbank

Dieser Teil der Arbeit entwickelt eine Datenstruktur, die darauf abzielt, sowohl einfachen Zugriff auf die Struktur, als auch auf die eigentlichen Inhalte der SGML-Dokumente zu erlauben. Zusätzlich wird eine Möglichkeit gegeben, auf die SGML-Dokumente als Ganzes zuzugreifen. Aus der an den Speicher gestellten Anforderung gängige Datenbankmanagementsysteme zu unterstützen, leitet sich weiterhin der Anspruch ab, eine Datenstruktur zu entwickeln, die einfach aufgebaut, erweiterbar und auf andere DBMS übertragbar ist.

Wie bereits in Abschnitt 4.3 beschrieben, werden die SGML-Dokumente als Datei (Bitstrom) zwischengespeichert, können also sequentiell verarbeitet werden.

Um getrennt auf Tags und Taginhalte zuzugreifen, liegt es nahe Tags und Daten in zwei Datenbankklassen separat abzulegen. Ein solches Konzept wurde bereits unter Kapitel 2.2 vorgestellt. Der einfache Zugriff auf das gesamte SGML-Dokument ist hier nicht gegeben. Beide Klassen müssen getrennt ausgewertet und dann gemischt werden.

Das hier vorgestellte Konzept nutzt aus diesem Grund den Mechanismus der Vererbung, der in objekt-orientierten Datenbanken zur Verfügung steht. Dazu wird eine im objekt-orientierten Sinne abstrakte Basisklasse eingeführt und von dieser getrennte Klassen für Tags und Taginhalte abgeleitet.

Basisklassen erlauben es, daß Abfragen über die Basisklasse und alle vererbten Datenbankklassen durchgeführt werden. Gleichzeitig definiert die Basisklasse eine Datenstruktur, auf der alle vererbten Klassen aufbauen. Zu dieser Datenstruktur gehören Informationen, wie die eindeutige Dokumenten-ID und eine fortlaufende Nummer, die analog zur Reihenfolge der Elemente im SGML-Dokument während des Speicherns vergeben wird.

Fragt man über die Basisklasse alle Datensätze mit der gleichen Dokumenten-ID ab und sortiert diese nach der fortlaufenden Nummer, so erhält man das originale SGML-Dokument als Ergebnis. So ist die für digitale Bibliotheken wichtige authentische Wiederherstellung der gespeicherten Informationen auf einfachste Weise gewährleistet.

Auf die Bedeutung des Strukturbaums bei der Verarbeitung von SGML-Dokumenten wurde bereits in Kapitel 2.2 hingewiesen und die vereinfachte Speicherung des Strukturbaums in Datenbanken beschrieben. Die Tag-Klasse sieht in dieser Datenstruktur zwei Felder vor, die die fortlaufende Nummer des Bruder-Tags und des Sohn-Tags aufnehmen. Da die Elemente der Dokumente der Reihe nach gespeichert werden, kann der Strukturbaum bzw. die Bruder- und Sohn-Tags während des Speicherns berechnet werden, erfordert also keine separaten Berechnungen nach dem Speichervorgang.

Der Zugriff auf den Strukturbaum erfolgt direkt über die Tag-Klasse, in der unter Angabe der Dokumenten-ID und Auswahl des Elements mit der fortlaufenden Nummer „Eins“ die Wurzel des Strukturbaums abgefragt werden kann. Unter Verwendung von Depth-First-Search ([Knu73]) kann der Baum durchlaufen werden.

Binäre Informationen wie Bild- oder Tondaten können in SGML-Dokumente eingebunden werden. Viele Datenbanksysteme unterscheiden jedoch prinzipiell zwischen Text (ASCII) und Binärdaten. Deshalb wurden innerhalb des Datenmodells ebenfalls zwei Klassen eingesetzt, die zum einen reine Textdaten zum anderen Binärdaten aufnehmen.

Das beschriebene Datenmodell ist in Abbildung 4.2 dargestellt.

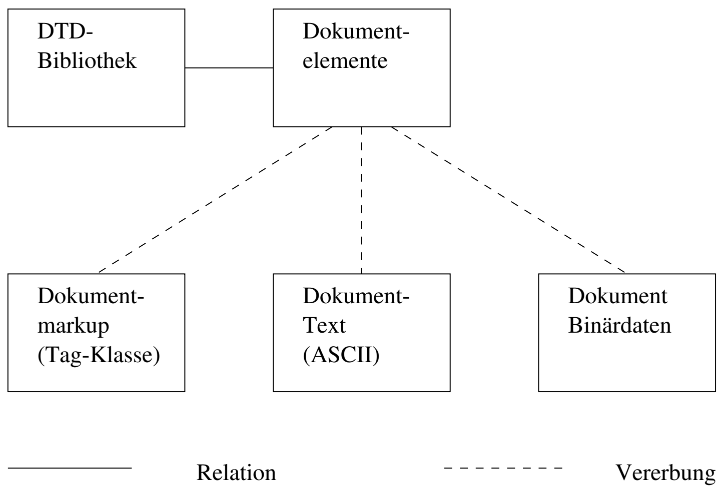


Abbildung 4.2: Datenmodell zur Speicherung von SGML-Dokumenten

In den einzelnen Klassen werden folgende Informationen abgelegt:

- DTD-Bibliothek; hier werden die Document Type Definitions gespeichert Jede DTD wird nur einmal abgelegt und über einen Schlüssel mit den jeweiligen Dokumenten verbunden

- Dokumentelemente; dies ist die abstrakte Basisklasse für die verschiedenen Dokumentelementklassen
  - Dokumentmarkup; in dieser Klasse werden alle Tags gespeichert
  - Dokument Text (ASCII); hier werden alle Textelemente der SGML-Dokumente abgelegt
  - Dokument Binärdaten; diese Klasse beinhaltet alle Binärdaten, wie Bilder, Programme oder Tondaten, der SGML-Dokumente

Zu bemerken ist, daß eine Klasse jeweils die Tag-, Text- oder Binärinformationen aller gespeicherten SGML-Dokumente beinhaltet und nicht für jedes einzelne Dokument eine solche Datenbanklassenhierarchie instantiiert wird. Aus diesem Grund ist es nicht sinnvoll Operationen wie Volltextsuchen über die Datenbank durchzuführen, obwohl dies generell möglich wäre. Die Leistungsgrenzen der Datenbank wären schnell erreicht und eine Suche würde immense Ressourcen beanspruchen.

Die Idee für das hier entwickelte Datenmodell ist an das Projekt VODAK angelehnt. Das SGML-Datenbanksystem VODAK, das unter Kapitel 2.2.2 bereits beschrieben wurde, instantiiert für jedes SGML-Dokument eine Objekthierarchie, die der Struktur des Dokuments entspricht. Das vorgestellte Datenmodell des Speichers speichert ebenfalls die Objekthierarchie, jedoch wird die Hierarchie durch einfaches Verknüpfen (fortlaufende Nummer, Bruder, Sohn) und nicht durch Vererbung und Aggregation von Objekten realisiert.

VODAK bietet eine wesentlich mächtigere und komfortablere Verarbeitung von SGML-Dokumenten. Aufgrund der proprietären Schnittstellen und der Komplexität des VODAK-Systems, konnte das System jedoch keine Verwendung finden.

Die Eigenschaften des in diesem Abschnitt entwickelten Datenmodells im Überblick:

- Das Datenbankmodell ist auf andere gängige DBMS übertragbar.
- Das Datenbankmodell bietet einfachen Zugriff auf den Strukturbaum der SGML-Dokumente, sowie auf die Dokumentinhalte.
- Der direkte Zugriff auf ganze Dokumente wird durch Verwendung des Vererbungsmechanismus objekt-orientierter DBMS gewährleistet.
- Das Speichern der Dokumente muß sequentiell erfolgen.

Dokumente sollten prinzipiell einmal strukturiert in der Datenbank und einmal unstrukturiert als Bitstrom (Datei) im Speicher abgelegt werden. So können effiziente Volltextsuchen durch Einsatz entsprechender Indizierer und strukturierte Suchen unter Verwendung der Datenbank durchgeführt werden. Allgemein können durch diese Vorgehensweise alle sequentiell auf den Dokumenten auszuführenden Operationen effizienter auf den als Bitstrom gespeicherten Informationen durchgeführt werden. Alle Operationen, die gezielt auf bestimmte Teile oder der gesamten Struktur des Dokuments aufsetzen, können effizienter auf den in der Datenbank gespeicherten Daten ausgeführt werden.

Diese Effizienz und Flexibilität bzgl. des Zugriffs auf die SGML-Dokumente kostet jedoch den doppelten Speicherplatz für jedes Dokument, auf den zur Ablage bestimmten Massenspeichern.

4.7 Die Programmbibliothek für Dokumentenmethoden

Dieser Abschnitt beschäftigt sich mit dem Entwurf der Programmbibliothek für Dokumentenmethoden, die speziell zur Bearbeitung von SGML-Dokumenten ausgelegt ist. Zunächst wird ein Konzept zur Organisation der bereitgestellten Funktionalitäten in Paketen entwickelt. Es schließt sich die Aufteilung der Funktionalitäten auf die Pakete und die darin enthaltenen Klassen an. Schließlich werden die Methoden der Klassen hergeleitet und an Beispielen erläutert. In Kapitel 5 werden die Pakete realisiert, eine vollständige Liste der Methoden findet sich in Anhang C.

Die Programmbibliothek für Dokumentenmethoden bildet das Herz des Speichers. Hier wird den Dokumentenmethoden die gesamte Funktionalität des Speichers zur Verfügung gestellt. Die Programmbibliothek verbirgt alle Details der eingesetzten Basisprodukte. Außerdem stellt sie auch Methoden zur Verfügung, die dazu dienen den Speicher zu warten und zu konfigurieren, so daß zur Pflege kein spezielles Werkzeug erforderlich ist. Es werden einfach die entsprechenden Methoden aktiviert.

Die Entwicklung der Methoden in den von den Paketen bereitgestellten Klassen erfolgte ausgehend von der in Kapitel 4.6 beschriebenen Vorgehensweise beim Speichern von Dokumenten, die ihrerseits von den Anforderungen an den Speicher abgeleitet wurde.

Die Methoden sind hierarchisch organisiert. Die Methoden sind zunächst in Klassen zusammengefaßt und die Klassen ihrerseits in Pakete (packages). So kann man sich z.B. ein Paket mit Namen SGML vorstellen, das die Klassen *sgml_store* und *sgml_search* bereitstellt, die erste beinhaltet Operationen zum Speichern, die zweite Operationen zum Durchsuchen eines Dokuments. Die Klasse *sgml_search* könnte ihrerseits Operationen, wie z.B. *search_tag* zum Auffinden eines bestimmten Tags oder *search_text* zum Auffinden eines bestimmten Textstrings in einem gespeicherten SGML-Dokument, (s. Kap. 2.1) beinhalten. Abbildung 4.3 soll das Paket-Klassen-Konzept verdeutlichen.

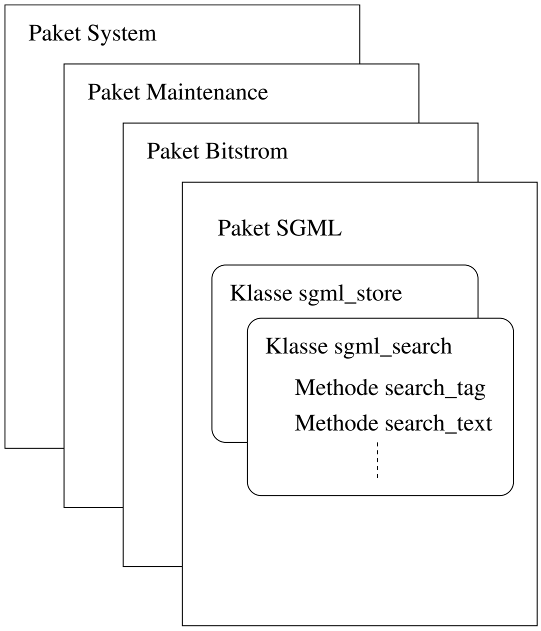


Abbildung 4.3: Das Paket-Klassen-Konzept

Durch das Gruppieren der Klassen (packaging) und damit der beinhalteten Methoden ist es einfach, die Programmbibliothek um neue Funktionalitäten zu erweitern und an gesamtsystemspezifische Gegebenheiten anzupassen. Diese Klassifizierung der zur Verfügung stehenden Operationen ermöglicht weiterhin einen übersichtlichen und organisierten Zugriff auf die einzelnen Operationen und vermeidet weitgehend Namenskonflikte der Methoden. Weiterhin ist es möglich, das gleiche Paket in unterschiedlichen Entwicklungsstufen (Versionen bzw. Releases) bereitzustellen und somit Abwärtskompatibilität zu gewährleisten.

So kann man sich folgende einfache Klassifizierung der Methoden vorstellen:

1. Systemmethoden; in diesem Paket findet man alle Klassen und Methoden, die der Speicher benötigt, um seine internen Daten zu verwalten. Dieses Paket wird nicht exportiert, d.h. die darin befindlichen Funktionalitäten stehen nur Administratoren und der Ablaufumgebung zur Verfügung.

2. Maintenancemethoden; in diesem Paket findet man alle Klassen und Methoden, die zur Wartung benötigt werden (z.B. Reorganisation der Datenbank oder Reindizierung der Volltextindizes). Dieses Paket darf nur

von autorisierten Instanzen benutzt werden, die darin befindlichen Klassen bzw. deren Methoden stehen nur Administratoren des Speichersystems zur Verfügung.

3. Bitstrommethoden; in diesem Paket findet man alle Klassen und Methoden, die zur Verfügung gestellt werden, um Bitstromdaten zu verarbeiten bzw. zu durchsuchen.

4. SGML-Methoden; in diesem Paket findet man alle Klassen und Methoden, die zur Verarbeitung bzw. zum Durchsuchen von SGML-Dokumenten zur Verfügung stehen.

Natürlich kann es Klassen geben, die mehreren Paketen zugeordnet sind. Beispielsweise könnte eine Methode, die den Dokumententyp eines SGML-Dokuments ausliest, im Paket-System und im Paket-SGML zu finden sein. Eine Dokumentenmethode, die zur Bearbeitung mehrerer Dokumenttypen dient, würde eine solche Methode benötigen, wie auch der Speicher, der grundsätzlich ermittelt um welchen Dokumenttyp es sich bei einem hinterlegten Dokument handelt, damit die Document Type Definition (DTD) des Dokuments immer nur einmal vorgehalten werden muß.

4.7.1 Definition der Basispakete des Speichers

Im letzten Abschnitt wurde eine einfache Klassifizierung der Klassen in vier Pakete vorgestellt. Die Klassen des System-Pakets und des Maintenance-Pakets sind implementierungsabhängig. Die Funktionalität des Maintenance-Pakets hängt immer von den verwendeten Basisprodukten (DBMS, Indizierer) ab, die Systemklassen von der Implementierung der Ablaufumgebungen.

Dem Bitstrom-Paket und dem SGML-Paket hingegen kann eine detailliertere Funktionalität zugeordnet werden, da die Vorgehensweise beim Speichern und damit auch die Bedingungen für den Zugriff auf die Dokumente bekannt sind. Im folgenden werden die Klassen dieser beiden Pakete näher spezifiziert.

Die Pakete Bitstrom und SGML sollen Klassen bereitstellen, die das Speichern und Durchsuchen von strukturiert oder als Bitstrom vorliegenden Dokumenten ermöglichen. Es werden prinzipiell vier Klassen benötigt:

- Klasse zur Speicherung eines Dokuments als Bitstrom (Paket Bitstrom)
- Klasse zur strukturierten Speicherung von Dokumenten in der Datenbank (Paket SGML)

- Klasse zum Zugriff auf als Bitstrom gespeicherte Dokumente (Paket Bitstrom)
- Klasse zum Zugriff auf Dokumente in der Datenbank (Paket SGML)

Die Definition der Methoden dieser Klassen, die ihrerseits die bereits erwähnten atomaren Operationen realisieren, folgt in den nächsten Abschnitten.

Die Trennung der Funktionalität in Pakete und Klassen erfolgt auf dieser Ebene, um die Operationen einzeln zu entwickeln. Aus der Sicht einer Instanz, die Aufträge an den Speicher übermittelt, bilden sie eine einheitliche Schnittstelle zum Zugriff auf den Speicher.

4.7.2 Die Speicherklassen

Eine besondere Rolle kommt dem Vorgang des Speicherns zu. Bei allen anderen Vorgängen (Durchsuchen, Präsentieren, etc.) wird auf einem bereits gespeicherten Dokument operiert. Soll ein Dokument gespeichert werden, dann wird es während des Speichervorgangs übertragen und temporär zwischengespeichert. Um das Dokument zu speichern, müssen die Informationen Stück für Stück (bzw. Byte für Byte) aus dem temporären Dokument gelesen und dann im Speicher abgelegt werden. Die Vorgehensweise beim Speichern ist in Abbildung 4.4 dargestellt.

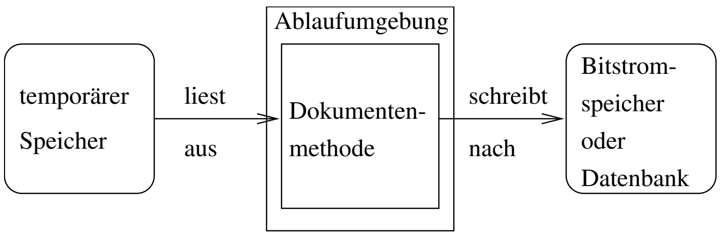


Abbildung 4.4: Der Speichervorgang

Das Speichern von Bitstromdaten ist eine verhältnismäßig einfache Aufgabe. Eine entsprechende Methode liest eine Anzahl von Bytes (oder Bits) aus dem temporären Zwischenspeicher, gegebenenfalls operiert sie auf den Daten (z.B. Dekomprimierung) und schreibt sie dann in den Bitstromspeicher. Ein einfacher Algorithmus in Python zum Speichern eines Bitstroms ist in Abbildung 4.5 gegeben.

```
obj=c_store() # instanziiere Objekt der Bitstromspeicherklasse
obj.open_in() # oeffne Eingabe-Bitstrom
obj.open_out() # oeffne Ausgabe-Bitstrom

decbuff=""
buff="␣"

# solange das Ende des Eingabe-Bitstroms nicht erreicht ist
while buff !="":
      # lese 1024 Bytes vom Eingabe-Bitstrom
      buff = obj.read_in(1024)

      # dekomprimiere eingelesene Daten
      decbuff=decompress(buff)

      # schreibe in Ausgabe-Bitstrom
      obj.write_out(decbuff)

# schliesse Bitstroeme
obj.close_out()
obj.close_in()
```

Abbildung 4.5: Algorithmus zur Bitstromspeicherung (Python)

Um aus der temporären Datei zu lesen, muß der Dokumentenmethode ein Dateihandle bzw. ein Dateiname mitgeteilt werden. Selbiges gilt für die Ausgabe in den Bitstromspeicher, auch hier muß ein Dateiname oder -handle bekannt sein. Diese Informationen werden von der Ablaufumgebung vor Ausführung der Methode ermittelt und dieser als globale Variable zur Verfügung gestellt. Diese Variablen benutzt die Bitstromspeicherklasse, um mittels entsprechender Methoden (*open_in*, *close_in*, *open_out* und *close_out*) auf die Bitströme zuzugreifen. Der Speichervorgang selbst benötigt Methoden, um aus der temporären Datei zu lesen (*read_in*) und in den Bitstromspeicher zu schreiben (*write_out*). Beide Methoden arbeiten byte-orientiert, so daß eine bestimmte Anzahl von Bytes gelesen oder geschrieben werden kann.

Für SGML-Dokumente stellt sich diese Aufgabe etwas schwieriger dar. Im Kapitel 2.1 wurde erläutert, aus welchen Elementen sich ein SGML-Dokument zusammensetzt, aus sogenannten Tags und den Taginhalten. Desweiteren benötigt man zur vollständigen Beschreibung eines SGML-Dokuments, dessen Document Type Definition (DTD). Um das Dokument zu speichern, muß die DTD nicht näher betrachtet werden, es muß lediglich sichergestellt werden, daß der Speicher bei Bedarf auf die DTD zugreifen kann.

Bei den im folgenden entwickelten Operationen zum Speichern von Dokumenten wird davon ausgegangen, daß diese in Normalform vorliegen. Das heißt, das Dokument ist vollständig strukturiert, es dürfen keine Endtags weggelassen werden. Da frei verfügbare SGML-Parser für alle Plattformen existieren, bedeutet dies keine Einschränkung.

Um komfortabel aus dem im Zwischenspeicher gehaltenen SGML-Dokument lesen zu können, bedarf es Methoden, um die einzelnen Elemente nacheinander einzulesen und die Attribute der Tags zu ermitteln. Weiterhin müssen Funktionalitäten vorgehalten werden, um den Strukturbaum des SGML-Dokuments speichern zu können, da dieser wie in Kapitel 2.2 für eine Reihe von Operationen benötigt wird. Das heißt, es muß möglich sein, nur die Tags eines SGML-Dokuments sequentiell auszulesen und im Speicher abzulegen.

In Abbildung 4.6 ist ein einfacher Algorithmus zur Speicherung eines SGML-Dokuments in der Datenbank dargestellt.

```
obj=c_sgmlstore() # instanziiere SGML-Speicherobjekt
obj.open_in() # oeffne Eingabe-Bitstrom
obj.save_doctype() # speichere DTD in der Datenbank


# lese solange naechstes SGML Element bis
# das Ende des Eingabe-Bitstroms erreicht ist
buff="␣"
while buff!=None:
      buff=obj.get_nextsgmlelement()

      # wenn es ein Tag ist speichere es in der
      # Tagklasse, sonst speichere es als Text
      # in der Datenbank
      if obj.is_tag(buff):
        obj.save_astag(buff)
      else:
        obj.save_astext(buff)

# schliesse Eingabe-Bitstrom
obj.close_in()
```

Abbildung 4.6: Algorithmus zur Dokumentenspeicherung (Python)

Wiederum erfolgt der Zugriff auf das zwischengespeicherte Dokument über ein zuvor von der Ablaufumgebung bereitgestelltes Dateihandle. Zur Speicherung der DTD des Dokuments existiert eine Methode (*save_doctype*), diese trägt die DTD in die dafür vorgesehene Klasse ein. Die Methoden *open_out* und *close_out* sind in diesem Fall nicht mehr erforderlich. Die Verbindung zu einem Datenbankserver wird automatisch und für den Methodenentwickler transparent auf- und abgebaut.

Eine Methode (*get_nextsgmlelement*), die unabhängig vom Elementtyp das nächste SGML-Element aus der temporären Datei liest, ist unverzichtbar. Stehen nur Methoden zur Verfügung, die lediglich bestimmte Elementtypen einlesen, muß der Autor der Dokumentenmethode immer die Reihenfolge der Elemente berücksichtigen, was zu einem nicht unerheblichen Mehraufwand bei der Programmierung führen würde.

Wie in Kapitel 4.6 beschrieben, wird beim Speichern in der Datenbank, nach Markup (Tags), Text und Binärdaten unterschieden. Daher ist es sinnvoll bereits beim Lesen aus dem Zwischenspeicher Mechanismen bereitzustellen, die ebenfalls diese Unterscheidung erlauben. Da beim Einlesen des nächsten Elements nicht bekannt ist, um welchen Elementtyp es sich dabei handelt, gibt es Methoden, die den Typ feststellen (*is_tag*, *is_text*, *is_bin*). Allerdings stößt man hier bei der Realisierung auf das Problem, Text und Binärdaten zu unterscheiden.

Neben der Methode zum Speichern eines Tags in der Datenbank (*save_astag*) existieren noch zwei weitere Methoden (*save_astext*, *save_asbin*) zum Ablegen der anderen beiden Elementtypen in den Datenbankklassen.

Ein weiteres Beispiel, das nur den Strukturbaum eines Dokuments abspeichert und dabei ein bestimmtes Tag überspringt, ist in Abbildung 4.7 dargestellt.

```
obj=c_sgmlstore() # instanziiere SGML-Speicherobjekt
obj.open_in() # oeffne Eingabe-Bitstrom

obj.save_doctype() # DTD in der Datenbank speichern

# lese solange naechstes SGML-Tag bis
# das Ende des Eingabe-Bitstroms erreicht ist
buff = "␣"

while buff != None:
      buff = obj.get_nexttag()

      # wenn das Tag eine Bemerkung ist,
      # ueberspringe den Inhalt durch suchen
      # des Endtags.
      if obj.split_tag(buff)[0]=="REMARK":
        obj.get_tag("REMARK")
      else:
        # sonst speichere Tag in der Datenbank
        obj.save_astag(buff)

# Bitstrom schliessen
obj.close_in()
```

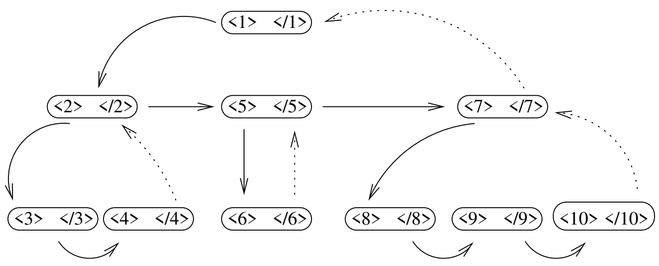
Abbildung 4.7: Algorithmus zur Strukturbaumspeicherung (Python)

Die Methode *get_nexttag* dient dazu, das jeweils nächste Tag, unabhängig davon ob es ein Start- oder Endtag ist, einzulesen. Das Pendant zu dieser Methode (*get_tag*) findet ein bekanntes Tag. Um ein gelesenes Tag und seine Attribute auswerten zu können, wird eine Methode (*split_tag*) verwendet. Zu einem gegebenen Tag liefert diese Methode eine Liste zurück, die den Tagnamen und alle Attribute beinhaltet.

Die Berechnung des vereinfachten Strukturbaums (Kap. 2.1) erfolgt zur Laufzeit der Dokumentenmethode. Dabei werden die Tags in der Reihenfolge, in der sie mittels der Methode *save_astag* abgelegt werden betrachtet und Bruder- und Sohn-Felder nach folgenden Regeln berechnet:

1. folgt auf ein Starttag wieder ein Starttag, dann ist das zweite Starttag der Sohn des Ersten
2. folgt auf ein Endtag ein Starttag, dann ist das zweite Tag der (rechte) Bruder des ersten Tags

In Abbildung 4.8 sollen diese Zusammenhänge veranschaulicht werden.



Tagreihenfolge: <1><2><3></3><4></4></2><5><6></6><7><8></8><9></9><10></10></7></1>

Abbildung 4.8: Durchlaufen eines Strukturbaums

Die Linien zeigen den Weg, der beim Durchlaufen des Baums mit Depth-First-Search abgeschritten wird. Die schwarzen Linien verdeutlichen die Vater-Sohn- bzw. Bruder-Beziehungen, die gepunkteten Linien vervollständigen den Weg. Der Einfachheit halber wurden die Tags numeriert. Korrespondierende Start- und Endtags stehen jeweils in einem Knoten, damit die Aufteilung der Teilbäume deutlicher wird. Mathematisch korrekt wäre es, die Endtags als (rechten) Bruder des Starttags darzustellen.

4.7.3 Die Zugriffsklassen

Der Zugriff auf die gespeicherten Dokumente kann auf die unterschiedlichsten Arten erfolgen. Die Art des Zugriffs soll jedoch nicht vom Speicher vorgegeben sein. Daher müssen die Zugriffsklassen einen generischen und flexiblen Ansatz verfolgen, der es ermöglicht Dokumentenmethoden zu entwickeln, die es erlauben ein oder mehrere Dokumente zu durchsuchen oder direkt auf Teile eines bekannten Dokuments zuzugreifen. Das Durchsuchen mehrerer Dokumente erfolgt über ein im Speicher abgelegtes „Zugriffs-Dokument“, dem Dokumentenmethoden zugeordnet sind. Diese erlauben es den ganzen oder Teile des Dokumentenbestands zu durchsuchen. Das „Zugriffs-Dokument“ beinhaltet selbst keine Daten.

Zum Durchsuchen aller gespeicherten Dokumente nach und/oder-verknüpften Begriffen wird der Volltextindizierer verwendet. Es wird eine Methode vorgesehen, die unter Benutzung der Schnittstelle zum Volltextindizierer, alle als Bitstrom gespeicherten Dokumente nach einem gegebenen Ausdruck durchsucht. Der Aufbau des Ausdrucks, sowie die Möglichkeiten zur Konfiguration hängen von der Auswahl des Indizierers ab. Da für die Schnittstellen von Volltextindizierern bislang keine Standardisierungsbemühungen existieren, ist es nicht möglich zu diesem Punkt genauere Vorgaben zu machen. Selbst reguläre Ausdrücke finden nur teilweise Verwendung bei den derzeit verfügbaren Indizierern. Als minimale Anforderung wird die Suche nach und/oder-verknüpften Begriffen definiert.

Eine weitere Möglichkeit die Dokumente auf bestimmte Merkmale zu prüfen, ist die Tag-Klasse der Datenbank zu durchsuchen. Dies ist in Kombination mit der Auswahl einer bestimmten DTD von Interesse. Hat man beispielsweise Kenntnis über die Semantik eines bestimmten Tags oder einer DTD, dann muß es Methoden geben, die folgende Dokumente aus dem Speicher selektieren können:

1. Alle Dokumente, die auf einer bestimmten DTD basieren
2. Alle Dokumente, die ein bestimmtes Tag beinhalten
3. Alle Dokumente, die einer bestimmten DTD entsprechen und ein bestimmtes Tag mit bestimmten Attributen enthalten

Als Ergebnis wird eine Liste der entsprechenden Dokumenten-ID's zurückgegeben. Um diese Funktionalität zu erreichen, bedarf es dreier Methoden: Eine, die zu einer gegebenen DTD alle Dokumente findet, die auf dieser DTD basieren, eine, um die Tag-Klasse nach einem gegebenen Tag zu durchsuchen, die Dritte kombiniert die beiden vorigen und findet alle Dokumente, die einer bestimmten DTD entsprechen und ein gegebenes Tag beinhalten.

Die beschriebenen Funktionalitäten dienen dem Auffinden eines Dokuments im Speicher. Im folgenden werden Operationen definiert, die zum Zugriff auf ein bestimmtes Dokument benötigt werden. Zunächst sind ein paar grundlegende Methoden zu definieren, um das gesamte Dokument auszulesen, den Strukturbaum eines Dokuments zu erhalten und die DTD eines Dokuments zu erfragen.

Für den byte-orientierten Zugriff stehen Methoden bereit, wie sie bereits beim Auslesen der temporären Datei beschrieben wurden. Der strukturierte Zugriff auf ein Dokument geschieht immer über Tags, da diese wie unter Kapitel 2.1 beschrieben, die Struktur des Dokuments manifestieren. Vorausgesetzt wird bei dieser Vorgehensweise immer die Kenntnis über die Struktur des Dokuments, die über die oben beschriebenen Operationen erlangt werden kann. Es werden Methoden bereitgestellt, die das n-te Tag oder alle Tags eines gegebenen Typs auffinden und dessen Inhalt (das von Start- und Endtag eingeschlossenen Dokumentenstück) auslesen können. Weiterhin stehen Methoden zum Auslesen der Attribute und der Anzahl der vorkommenden Tags eines Typs zur Verfügung.

Als Beispiel findet sich in Abbildung 4.9 eine Dokumentenmethode, die als Ergebnis ein Inhaltsverzeichnis aller Kapitel des Dokuments zurückgibt.

```
obj=c_sgmlquery() # instanziiere Zugriffsobjekt
# initialisiere Lauf- und Ergebnisvariablen
n=1
erg=["␣",0]
contents={}

# solange noch ein entsprechendes Tag gefunden wird
while erg!=["",0]:
      # hole n-tes KAPITEL-Tag aus DB
      erg=obj.sgmlquery_tag("KAPITEL",n)
      # rette Tag in Ergebnis Array
      contents[erg[1]]=erg[0]
      n=n+1

return contents # gebe Ergebnis zurueck
```

Abbildung 4.9: Dokumentenmethode zur Erstellung eines Inhaltsverzeichnisses

Hierbei wird die Methode (*sgmlquery_tag*), die das n-te Tag eines bestimmten Typs ermittelt, benutzt um nacheinander alle Tags vom Typ „Kapitel“ aus der Datenbank abzufragen. Diese werden in der richtigen Reihenfolge als Ergebnis zurückgegeben.

Über den Zugriff auf und die Speicherung von Dokumenten-Metadaten im Speicher kann keine definitive Aussage getroffen werden, da diese in Abhängigkeit vom Gesamtsystem festgelegt werden. Es ist aber positiv davon auszugehen, daß eine eindeutige Dokumenten-ID, ein Eigentümer, ein Autor und ein Name (Titel) zu jedem Dokument vorgehalten werden. Hier werden Methoden vorgesehen, die es ermöglichen nach eben diesen Merkmalen zu suchen und die ebenfalls eine entsprechende Liste mit Dokumenten-ID's zurückliefern.

4.8 Sicherheitsaspekte

Das Thema Sicherheit in digitalen Bibliotheken oder genauer in Speichersystemen ist äußerst komplex. An dieser Stelle sollen aus diesem Grund lediglich einige grundsätzliche Aspekte genannt werden, die bei der Realisierung von Speichersystemen Berücksichtigung finden sollten:

- Dokumentenechtheit; Es muß dafür Sorge getragen werden, daß die Dokumente konsistent sind, d.h. die Dokumente sind vollständig gespeichert und können nur im Original, also unmanipuliert ausgelesen werden.

- Systemsicherheit; Methoden dürfen nicht alle Privilegien bekommen (lowlevel Systemzugriffe), da sonst eventuell unkontrollierte Zugriffe auf Dokumente erfolgen können.
- Authentifizierung; Wenn eine Methode auf einem Dokument aktiviert werden soll, so muß vorher sichergestellt werden, daß die Instanz, die die Methode ausführen möchte, dazu berechtigt ist. Verschiedene Arten des Zugriffschutzes können über die Standardmechanismen gängiger DBMS für den SGML-Speicher und gängiger Betriebssysteme für den Bitstromspeicher realisiert werden.

In [Lag95] werden weitere Aspekte der Sicherheit in Speichersystemen besprochen und Ansätze zur Lösung diskutiert.

Kapitel 5

Realisierung eines Prototyps

Zur Evaluierung der getroffenen Entwurfsentscheidungen aus den Kapiteln 3 und 4 wurde ein Prototyp des Dokumentenspeichersystems entwickelt, dessen Realisierung hier erläutert werden soll. Zunächst wird die Auswahl der verschiedenen Basisprodukte und der Funktionsumfang des Prototyps dargestellt. Das Kernstück bildet die Beschreibung der einzelnen Komponenten des Dokumentenspeichersystems und der Programmbibliothek. Dabei soll im wesentlichen auf die Konfigurationsmöglichkeiten der einzelnen Komponenten eingegangen werden. Zum Abschluß werden die durchgeführten Tests dargestellt. Der vollständige Quellcode der Implementierung ist in Anhang E gegeben.

5.1 Rahmenbedingungen

Für die Implementierung des Prototyps wurde die Programmiersprache Python ([vR96c], [vR96b], [LW96]) ausgewählt. Python ist eine objekt-orientierte, frei verfügbare Interpretersprache, die standardmäßig eine Reihe von Klassenbibliotheken ([vR96a]) zur Netzwerkkommunikation mitbringt. Zur Realisierung eines Monitorelements mit graphischer Benutzeroberfläche stehen Schnittstellen zu mehreren Widget-Bibliotheken zur Verfügung. Unter diesen wurde Tk ([Ous94]) ausgesucht, da diese für mehrere Plattformen frei verfügbar ist. Weiterhin kann Python unter Benutzung der vorhandenen C/C++ Schnittstellen mit geringem Aufwand um eigene Klassenbibliotheken erweitert werden. Python-Implementierungen existieren für viele UNIX-Plattformen, darunter Linux, HPUX, AIX, Minix, SCO Unix, Sun OS, NeXt und Digital Unix. Eine Portierung auf die Microsoft Windows Welt ist ebenfalls verfügbar.

Als verwendete Basisprodukte wurden die ebenfalls frei verfügbaren Programme PostgreSQL 6.1 (Datenbankmanagementsystem, [JC+96]) und Glimpse 4.0 (Volltextindizierer, [MS94]) herangezogen.

PostgreSQL ist weitgehend zur „Structured Query Language“ (SQL, [BED96]) kompatibel und unterstützt objekt-orientierte Datenmodelle. Die verschiedenen plattformübergreifenden Schnittstellen (JDBC, ODBC, C, C++) sorgen dafür, daß PostgreSQL sehr geeignet für den Einsatz in verteilten Systemen ist.

Der Volltextindizierer Glimpse basiert auf einem Werkzeug zur Mustersuche in Volltexten, namens „agrep“ ([WM96]) und wurde bereits innerhalb des Projekts Harvest ([BDHM95]) erfolgreich eingesetzt. Auch wenn Glimpse nicht die speziellen Funktionalitäten von SGML-Indizierern aufweißt, so qualifizierte sich dieses Programm durch seinen hohen Grad an Konfigurierbarkeit und auf Grund der freien Verfügbarkeit.

Beide Basisprodukte sind auf einer Reihe von Plattformen lauffähig, in erster Linie auf den verschiedenen UNIX-Plattformen.

Als Bitstromspeicher wurden über das „Network Filesystem“-Protokoll (NFS) verbundene Massenspeicher (Festplatten) eingesetzt. NFS-Implementierungen existieren ebenfalls für alle gängigen Plattformen. Der Zugriff auf den Bitstromspeicher erfolgt über Verzeichnis- bzw. Dateinamen.

Der Prototyp wurde unter Linux auf Intel-Architekturen und Digital-Unix auf Alpha-Architekturen realisiert. Die prototypische Implementierung, weist gegenüber dem Entwurf einige Restriktionen auf:

- Die Speicher stellen nur eine Ablaufumgebung für Dokumentenmethoden bereit und zwar für Python.
- Es wurden keine Verfahren zur Authentifizierung implementiert.
- Es wurde keine Unterstützung zur Fehlerkorrektur bei der Datenübertragung zwischen den Komponenten realisiert.

Wie in Kapitel 3 beschrieben, sollen die verschiedenen Komponenten über eine zuverlässige Netzverbindung unter Benutzung des TCP/IP-Protokolls miteinander kommunizieren. Zu diesem Zweck werden Sockets der Unix-Domäne eingesetzt ([Ste92]), die dieses Protokoll unterstützen. Der damit zur Verfügung stehende Mechanismus bietet einen verbindungsorientierten Dienst zur Kommunikation zwischen Prozessen auf demselben oder verschiedenen Computern im Netz. Da diese Art der Kommunikation als zuverlässig gilt, wurde auf Verfahren zur Fehlerkorrektur bei der Implementierung verzichtet.

5.2 Die Schnittstellenkomponente

Die Aufgaben und die grundlegenden Funktionalitäten einer Schnittstellenkomponente wurden bereits in Kapitel 3.4.2 beschrieben. Dieser Abschnitt beschäftigt sich mit der Realisierung der Schnittstellenkomponente.

Die realisierte Schnittstellenkomponente wird über eine Konfigurationsdatei parametrisiert. In Tabelle 5.1 sind die Parameter und ihre Bedeutung zusammengefaßt.

Parameter	**Argumente**	**Bedeutung**
RECEIVE_PORT	<port>	Portadresse, unter der die Schnittstellenkomponente auf Aufträge wartet
BROKER_ADDRESS	<host>,<port>	Die Connectadresse des Brokers
MAP_MODULE	<filtermodule>	Pythonmodul, das die Filterroutinen bereitstellt
MAP_COMMAND	<command>, <filterfunction>	Zuordnung der Kommandos zu den Filterroutinen
MAX_ORDERS	<maxorders>	maximale Anzahl der offenen Aufträge der Schnittstellenkomponente
LOGFILE	<logfilename>	Dateiname der LOG-Datei

Tabelle 5.1: Parameter der Schnittstellenkomponente

Von besonderem Interesse sind die Parameter *MAP_MODUL* und *MAP_COMMAND*. Durch sie wird festgelegt, in welchem Pythonmodul die Filterroutinen definiert sind und welches Kommando durch welche Funktion gefiltert wird. Dabei wird der gesamte Auftrag als Parameter an die Funktion übergeben. Der Rückgabewert der Filterfunktion entspricht dem konvertierten Auftrag. Diese Filterroutinen können ebenfalls für die Rückgabewerte von Speicher bzw. Broker festgelegt werden. Die Funktionsweise der Schnittstellenkomponente wird in Abbildung 5.1 verdeutlicht.

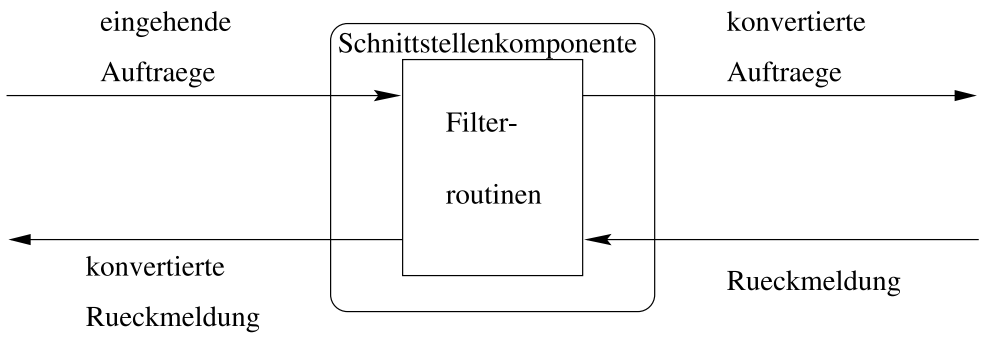


Abbildung 5.1: Funktionsweise der Schnittstellenkomponente

Eingehende Aufträge werden derart konvertiert, daß sie dem internen Protokoll des Speichersystems entsprechen.

Die an den Broker gesendeten Aufträge entsprechen einem der folgenden Formate:

- PUT <metadocument> <methodname> <methodparameters>
- ACTIVATE <docid> <methodname> <methodparameters>
- INFO

Die Rückmeldungen, die der Schnittstellenkomponente vom Broker übergeben werden, können folgende Gestalt annehmen:

- RESULT <docid> <result>
- ERROR <errorcode> <errormessage>

Bei Ausfall des Brokers und damit bei Unterbrechung der Verbindungen sendet die Schnittstellenkomponente eine Fehlermeldung an die auftraggebenden Instanzen.

5.2.1 Die Netzverbindungsklasse

Da sowohl der Broker als auch die Schnittstellenkomponenten eine Reihe von Netzverbindungen und damit Sockets parallel überwachen müssen, wurde für diese Aufgabe eine eigene Klasse implementiert. Die Klasse *c_multiconn* (multible connections) erzeugt, überwacht und verwaltet eine Menge von TCP/IP-Sockets der Internet-Domäne und unterstützt alle gängigen Socketoperationen.

Die Identifikation der einzelnen Sockets innerhalb der Socketmenge erfolgt über den eindeutig zugeordneten Socketdeskriptor. So liefert beispielsweise die Methode *open_sock*, die einen neuen Socket erzeugt, den korrespondierenden Socketdeskriptor zurück. Unter Angabe des Socketdeskriptors können dann weitere Operationen (z.B. lesen, schreiben, schließen) auf dem Socket ausgeführt werden. Als besonders nützlich erwies sich die Methode *wait_event*. Sie ermittelt unter Benutzung der Python select-Klasse alle Sockets, die innerhalb eines gegebenen Zeitraums aktiv sind und liefert eine Liste der zugehörigen Socketdeskriptoren zurück. Anhand der Deskriptoren und der Benutzung der Methoden *accept* und *sock_read* können die Ereignisse abgearbeitet werden. Eine Gesamtübersicht der implementierten Methoden dieser Klasse zeigt Tabelle 5.2.

Methode	Parameter	Funktionalität
Konstruktor	-	Initialisiert interne Variable.
Destruktor	-	Löscht interne Variable.
open_sock	<host>,<port>	Erzeugt einen Socket und bindet ihn an die übergebene Adresse.
close_sock	<socketnr>	Schließt den Socket, mit der übergebenen Socketnummer.
wait_event	<seconds>	Wartet <seconds> Sekunden auf Ereignisse an allen in der Socketmenge befindlichen Sockets. Für seconds=0 wird solange gewartet bis ein Ereignis eintritt.
write_sock	<socketnr>, <message>	Schreibt <message> auf den Socket, mit der Nummer <socketnr>.
read_sock	<socketnr>	Liest vom Socket <socketnr> und liefert als Ergebnis, die gelesene Nachricht zurück.
poll_sock	<socketnr>, <seconds>	Wartet <seconds> Sekunden an Socketnummer <socketnr> auf eine Nachricht.
connect	<socketnr>, <host>,<port>	Verbindet Socket <socketnr> mit der durch <host>,<port> angegebenen Adresse.
accept	<socketnr>	Führt einen Accept an Socketnummer <socketnr> aus und liefert als Ergebnis den neuen Verbindungssocket zurück.
get_localaddr	<socketnr>	Gibt die lokale Adresse des Sockets zurück

Tabelle 5.2: Methoden der Klasse c_multiconn

5.3 Der Broker

Der Broker stellt die zentrale Verwaltungsinstanz des Speichersystems dar. Er verteilt die von den Schnittstellenkomponenten eingehenden Aufträge an die Speicher, er beobachtet alle Netzverbindungen und steuert das ganze Speichersystem.

Die Konfiguration des Brokers wird aus einer Konfigurationsdatei gelesen. Die möglichen Parameter sind in Tabelle 5.3 zusammengestellt.

Parameter	Argumente	Bedeutung
RECEIVE_PORT	<port>	Portnummer für eingehende Aufträge von den Schnittstellenkomponenten
CONTROL_PORT	<port>	Portnummer zum Verbinden eines Monitor-/Steuerungswerkzeuges
CONNECT_PORT	<port>	Portnummer zum Verbinden der Speicher
MAX_ORDERS	<maxorders>	Maximale Anzahl der offenen Aufträge
LOGFILE	<logfilename>	Dateiname der LOG-Datei

Tabelle 5.3: Parameter des Brokers

Die Parameter *RECEIVE_PORT*, *CONTROL_PORT* und *CONNECT_PORT* können mehr als einmal in der Konfigurationsdatei angegeben werden. Der Broker wartet dann an allen Ports auf einen Verbindungsaufbau durch die jeweiligen Komponenten.

Zur Verteilung der Aufträge auf die Speicher wurde ein einfaches Verfahren angewandt. Der Broker unterhält eine Liste aller angemeldeten Speicher. Soll ein Auftrag vermittelt werden, ermittelt der Broker den ersten Speicher in der Speicherliste, der im Zustand *active idle* ist und leitet den Auftrag an diesen weiter (s. Kapitel 3.4.3).

Verbindet sich ein Monitor-/Steuerungswerkzeug an den Broker, so kann die aktuelle Konfiguration des Brokers durch Senden eines *CMD_CONFGET* Kommandos erfragt werden. Diese beinhaltet Informationen über

- den Zustand des Brokers
- alle angemeldeten Speicher und deren Zustände
- alle vorliegenden Aufträge und deren Zustände

Allen Speichern und dem Broker selbst wird ein innerhalb des Speichersystems eindeutiger Name zugeordnet, der zusammen mit der Konfigurationsinformationen an das Steuerungswerkzeug übertragen wird. Über das Kommando *CMD_CHGST* <Elementname> <Zustand> kann der Zustand dieser Elemente verändert werden. Der Parameter <Zustand> kann die Werte *activ*, *passiv* oder *terminated* annehmen, um Komponenten zu aktivieren, passivieren oder zu beenden.

Um den Broker anzuhalten oder fortzusetzen, stehen die zwei Kommandos *CMD_PAUSE* und *CMD_CONT* zur Verfügung. Zum Wechseln zwischen automatischer Verteilung der Aufträge durch den Broker und manueller Verteilung durch das Monitor-/Steuerungswerkzeug existieren die Kommandos *CMD_AUTO* und *CMD_MAN*.

Übermittelt das Monitor-/Steuerungswerkzeug ein Kommando der Form

CMD_MAPJOB <brokerinterne Auftrags-ID> <Speichername>,

während der Broker im Modus für manuelle Verteilung ist, so übermittelt der Broker den Auftrag zur Ausführung an den Speicher und vermerkt dies in seiner Auftragsverwaltung.

Weiterhin besteht die Möglichkeit die Speicher und den Broker durch Senden eines *CMD_EXIT* oder *CMD_QUIT* Kommandos auf einmal zu beenden. Im Unterschied zum *CMD_QUIT* Kommando wartet das *CMD_EXIT* Kommando bis alle in Bearbeitung befindlichen Aufträge abgearbeitet wurden, um die Speicher und den Broker zu beenden. Das *CMD_QUIT* Kommando trennt die Speicher und beendet den Broker sofort. Um einen Speicher zu beenden, sendet der Broker ein *CMD_EXIT* an ihn.

5.3.1 Verwaltung der Aufträge

Der Broker ist für die Auftragsverwaltung zuständig. Die aktiven Verbindungen zu den Schnittstellenkomponenten müssen auf die Verbindungen zu den Speichern abgebildet werden. Zu diesem Zweck wurde eine Klasse (*c_joblist*) implementiert, die zu jedem Auftrag folgende Informationen speichert:

1. Auftragsnummer
2. Socketdeskriptor der Verbindung zur Schnittstellenkomponente, die den Auftrag erteilt hat.
3. interne Nummer des Speichers
4. Socketdeskriptor der Verbindung zum Speicher, der den Auftrag erhalten hat.

Die Klasse *c_joblist* stellt Methoden bereit, um neue Aufträge der Liste hinzuzufügen (*addjob*), zu löschen (*remjob*) oder zu modifizieren, sowie Methoden, die unter Angabe eines Socketdeskriptors, die korrespondierende Verbindung ermitteln (*get_job_by_iffd*, *get_job_by_repfd*). Abbildung 5.2 zeigt die Klassendefinition der Klasse. Diese Klasse ermöglicht die Identifikation von Auftraggebern und den jeweiligen Speichern, ohne den Auftrag oder das Ergebnis zu lesen oder zu modifizieren. Die Zuordnung erfolgt alleine über die Netzverbindung.

```
class c_joblist:
    def __init__(self,maxjobs=20):
    def addjob(self,if_fd,ifstate,repnr,rep_fd=-1,repstate=""):
    def remjob(self,jobnr):
    def get_job_by_iffd(self,if_fd):
    def get_job_by_repfd(self,rep_fd):
    def set_ifstate(self,jobnr,newstate):
    def set_repstate(self,jobnr,newstate):
```

Abbildung 5.2: Die Klassendefinition der Klasse c_joblist

5.4 Das Monitor-/Steuerungswerkzeug

Das Monitor-/Steuerungswerkzeug realisiert eine graphische Benutzeroberfläche zur Steuerung des Dokumentenspeichersystems. Es besteht die Möglichkeit sich an den Broker des Dokumentenspeichersystems zu verbinden und aktuelle Statusinformationen über das System am Broker auszulesen. Diese Informationen werden graphisch dargestellt.

Ein Zeitintervall für die Aktualisierung der Statusinformationen kann eingestellt werden. Der Zustand der Speicher und des Brokers kann verändert werden, beispielsweise kann ein Speicher vorübergehend angehalten werden. Die Verteilung der Aufträge durch den Broker kann entweder automatisch oder manuell erfolgen. Wählt man die manuelle Verteilung, werden die eingehenden Aufträge beim Broker in eine Warteschlange gestellt und nur durch eine explizite Zuordnung zu einem Speicher an diesen weitergeleitet. Abbildung 5.3 zeigt das Hauptfenster der graphischen Oberfläche.

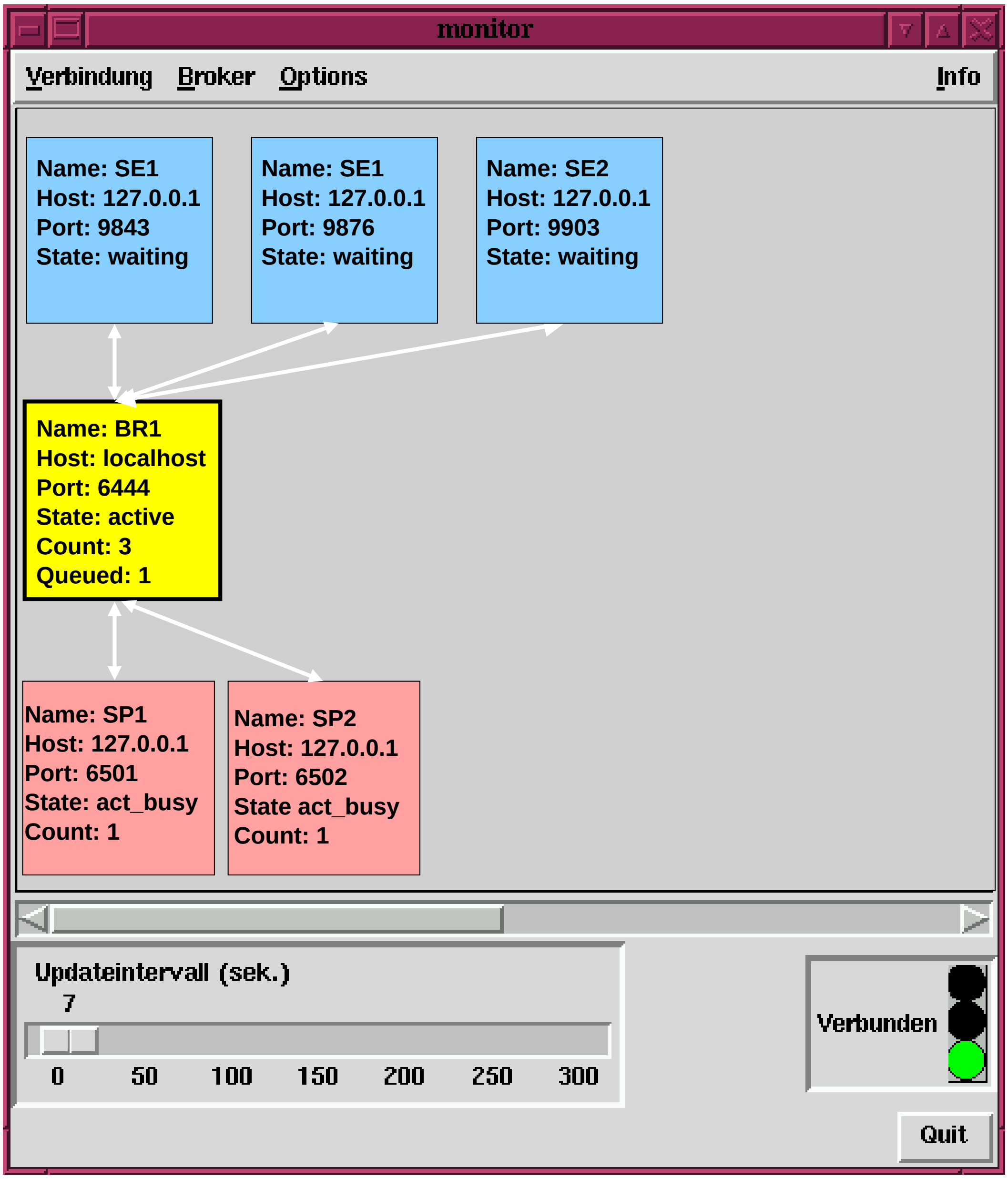


Abbildung 5.3: Screenshot des Monitor-/Steuerungswerkzeugs

Um die Bedienung so komfortabel wie möglich zu gestalten, sind die meisten Funktionalitäten direkt über Mausoperationen durchzuführen. Durch Doppelklick auf das Symbol des Brokers oder eines der Speicher wird ein Konfigurationsfenster geöffnet, das es gestattet den Zustand der ausgewählten Komponente zu ändern. In Abbildung 5.4 ist ein solches Fenster gezeigt.

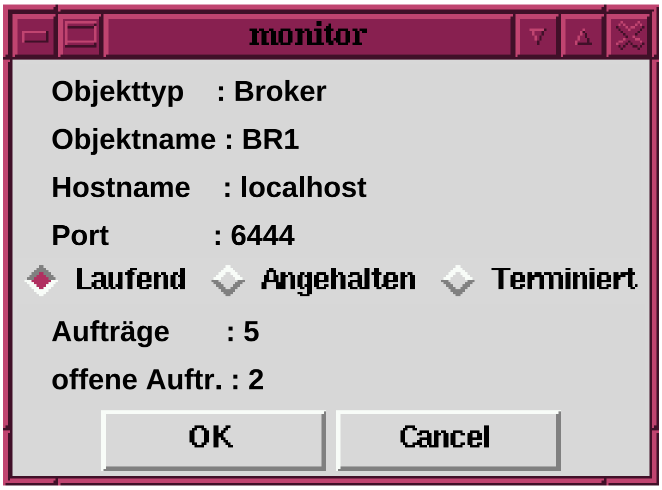


Abbildung 5.4: Screenshot des Konfigurationsfensters

Die manuelle Zuordnung von Aufträgen zu den Speicherkomponenten erfolgt ebenfalls über ein gesondertes Fenster. Es wird eine Liste aller verfügbaren Speicher angezeigt, in der man den Speicher auswählen kann, dem der Auftrag übergeben werden soll.

Die graphische Benutzeroberfläche wurde mittels der Python-Tk-Schnittstelle „Tkinter“ realisiert, die in [Haa96] beschrieben ist.

5.5 Der Speicher

Die Implementierung des Speichers wird, wie die der anderen Komponenten, über eine Konfigurationsdatei parametrisiert. Die möglichen Paramter und ihre Bedeutung sind in Tabelle 5.4 zusammengefaßt.

Parameter	**Argumente**	**Bedeutung**
RECEIVE_PORT	<port>	Portnummer für eingehende Aufträge vom Broker
BROKER_ADDRESS	<port>	Netzadresse des Brokers
LOGFILE	<logfilename>	Dateiname der LOG-Datei
MAP_MODULE	<filtermodule>	Pythonmodul, das die Filterroutinen bereitstellt
MAP_COMMAND	<command>, <filterfunction>	Zuordnung der Kommandos zu den Filterroutinen
EXECPYTHON	<pythonpath>	Pythoninterpreter, zur Ausführung der Dokumentenmethoden.
DB_SERVER_ADDRESS	<host>,<port>	Adresse des Datenbank-Servers
DB_NAME	<dbname>	Name der Datenbank
TMPPATH	<tmppath>	Verzeichnis für temporäre Dateien
BITSTREAMPATH	<bspath>	Verzeichnis für als Bitstrom gespeicherte Informationen
LIB_PATH	<libmcpath>	Verzeichnis der Programmbibliothek für mobilen Code (Paketverzeichnis)
FULLTEXT_RESULT	<resultfile>	temp. Datei für Volltextsuche

Tabelle 5.4: Parameter des Speichers

Wird ein Speicher gestartet, so meldet er sich beim Broker an und teilt diesem die Adresse mit unter der er auf Aufträge wartet.

Empfängt der Speicher ein *PUT* Kommando der Form

PUT <metadoc> <methodname> <methodparamters>,

so extrahiert er zunächst die bereits unter Kapitel 2.4.2 beschriebenen Metadaten und speichert sie unter Benutzung des System-Pakets der Programmbibliothek in der Datenbank ab. Bei diesem Vorgang wird auch eine eindeutige Dokumenten-ID für das neue Dokument erzeugt. Dann wird die Dokumentenmethode mit dem Namen <methodname> und den Parametern <methodparameters> auf dem Dokument aktiviert. Im allgemeinen wird es sich hierbei um eine Methode zur Speicherung der Dokumentendaten handeln. Ohne Aufruf einer solchen Methode werden nur die Metainformationen des Dokuments im Speicher abgelegt. Als Ergebnis wird die Dokumenten-ID (als Zähler implementiert) zurückgeliefert.

Handelt es sich bei dem empfangenen Kommando um eine Informationsanforderung (Kommando: *INFO*) über die in der Ablaufumgebung verfügbaren Pakete der Programmbibliothek, werden alle im Verzeichnis *LIB_PATH* vorhandenen Pakete durchsucht, die Klassen- und Methodeninformationen extrahiert und als Ergebnis (ASCII) zurückgeliefert.

Wird dem Speicher ein Kommando *CMD_EXIT* vom Broker gesendet, so verarbeitet der Speicher den momentan laufenden Auftrag und beendet sich.

Ein *ACTIVATE* Kommando bewirkt den Aufruf einer Dokumentenmethode auf einem bestimmten Dokument. Ein *ACTIVATE* Kommando hat folgende Gestalt:

ACTIVATE <docid> <methodname> <methodparameters>

Zunächst wird geprüft, ob ein Dokument mit der entsprechenden <docid> und einer Methode namens <methodname> gespeichert ist.

Die Methode wird dann in einer temporären Datei abgelegt. Diese Datei, die Dokumenten-ID und die Parameter werden an die Ablaufumgebung übergeben. Der Rückgabewert der Ablaufumgebung repräsentiert das Ergebnis der Methodenausführung, er wird als Antwort zurückgeschickt.

5.5.1 Anbindung an die Datenbank

Die Speicher brauchen Zugriff auf die Datenbank, um Metadaten abzulegen oder darauf zuzugreifen. Weiterhin müssen Dokumentenmethoden, die in der Ablaufumgebung der Speicher ausgeführt werden, ebenfalls auf die Datenbank Zugriff haben, um Dokumente zu speichern oder abzufragen.

Vereinfacht heißt das, daß aus Python heraus mit der Datenbank kommuniziert werden muß. Zu diesem Zweck wurde Python um ein Modul zur Anbindung an die PostgreSQL-Datenbank erweitert. Das PostgreSQL-Datenbankmanagementsystem wird über eine TCP/IP-Verbindung angesprochen. Über diese Verbindung (Connectionhandle) werden SQL-Anweisungen an die Datenbank gesendet. Als Ergebnis erhält man ein sogenanntes „Resulthandle“. Dieses Resulthandle wird dazu verwendet, auf die Ergebnisse und Statusmeldungen der ausgeführten SQL-Anweisung zuzugreifen. Eine besondere Rolle spielen binäre Objekte (Bilder, Tondateien, u.s.w.), um diese Informationen in Postgres ablegen zu können, bedarf es spezieller Routinen.

Die Kommunikation mit der Datenbank unter Benutzung von „Connectionhandles“ und „Resulthandles“ ist der von Microsoft definierten „Open Database Connectivitiy“-Schnittstelle (ODBC) sehr ähnlich ([SSC95]).

Das realisierte Modul stellt drei Klassen zur Verfügung. Eine Klasse (*pgconn*) stellt alle Funktionalitäten bereit, die zum Verbindungsauf- und abbau, sowie zum Versenden von SQL-Statements notwendig sind. Eine weitere Klasse (*pgres*) unterstützt die Auswertung von Resulthandles zur Verwertung der Ergebnisse. Eine dritte Klasse (*pgblob*) schließlich realisiert den Zugriff auf Binärobjekte.

5.5.2 Realisierung der Anbindung

Das DBMS PostgreSQL bietet zur Einbindung in Fremdapplikationen eine C-Bibliothek. Auf der anderen Seite unterstützt Python Mechanismen zur Integration von C-Routinen ([vR95]). Es bot sich daher an, Python um spezielle Klassen zur Kommunkiation mit PostgreSQL zu erweitern. Damit können SQL-Befehle genauso wie Standard-Python-Befehle benutzt werden. Der Zugriff kann aus der Ablaufumgebung, sowie aus dem Speicherprozeß heraus erfolgen.

Die C-Bibliothek operiert im wesentlichen auf drei verschiedenen Datenstrukturen: Eine für die Verbindung zum Datenbankserver (Connectionhandle), eine für die Verarbeitung von Abfrageergebnissen (result handle) und eine für den Zugriff auf binäre Datenbankobjekte (blobs). Diese drei Datenstrukturen wurden auf drei Pythonklassen abgebildet und deren Prozeduren auf die Methoden der Klassen. Die Implementierung dieser Klassen erfolgte in der Programmiersprache C ([KR88]). Die Vorgehensweise soll im folgenden skizziert werden.

Das Arbeiten mit Klassen und deren Instanzen (Objekten) erfolgt unter Python dreistufig.

1. Importieren einer Klassenbibliothek (Modul)
2. Instanziieren des Objekts
3. Methodenaufrufe ausführen

Ein Stück Pythoncode, das diese drei Schritte beschreibt ist in Abbildung 5.5 dargestellt.

```
# importieren des PostgreSQL <--> Python Moduls
import pgpy

# instanziieren eines Objekts der Verbindungsklasse
# DL_PGHOST, DL_PGPORT geben die Netzadresse des DB-Servers an
# DL_DBNAME gibt den Namen der Datenbank an
obj = pgpy.pgconn( DL_PGHOST, DL_PGPORT, DL_DBNAME )

# Aufruf der Methode exec_query, Ausfuehren einer SQL Abfrage
query_result = obj.exec_query("select␣*␣from␣doc_desc;")
```

Abbildung 5.5: Arbeiten mit Klassen unter Python

Diese drei Schritte können auf die C-Implementierung der Pythonerweiterung abgebildet werden. Um die Vorgehensweise von Python beim Modulgebrauch zu erläutern, sollen die Schritte den entsprechenden C-Abläufen gegenübergestellt werden.

Um ein Modul und damit die darin enthaltenen Klassen in einem Pythonprogramm verfügbar zu machen, wird eine import-Anweisung ausgeführt. Die import-Anweisung bewirkt das Ausführen der Modulinitialisierungsroutine, diese liest eine im Modul zu definierende Klassentabelle ein, die ihrerseits auf die jeweiligen Klasseninitialisierungsroutinen verweist. Dieser Zusammenhang ist in Abbildung 5.6 dargestellt.

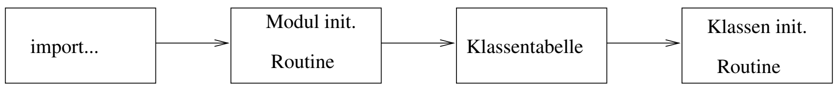


Abbildung 5.6: Import eines Moduls

Wird ein Objekt der geladenen Klasse instanziiert, so wird die jeweilige Klasseninitialisierungsroutine aufgerufen, diese führt den Klassenkonstruktor aus und liefert als Ergebnis ein Objekt der Klasse zurück.

Der Aufruf des Konstruktors bewirkt, daß ein neues Objekt erzeugt wird, alle dazu notwendigen Informationen werden in einer Objektbeschreibungstabelle festgelegt. Der Ablauf einer Instanziierung ist in Abbildung 5.7 verdeutlicht.

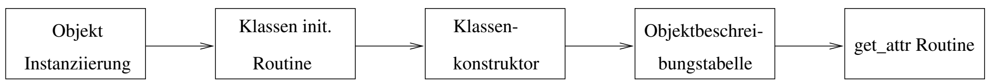


Abbildung 5.7: Objektinstanziierung

Eine wichtige Methode jeder Klasse ist die *get_attr* Routine. Sie wird benötigt, um die Methodenaufrufe aufzulösen und die eigentlichen Methoden auszuführen. Der Vorgang eines Methodenaufrufs ist in Abbildung 5.8 skizziert.

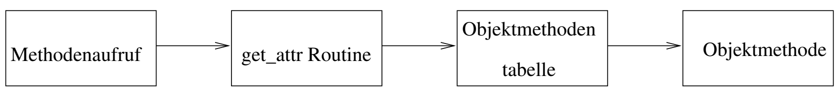


Abbildung 5.8: Methodenaufruf

Die Methode *get_attr* ermittelt unter Zuhilfenahme der Objektmethodentabelle die auszuführende Methode und ruft diese auf. Der Vorteil dieser Vorgehensweise ist die Möglichkeit der einfachen Erweiterbarkeit und der übersichtlichen Programmierung.

Der vollständige Quellcode zu diesem Pythonmodul ist in Anhang D abgedruckt.

5.5.3 Repräsentation der Dokumente in der Datenbank

Die in Abschnitt 4.6 vorgestellte einfache Klassenhierarchie wurde benutzt und um einige Klassen und Relationen zur Speicherung der Daten aus den Metadokumenten (Kap. 2.4.2) erweitert. Das erweiterte Datenmodell der Implementierung ist in Abbildung 5.9 skizziert.

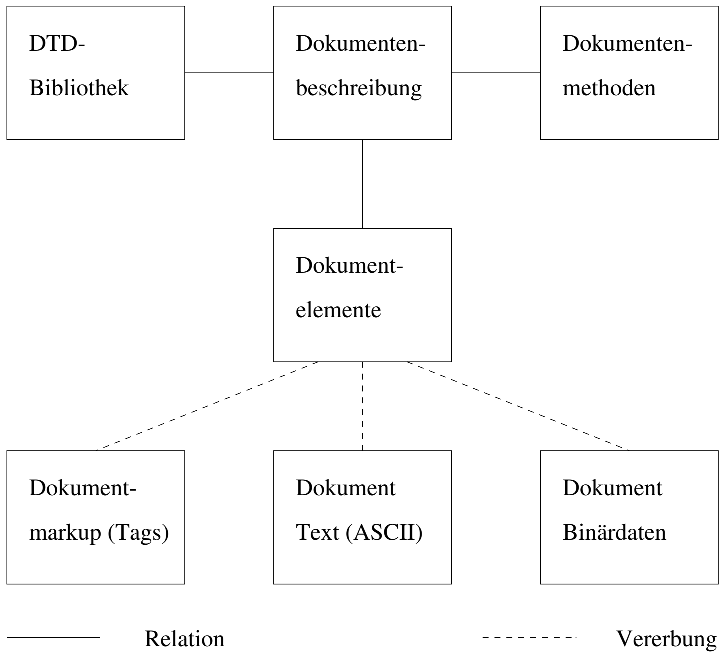


Abbildung 5.9: Datenmodell der Implementierung

Den hinzugekommenen Klassen kommen folgende Aufgaben zu:

- Dokumentenbeschreibung; beinhaltet alle Merkmale aus den Metadokumenten (Autor, Eigentümer, Titel, Info, u.s.w.), außer den Methoden und dem eigentlichen Dokument
- Dokumentenmethoden; beinhaltet die Referenzen der Dokumentenmethoden aus den Metadokumenten

Die Klassen sind über die eindeutige Dokumenten-ID miteinander verknüpft. Da die Klassen jeweils die Daten aller gespeicherten Dokumente aufbewahren und die gespeicherten Datenmenge pro Klasse dadurch sehr schnell wächst, war es notwendig den Datenzugriff zu beschleunigen.

Hierzu wurden Hash-Indizes auf das Verknüpfungsfeld (Dokumenten-ID) aller Klassen gesetzt. Um den Zugriff auf die Strukturinformationen ebenfalls effizient zu gestalten, wurden die Felder, die Bruder- und Sohnverweise, sowie die fortlaufende Elementnummer aufnehmen ebenfalls indiziert.

5.5.4 Ablaufumgebung für Dokumentenmethoden

Die Ablaufumgebung für Dokumentenmethoden wird innerhalb des Speichers durch eine einzige Pythonfunktion (*exec_mc*) realisiert. Als Parameter werden dieser Funktion die Dokumenten-ID, eine temporäre Datei, die die Dokumentenmethode enthält, die Argumente der Methode und optional eine Datei, die das eigentliche Dokument enthält, übergeben.

Die Übergabe der Methodenargumente erfolgt über die Shell-Schnittstelle. Eine Dokumentenmethode muß diese Argumente selbständig einlesen, um sie verarbeiten zu können. Der letzte Parameter wird immer dann benötigt, wenn durch die Methode ein neues Dokument im Speichersystem abgelegt werden soll.

Die Methode erzeugt eine temporäre Datei. Diese Datei wird aus drei Abschnitten zusammengesetzt: Dem Dateiheader, dem Initialisierungsabschnitt und dem eigentlichen mobilen Code. Der Dateiheader legt anhand der in der Konfigurationsdatei gesetzten Variablen *EXECPYTHON* den zu benutzenden Interpreter fest. Auf diese Art ist es möglich, einen Interpreter mit verminderter Funktionalität einzusetzen, um Sicherheitsrisiken zu vermindern. Der Header beinhaltet noch einige Kommentarzeilen, die lediglich der Fehlersuche und dem Verständnis dienen sollen.

Im Initialisierungsabschnitt werden einige Variablen gesetzt, die von verschiedenen Methoden der Programmbibliothek benutzt werden. Dazu gehören die Variable für die Dokumenten-ID (*DOCID*) und die Variable für die Dokumentendatei (*CURRENT_INFILE*), falls dieser Wert als Parameter übergeben wurde. Der letzte Abschnitt beinhaltet schließlich den mobilen Code.

Wurde die temporäre Datei zusammengesetzt, so wird sie ausgeführt. Jegliche Ausgaben der Methode werden wiederum zwischengespeichert. Dieser Teil des Quellcodes ist in Abbildung 5.10 zu sehen.

```
def exec_mc(docid,methodtmpfile,params,datafname=""):
      # erzeuge eindeutigen Dateinamen fuer temporaere Datei
      exectmpfile=class_funcs.getuniquename()
      # erzeuge Fileobjekt und oeffne temporaere Datei
      fd1=c_bsio.c_bsio(exectmpfile,"a+")
      fd1.open()
      # setze Methode zusammen
      .
      .
      .
      # Methode ausfuehren und Rueckmeldung schicken
      posix.chmod(exectmpfile,0755)
      execstr=exectmpfile+" "+params+"&1>"+TMPPATH+"output"+"&2>"
        +TMPPATH+"errout"

      if os.system(execstr)==0:
        # Methode fehlerfrei ausgefuehrt
        fdd=open(DL_TMPPATH+"output","r")
        ret="RESULT "+str(docid)+"\n"
      else:
        # es ist ein Fehler aufgetreten
        fdd=open(DL_TMPPATH+"errout","r")
        ret="ERROR "+str(docid)+"\n"

      # hole Methoden Output
      erg=""
      buff=" "
      while buff!="":
        buff=fdd.readline()
        erg=erg+buff
      fdd.close()

      ret=ret+erg
      del erg,buff,fdd
      .
      .
      .
```

Abbildung 5.10: Ausschnitt aus dem Quellcode der Ablaufumgebung

Terminiert die Methode ohne Fehler, wird als Ergebnis eine Zeile mit dem Inhalt:

RESULT <docid>

gefolgt von der Standardausgabe der Methode zurückgegeben. Beendet sich die Methode mit einer Fehlermeldung, wird als Ergebnis die Standardfehlerausgabe der Methode zurückgegeben, diesem steht folgende Zeile voran:

ERROR <docid>

Eventuell nicht terminierende Endlosschleifen wurden nicht berücksichtigt und werden nicht entdeckt. Die Ablaufumgebung stellt also nicht sicher, daß eine ausgeführte Methode auch terminiert.

5.6 Die Programmbibliothek für Dokumentenmethoden

Die Programmbibliothek für Dokumentenmethoden, die für den Prototyp realisiert wurde, stellt fünf verschiedene Pakete (s. auch Kap. 4.7) zur Verfügung. Es handelt sich dabei um Pakete

- zur Verarbeitung von Bitströmen und zur Benutzung des Volltextindizierers Glimpse (Paket Bitstrom)
- zum Einlesen, Speichern und Abfragen von SGML-Dokumenten (Paket SGML)
- zum Abfragen, Speichern von Metainformationen (Paket System)
- zur Wartung der Datenbank und des Volltextindizierers (Paket Maintenance)
- zum Versenden von Dokumenten oder Dokumentteilen über eine Netzverbindung, sowie zur Kommunikation über TCP-Sockets, aus den Dokumentenmethoden heraus. (Paket Netz)

Eine komplette Aufstellung der Inhalte der Pakete findet sich in Anhang C. Auf das Paket Netz wird an dieser Stelle nicht näher eingegangen, da das Paket nicht zwingend notwendig zum Betrieb des Speichersystems ist.

Das Einbinden der Pakete in Python erfolgt mittels einer import-Anweisung folgender Art:

```
from p_<paketname> import [*|c_<klasse1>,c_<klasse2>,...]
```

Dabei beginnen alle Paketnamen mit p_ und alle Klassennamen mit c_.

Die Pakete *p_bitstream*, *p_sgml*, *p_system* und *p_maintenance* werden im folgenden genauer erläutert, sofern dies nicht bereits in Kapitel 4.7 geschehen ist. Diese Pakete stellen die Funktionalitäten bereit, die zur Speicherung und den Zugriff auf die Dokumente notwendig sind. Die vollständige Klassenhierarchie der Pakete ist in Anhang B gegeben.

Bei der Implementierung der Pakete ist zu beachten, daß bei Zugriffen auf die Datenbank oder die Bitstromschnittstelle, so wenige Datensätze bzw. Dateien wie möglich zur Bearbeitung gesperrt werden. Ist das nicht der Fall, so kann es unter Umständen zu langen Wartezeiten bei der Ausführung von Methoden kommen, weil die Daten durch einen anderen Speicher bzw. durch die Ausführung einer anderen Dokumentenmethode gesperrt sind.

5.6.1 Das Bitstrom-Paket

Das Bitstrom-Paket (*p_bitstream*) stellt vier Klassen bereit, um Bitströme zu bearbeiten. Die Basisklasse *c_bsio* dient dazu Standardoperationen auf Dateien auszuführen. Sie unterstützt das Öffnen, Schließen, Lesen, Schreiben, das zeilenweise Lesen von Textdateien und das Positionieren der Lese-/Schreibmarke auf einer Datei.

Die Klasse *c_store* wird von der Basisklasse *c_bsio* abgeleitet und bietet Methoden, um ein Dokument als Bitstrom im Speichersystem abzulegen. Diese Klasse besitzt jeweils zwei open- und zwei close-Methoden. Davon wird ein Paar benutzt, um auf die zuvor von der Ablaufumgebung erstellten temporären Kopie des Dokuments zuzugreifen, das zweite Paar öffnet und schließt den neuen Bitstrom im Speichersystem. Zu bemerken ist, daß sowohl die Zuordnung des neuen Bitstroms zum Dokument, als auch die Bereitstellung des Dokumenteninhalts als temporäre Datei transparent erfolgen und vom Entwickler der Dokumentenmethode nicht berücksichtigt werden muß.

Hinzu kommen, eine Methode um von der temporären Kopie des Dokuments zu lesen, eine Methode um Daten in den Bitstromspeicher zu schreiben und eine Methode (*register_store*), um den Vorgang der Bitstromspeicherung in den im Speichersystem aufbewahrten Metainformationen des Dokuments zu registrieren. Weiterhin stehen Methoden zur Positionierung der Lese-/Schreibmarke beider Bitströme zur Verfügung Abbildung 5.11 zeigt die Definition der Klasse.

```
class c_store:
        def __init__(self):
        def open_in(self):
        def open_out(self):
        def close_in(self):
        def close_out(self):
        def read_in(self,size):
        def write_out(self,outstr):
        def register_store(self):
        def seek_in(self,offset,whence):
        def seek_out(self,offset,whence):
        def tell_in(self):
        def tell_out(self):
```

Abbildung 5.11: Definition der Klasse c_store

Eine Dokumentenmethode zum Abspeichern eines Dokuments als Bitstrom, sowie genaue Erläuterungen der Funktionalitäten wurden bereits in Kapitel 4.6 angegeben.

Die Klasse *c_glimp_search* realisiert eine Schnittstelle zur Volltextsuche. Das Basisprodukt Glimpse wird über die Shell-Schnittstelle aufgerufen, dazu wird ein Kommandozeileninterpreter (shell) gestartet, dem ein entsprechender Programmaufruf übergeben wird.

Bei der Instanziierung eines Objekts der Klasse *c_glimp_search* ist es möglich Parameter für die Glimpse-Aufrufe dieses Objekts festzulegen. Damit steht die gesamte Funktionalität von Glimpse zur Verfügung.

Abbildung 5.12 zeigt den Quellcode der Methode der *c_glimpse_search*-Klasse, die den Aufruf von Glimpse realisiert.

```
def glimp_searchor(self,searchstr):
  import os
  erg=[]
  # Aufruf von Glimpse mittels der Shell-Schnittstelle
  os.system('glimpse -y -H '+self.dir+' '+self.params
       +' '+searchstr+' >'+self.resfile)
  # Einlesen der Ergebnisse in die Ergebnis-Variable
  fd=open(self.resfile,'r')
  buff=' '
  while buff!='':
    buff=fd.readline()
    erg.append(buff[:-1])

  fd.close()
  # temp. Datei loeschen
  os.unlink(self.resfile)
  # Ergebnis zurueckgeben
  return erg[:-1]
```

Abbildung 5.12: Die Methode glimp_searchor der c_glimp_search Klasse

Die Methode *glimp_search* realisiert eine Suche nach mehreren und/oder- verknüpften Zeichenketten. Da Glimpse in der vorliegenden Version lediglich nach oder-verknüpften Zeichenketten suchen kann, übernimmt die Methode die und-Verknüpfung der einzelnen Terme. Zu beachten ist, daß der Suchstring in konjunktiver Normalform angegeben werden muß. Dabei wird eine Konjunktion durch ein & und eine Disjunktion durch ein ; angegeben. Beispielsweise wird durch Angabe von „verteilte&systeme;bibliotheken“ nach allen Dokumenten gesucht, die das Wort „verteilte“ und entweder das Wort „system“ oder „bibliotheken“ enthalten. Diese Klasse soll als Beispiel dafür angeführt werden, daß man mit geringem Aufwand andere Basisprodukte zur Benutzung aus den Dokumentenmethoden einbinden kann.

Die vierte Klasse des Pakets (*c_search*) stellt Methoden zur Suche in einem bekannten Dokument bereit. Unter Angabe der Dokumenten-ID kann man mit dieser Klasse gezielt nach Begriffen innerhalb des benannten Dokuments suchen. Die Funktionalität und Benutzung der Methoden ist analog zu denen der Klasse *c_glimp_search* definiert.

5.6.2 Das SGML-Paket

Das SGML-Paket stellt drei Klassen bereit, eine zum Speichern von SGML-Dokumenten in der Datenbank, eine zum Zugriff auf ein Dokument und eine zum Durchsuchen aller Dokumente nach SGML-Strukturmerkmalen.

Die benötigten Methoden zum Speichern von SGML-Dokumenten wurden, ebenso wie die Vorgehensweise beim Zugriff auf SGML-Dokumente bereits in Kapitel 4.7 detailiert beschrieben.

Die Basisklasse *c_sgmlread* und die abgeleitete Klasse *c_sgmlstore* stellen die dort definierten Methoden zu Speicherung von SGML-Dokumenten bereit.

Zum Zugriff auf die in der Datenbank gespeicherten SGML-Dokumente wurden die Klassen *c_sgmlquery* und *c_sgmlsearch* implementiert.

Die Instanziierung eines Objekts der Klasse *c_sgmlquery* erfolgt unter Angabe einer Dokumenten-ID. Diese Klasse dient dem Zugriff auf ein bestimmtes Dokument. Es werden hier Methoden bereitgestellt, um auf das ganze Dokument oder nur auf den Strukturbaum (*sgmlquery_structure*) zuzugreifen.

Zur Abfrage bestimmter Strukturmerkmale werden die Methoden *sgmlquery_tagcontents*, zur Abfrage des Inhalts eines Tags und *sgmlquery_tag*, zur Abfrage des Tags bereitgestellt. Beide Methoden erwarten als Parameter den Namen eines Tags und eine Zahl, die angibt das wievielte Tag dieser Art aus dem Dokument abgefragt werden soll. Die Anzahl der Vorkommen eines bestimmten Tags im Dokument kann mittels der Methode *sgmlquery_tagcount* festgestellt werden. Weiterhin steht zur Vearbeitung von abgefragten Tags eine Methode zur Vefügung, um ein Tag in seinen Namen und seine Attribute aufzuteilen (*sgmlquery_splittag*).

Die Klasse *c_sgmlsearch* realisiert sechs Methoden zum Durchsuchen der Datenbank nach Begriffen bzw. Zeichenketten. Die Methode *sgmlsearch_tag* durchsucht die vollständige Tagklasse, die Methode *sgmlsearch_text* die gesamte Textklasse und die Methode *sgmlsearch_doc* beide Klassen nach einer gegebenen Zeichenfolge.

Zusätzlich wurde eine virtuelle Methode (*sgmlsearch_bin*) definiert, die zum Durchsuchen von Binärdaten in der Datenbank vorgesehen ist. Diese Methode muß von der Dokumentenmethode in einer abgeleiteten Klasse überschrieben werden.

Um auch in Abhängigkeit einer bestimmten DTD, Dokumente auffinden zu können, wurden die Methoden *sgmlsearch_dtddocs* und *sgmlsearch_dtdandtag* realisiert. Die Methode *sgmlsearch_dtddocs* verlangt als Parameter die Public- oder System-ID einer DTD. Als Ergebnis wird eine Liste der Dokumenten-ID aller

Dokumente, die dieser DTD entsprechen zurückgegeben. Die Methode *sgml-search_dtdandtag* verlangt zusätzlich einen Parameter, der ein Tag angibt. Sie ermittelt, die Dokumenten-ID's aller Dokumente, die der übergebenen DTD entsprechen und solche Tags beinhalten.

5.6.3 Das System-Paket

Das System-Paket wird, wie unter Kapitel 4.7 beschrieben, hauptsächlich vom Speicher selbst benutzt. In ihm werden Klassen bereitgestellt, die zur Speicherung der und dem Zugriff auf Metadokumente dienen. Weiterhin verwenden die Dokumentenmethoden des „Zugriffs-Dokuments" (Kapitel 4.7.1) Teile dieser Klassen bei der Durchsuchung der Metadaten.

Das Paket beinhaltet drei Klassen. Eine Klasse dient dazu, als Bitstrom vorliegende Metadokumente auszulesen (*c_metadoc*), eine Klasse, um die Metainformationen eines Dokuments aus der Datenbank auszulesen (*c_metaquery*) und eine Klasse, um alle in der Datenbank vorliegenden Metainformationen zu durchsuchen (*c_metasearch*).

Wird dem Speicher zusammen mit einem *PUT*-Kommando ein Metadokument übermittelt, werden die unter Kapitel 2.4.2 beschriebenen Informationen (Autor, Eigentümer, Titel, u.s.w.) in der Datenbank gespeichert. Zu diesem Zweck stellt die Klasse *c_metadoc* Methoden zum Auslesen des Autors, des Eigentümers, des Titels und der DTD bereit.

Analog dazu stellt die Klasse *c_metaquery* Funktionalitäten bereit, um diese Informationen unter Angabe einer Dokumenten-ID aus der Datenbank abzufragen. Die Klassendefinition beider Klassen ist in Abbildung 5.13 angegeben.

```
class c_metadoc(c_sgmlread.c_sgmlread):
        def __init__(self,fname):
        def get_dtd(self):
        def get_header(self):
        def get_info(self):
        def get_property(self):
        def get_methods(self):
        def get_data(self):
        def __del__(self):

class c_metaquery:
        def __init__(self,ident):
        def mquery_info(self):
        def mquery_property(self):
        def mquery_methods(self):
```

Abbildung 5.13: Klassendefinition der Klassen c_metadoc und c_metaquery

Zu beachten ist, daß die Methoden *get_property* und *mquery_property* das komplette PROPERTY-Tag des Metadokuments (Anhang A) auslesen und die darin enthaltenen Informationen in einem assoziativen Feld (dictionary) zurückgeben.

Das „Zugriffs-Dokument“ benutzt die Klasse *c_metasearch* zum Durchsuchen aller in der Datenbank gespeicherten Metainformationen. Alle Methoden dieser Klasse durchsuchen die Datenbank nach einer übergebenen Zeichenkette, ohne Unterscheidung von Groß- und Kleinschreibung (case-insensitive). Dabei wird der SQL-LIKE-Operator verwendet. Auf diese Weise werden auch ähnliche Zeichenketten erkannt. Die Klasse sieht Methoden zum Auffinden von Dokumenten-ID, Autor, Eigentümer und Titel vor.

5.6.4 Das Maintenance-Paket

Das Maintenance-Paket besteht aus zwei Klassen. Eine Klasse ist zur Administration der vom Volltextindizierer (glimpse) erstellten Indizes (*c_glimp_index*) bestimmt und eine zur Wartung der Datenbank (*c_maintenance*).

Die Klasse *c_glimp_index* ermöglicht das Löschen und Erzeugen von Indizes mittels glimpse. Dazu wurden die Methoden *build_index* und *remove_index* implementiert.

Zur Reorganisation der Indizes ist es ausreichend die Methode *build_index* aufzurufen. Zu bemerken ist, daß die Reorganisation auch während des Betriebs des Speichersystems durchgeführt werden kann. Das System muß dazu nicht angehalten werden.

In der Klasse *c_maintenance* finden sich Methoden zur Reorganisation der Datenbank und zum Löschen eines Dokuments. Eine SQL-Schnittstelle zur Datenbank ist ebenfalls vorgesehen, diese kann benutzt werden, um beispielsweise statistische Daten (Anzahl der gespeicherten Dokumente, usw.) abzufragen oder andere administrative Vorgänge durchzuführen. Diese Methode liefert eine Ergebnistabelle im ASCII-Format zurück.

5.7 Evaluation des Prototyps

Die Evaluation des Prototyps erfolgte unter Zuhilfenahme des unter Kapitel 5.4 beschriebenen Monitor-/Steuerungswerkzeuges und eines Testwerkzeuges zur Simulation von Instanzen, die Aufträge an das Speichersystem senden. Ein Screenshot dieses Testwerkzeugs ist in Abbildung 5.14 gezeigt. Es können die Kommandos *PUT*, *ACTIVATE* und *INFO* mit den beschriebenen Parametern an eine Schnittstellenkomponente abgesetzt werden.

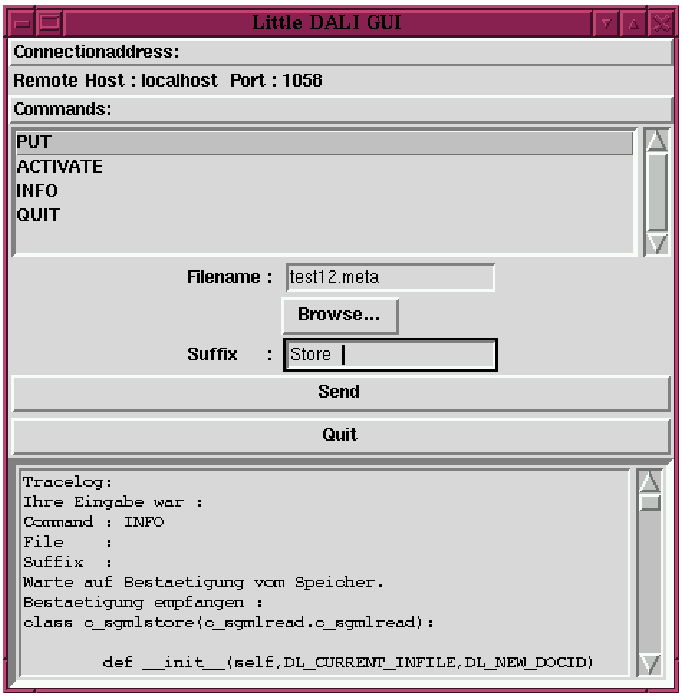


Abbildung 5.14: Screenshot des Testwerkzeugs

Als Testdaten wurden SGML-Dokumente verwendet, die innerhalb des Projekts „Digitale Bibliothek Frankfurt“ zur Verfügung standen. Zu diesen Dokumenten wurden Dokumentenmethoden mit verschiedenen Funktionalitäten entwickelt. Die Dokumentenmethoden im einzelnen:

1. Speichern des Dokuments (Bitstrom und Datenbank)
2. Durchsuchen des Dokuments nach Begriffen
3. Generierung des SGML-Strukturbaums
4. Abfragen von Teildokumenten
5. Ermitteln der Anzahl eines Tagtyps im Dokument

Es wurde ein „Zugriffs-Dokument“ eingerichtet und mit Dokumentenmethoden verknüpft. Die Methoden im einzelnen:

1. Volltextsuche über alle Dokumente
2. Suche in den Metainformationen der Datenbank, nach Autor, Eigentümer, Titel oder Dokumenten-ID
3. Suche nach Tags in allen Dokumenten

Für die Testumgebung wurden mehrere Speicher und Schnittstellenkomponenten in das Speichersystem eingebunden. Die Durchführung der Tests erfolgte in zwei unabhängig voneinander durchgeführten Testreihen.

Die erste Testreihe galt der Verifikation der korrekten Ausführung der Kommandos. Hierzu wurde das Speichern (*PUT*), die Aktivierung unterschiedlicher Methoden auf den Dokumenten inkl. des Zugriffs-Dokuments (*ACTIVATE*) und das Abfragen von Informationen über die Programmbibliothek (*INFO*) ausgiebig getestet.

Die zweite Testreihe diente der Überprüfung der Funktionalitäten des verteilten Systems. Zuerst wurde das Monitor-/Steuerungswerkzeug gegen die in Kapitel 3.4.3 gegebene Spezifikation getestet. Das Ein- und Ausgliedern, sowie das Anhalten und Fortsetzen der Speicher und des Brokers und die manuelle Verteilung der Aufträge auf die Speicher wurden geprüft.

Die Anbindung an die PostgreSQL-Datenbank durch das in dieser Arbeit realisierte Python-Modul wurde zusätzlich innerhalb anderer Anwendungen des Projekts getestet und funktionierte einwandfrei. Im Verlauf der Tests stellte sich jedoch heraus, daß PostgreSQL bei der Abfrage von Binärdaten gelegentlich während der Datenübermittlung mit einer Fehlermeldung abbricht. Der Fehler soll, nach Auskunft der Entwickler in einer der nächsten Releases von PostgreSQL behoben werden.

Alle anderen Tests verliefen erfolgreich.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel faßt den Inhalt der vorliegenden Arbeit zusammen und gibt einen Ausblick auf weitere, in dieser Arbeit nicht behandelte Problemstellungen im Zusammenhang mit Speichersystemen in verteilten digitalen Bibliotheken.

6.1 Zusammenfassung

Die Entwicklung eines Konzepts und die Realisierung eines Prototyps eines Dokumentenspeichersystems für digitale Bibliotheken wurden in dieser Arbeit beschrieben.

Ziel dieser Arbeit war es, ein Dokumentenspeichersystem für verteilte digitale Bibliotheken zu entwickeln, das skalierbar ist und mit wenig Aufwand in vorhandene und zukünftige Gesamtsysteme digitale Bibliotheken eingebunden werden kann. Weiterhin sollten Funktionalitäten bereitgesellt werden, die es ermöglichen multimediale Dokumente im Speichersystem zu verwalten. Besondere Aufmerksamkeit sollte dabei der Verarbeitung von SGML-Dokumenten gewidmet werden.

Aus [MD98] wurden Konzepte zur typgerechten Verarbeitung von Dokumenten in einer verteilten digitalen Bibliothek übernommen. Die Grundidee wurde aus den in [DK95] vorgestellten wünschenswerten Eigenschaften eines Speichersystems zur Verwendung in digitalen Bibliotheken entwickelt.

Diese Eigenschaften wurden in die Ziele, die mit einem solchen Speichersystem verfolgt werden, umgesetzt. Nach Einführung einiger Grundlagen über digitale Bibliotheken, SGML und mobilen Code wurde das Projekt „Digitale Bibliothek Frankfurt“ detailiert besprochen.

Im nächsten Schritt wurde die Architektur des Speichersystems entwickelt. Dabei wurden zunächst die Anforderungen an ein solches verteiltes System aus den verfolgten Zielen abgeleitet. Die Auswahl einer geeigneten Architektur schloß sich an. Das entwickelte Speichersystem basiert auf einer Three-Tier-Architektur mit Broker und besteht aus unterschiedlichen Komponenten, die über ein definiertes Kommunikationsprotokoll interagieren. Die Schnittstellenkomponente sieht Mechanismen zur einfachen Anbindung an Gesamtsysteme digitaler Bibliotheken vor und steuert die Kommunikation mit auftraggebenden Instanzen. Der Broker übernimmt die vollständige Vermittlung der an das Speichersystem gerichteten Aufträge und der internen Steuerbefehle. Das Monitor-/Steuerungswerkzeug dient der Beobachtung und Konfiguration des Systems. Den Speicherkomponenten kommt die eigentliche Auftragsverarbeitung zu.

Dabei ist die Anzahl der Schnittstellenkomponenten und der Speicher frei skalierbar, so daß das Speichersystem an die Anzahl der zu verarbeitenden Aufträge angepaßt werden kann. Die Menge der Speicher fungiert dabei als Lastverbund auf einem Datenbestand. Durch die Bündelung der Vermittlungsfunktionalitäten im Broker, wird die Anpassung von Speichern und Schnittstellenkomponenten an Basisprodukte bzw. andere Gesamtsysteme zusätzlich vereinfacht.

Das entwickelte verteilte System kann als Grundlage für die Entwicklung anderer Elemente (Katalog-, Erfassungselemente) verwendet werden, da die Konzeption von Schnittstellenkomponenten, Broker und Managementkomponenten unabhängig von der Funktionalität der Speicher erfolgte.

Es schloß sich die Entwicklung der Funktionalitäten der Speicher an. Auch hier galt es zunächst aus den gesteckten Zielen die Anforderungen abzuleiten. Der entworfene aktive Speicher verarbeitet Metadokumente und erlaubt die Ausführung von Dokumentenmethoden auf den Dokumenten. Zur Ausführung von Dokumentenmethoden stellt der Speicher eine oder mehrere Ablaufumgebungen bereit. Die Funktionalitäten und Ressourcen des Speichers werden den Dokumentenmethoden durch die Programmbibliotheken zur Verfügung gestellt. Zur Organisation der Programmbibliotheken wurde das Paket-Klassen-Konzept entwickelt. Die von der Programmbibliothek bereitgestellten Operationen bilden die Schnittstelle zwischen Dokumenten und Speicher. Durch den gänzlich modularen Aufbau des Speichers wurde dafür Sorge getragen, daß der Speicher leicht modifiziert und erweitert werden kann.

Besondere Berücksichtigung fand die Entwicklung SGML-spezifischer Funktionalitäten. Zu diesem Zweck wurde ein spezielles Datenmodell, sowie entsprechende Pakete hergeleitet. Der Speicher wurde derart konzipiert, daß die Basisprodukte (DBMS, Volltextindizierer) mit geringem Aufwand eingebunden werden können. Daraus resultierend stehen viele gängige Datenbanksysteme und Indizierer für den Einsatz im Speichersystem zur Auswahl.

Unabhängig voneinander wurden zum einen eine Architektur für das Dokumentenspeichersystem und zum anderen der Aufbau, der im Speichersystem verwendeten Speicher entworfen.

In beiden Fällen geschah dies unter Berücksichtigung verschiedener vorhandener Ansätze und bekannter Verfahren.

Schließlich wurde ein Prototyp zur Evaluierung der getroffenen Entwurfsentscheidungen realisiert. Der Prototyp zeichnete sich neben den entwurfsbedingten Eigenschaften, durch den Einsatz von frei verfügbaren Basisprodukten, eine integrierte Datenbankanbindung und eine graphische Benutzeroberfläche aus.

6.2 Ausblick

Das in dieser Arbeit entwickelte Dokumentenspeichersystem bietet eine Reihe von Ansatzpunkten zur Weiterentwicklung und zeigt Problemstellungen auf, die in der Arbeit nicht behandelt werden konnten, zum einen, zur Verbesserung des verteilten Systems, zum anderen, zum Ausbau des Speichers um neue Funktionalitäten.

Eine Erweiterung zur Verbesserung der Leistungsfähigkeit ist die Definition eines Broker-Broker-Protokolls zur Kommunikation zwischen Brokern. Mit Hilfe dieses Protokolls ist es möglich mehrere Broker parallel einzusetzen. Zu bemerken ist, daß Mechanismen für den Fall eines Broker-Ausfalls vorgesehen werden müssen, die es ermöglichen, daß ein anderer Broker den ausgefallenen Broker lückenlos ersetzen und alle Aufträge übernehmen kann.

Zur Absicherung des in dieser Arbeit vorgestellten Konzepts, ist es wünschenswert detailierte Anforderungen an die Basisprodukte zu definieren. Dazu gehören Anforderungen an die Ausfallsicherheit und den Leistungsumfang, der Datenbank und des Volltextindizierers.

Zur Entkoppelung von Steuerfunktionen und den eigentlichen Funktionalitäten der Komponenten ist der Einsatz von Threads in Erwägung zu ziehen. Jede Komponente startet einen Steuerthread, der Befehle vom Broker (Anhalten, Fortsetzen, usw.) abarbeitet und einen Auftragsthread, der die eigentlichen Aufträge verarbeitet.

Auch zur Erweiterung des aktiven Speichers, bieten sich einige interessante Möglichkeiten.

Zunächst ist die Definition neuer Pakete der Programmbibliothek zu erwähnen. Ein gutes Beispiel für diese Art der Erweiterung stellt die Definition von Operationen auf Video-, Bild- und Tondaten dar. Dies schließt den Entwurf von Speichern ein, die auf die Besonderheiten anderer Dokumenttypen eingehen. Die Implementierung von Ablaufumgebungen für andere geeignete Programmiersprachen zählt ebenfalls zu den nennenswerten Ausbaumöglichkeiten des Speichers.

In diesem Bereich der Forschung über digitale Bibliotheken gibt es viel Raum für neue Ideen und interessante Problemstellungen.

Literaturverzeichnis

[ANS86] American National Standards Institute (Hrsg.): *Coded character set – 7-bit American national standard code for information interchange, ANSI X3.4-1986*, 1986.

[BAH94] Böhm, Klemens, Karl Aberer und Christoph Hüser: *Extending the Scope of Document Handling: The Design of an OODBMS Application Framwork for SGML Document Storage*. Technical Report 811, GMD, 1994.

[BAN95] Böhm, Klemens, Karl Aberer und Erich Neuhold: *Administering Structured Documents in Digital Libraries*. In: *Advances in Digital Libraries*, Heidelberg, 1995. Lecture Notes in Computer Science, Springer Verlag.

[BDHM95] Bowman, C., P. B. Danzig, Darren R. Hardy und U. Manber: *Harvest: A Scalable Customizable Discovery and Access System*. [Online im Internet], Mai 1995. URL:ftp://ftp.cs.colorado.edu/pub /techreports/schwartz/Harvest.Conf.ps.Z [Stand: 21.11.1997].

[BED96] Bowman, Judith S., Sandra L. Emerson und Marcy Darnovsky: *The practical SQL handbook*. Addison-Wesley, Reading, MA, 1996.

[BF93] Borenstein, N. und N. Freed: *MIME (Multipurpose Internet Mail Extension) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies, RFC 1521*. Technischer Bericht Network Working Group, 1993.

[Bir95] Birmingham, William P.: *An Agent-Based Architecture for Digital-Libraries*. D-Lib Magazine, [Online im Internet], July 1995. URL:http://www.dlib.org/dlib/July95/07birmingham.html [Stand: 08.01.1998].

[BLFFN95] Berners-Lee, T., R.T. Fielding und H. Frystyk Nielsen: *Hypertext Transfer Protocol - HTTP/1.0, Internet Draft draft-ietf-http-v10-spec-00*. Technischer Bericht HTTP Working Group, 1995.

[Bry88] Bryan, Martin: *SGML: An Author's Guide to the Standard Generalized Markup Language*. Addison-Wesley, 1988.

[CK+81] Cerf, V., R. Kahn et al.: *Transmission Control Protocol (TCP), Darpa Internet Program Protocol Specification, RFC793*. Technischer Bericht Information Sciences Institute, University of Southern California, Marina del Rey, California, September 1981.

[DIN91] Deutsches Institut für Normung (Hrsg.): *Informationsverarbeitung - Textverarbeitung und -kommunikation - Genormte Verallgemeinerte Auszeichnungssprache (SGML), DIN EN 28879:1991*, 1991.

[DK95] Drobnik, O. und H. J. Kiowski: *Entwicklung eines Systems zur Strukturierung, Speicherung und Bereitstellung von Dokumenten als Teil einer Infrastruktur für digitale Bibliotheken*. Antrag auf Gewährung von Sachbeihilfe, 1995.

[Gol88] Goldfarb, Ch. F.: *The SGML Handbook*. Oxford University Press, 1988.

[Haa96] Haas, Markus: *Referenz zum Python Modul Tkinter*. Institut für angewandte Informatik und formale Beschreibungskomplexität, Universität Karlsruhe (TH), 1996.

[Heu94] Heuschkel, D.: *Basisexperimente zur Leistungsmessung verteilter Anwendungen*. Diplomarbeit, Universität Frankfurt, 1994.

[IBM88] IBM Corp.: *IBM Local Area Network Technical Reference, Dok.-Nr. SC30-3383-2*, 3. Auflage, November 1988.

[ISO86] International Organization for Standardization: *Information processing - Text and office systems - Standard Generalized Markup Language (SGML), ISO 8879 :1986*, 1986.

[ISO92] International Organization for Standardization: *Database Language SQL*, 1992. ISO/IEX 9075:1992.

[JC+96] Ju, A., J. Chen et al.: *The PostgreSQL User Manual*. [Online im Internet], 1996. URL:http://s2k-ftp.cs.berkeley.edu:8000/postgres95 [Stand: 21.11.1997].

[KNS90] Klas, Wolfgang, Erich Neuhold und Michael Schrefl: *Metaclasses in VODAK and their Application in Database Integration*. Technischer Bericht 462, GMD, Sankt-Augustin, 1990.

[Knu73] Knuth, Donald E.: *The Art of Computer Programming Volume 3 Sorting and Searching*. Addison-Wesley, Reading, MA, 1973.

[Knu84] Knuth, Donald E.: *The TeXbook*. Addison-Wesley, Massachusetts, 1984.

[KR88] Kernighan, Brian W. und Dennis M. Ritchie: *The C Programming Language*. Prentice Hall, Englewood Cliffs, NJ, 2. Auflage, 1988.

[Lag95] Lagoze, Carl: *A Secure Repository Design for Digital Libraries*. D-Lib Magazine, [Online im Internet], Dezember 1995. URL:http://www.dlib.org/dlib/december95/12lagoze.html [Stand: 15.12.1997].

[Lam86] Lamport, L.: *LaTex: A Document Preparation System*. Addison-Wesley, Massachusetts, 1986.

[LDD95] Lingnau, A., O. Drobnik und P. Dömel: *An HTTP-based Infrastructure for Mobile Agents*. In: *World Wide Web Journal - Fourth International World Wide Web Conference Proceedings*, Boston, MA, Dezember 1995.

[LW96] Lutz, Mark und Frank Willson: *Programming Python*. Nutshell Handbook. O'Reilly & Associates, Inc., Sebastopol, CA, 1. Auflage, Oktober 1996.

[MD98] Mönch, Christian und Oswald Drobnik: *Integrating new document types into digital libraries*. Erscheint in : Proceedings of the 1998 Advances in Digital Libraries Conference, IEEE Computer Society Press, 1998.

[Mic89] Microsofot Corporation: *SMB Protocol Extensions Version 2.0 [Online im Internet]*, November 1989. URL:ftp://ftp.microsoft.com/developr/drg/CIFS/SMB.TXT [Stand: 15.01.1998].

[MS94] Manber, Udi und Wu Sun: *GLIMPSE: A Tool to Search Through Entire File Systems*. In: *Winter USENIX Technical Conference*, Tucson, Arizona, 1994. University of Arizona.

[NFL+95] Nürnberg, Peter, Richard Furuta, John J. Leggett, Catherine C. Marshall und Frank M. Shipman II: *Digital Libraries: Issues and Architectures*. In: Shipman III, Frank M., Richard Furuta und David M. Levy (Herausgeber): *The Second Annual Conference on the Theory and Practice of Digital Libraries*, Proceedings of Digital Libraries 95, Seiten 147–153, Austin, Texas, Juni 1995.

[Nov92] Novell Inc.: *Novell IPX Router Specification Novell Part Number 107-000029-001.* [Online im Internet], September 1992. URL:ftp://ftp.innet.net/pub/networking/routing/ipx /ipxrout.zip [Stand: 08.12.1997].

[Ous94] Ousterhout, J.K: *Tcl and the Tk Toolkit*. Addison Wesley, Reading, MA, 1994.

[Pos82] Postel, John: *Simple Mail Transfer Protocol (SMTP), RFC821*. Technischer Bericht Information Sciences Institute, University of Southern California, Marina del Rey, California, August 1982.

[RBP+91] Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy und W. Lorensen: *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[Rie95] Rieger, Wolfgang: *SGML für die Praxis, Ansatz und Einsatz von ISO 8879*. Springer Verlag, Heidelberg, 1995.

[Sch90] Schirmer, C.: *troff-Programmierung*. Carl Hanser Verlag, München, Wien, 1990.

[SM94] Sollins, K. und L. Masinter: *Functional Requirements for Uniform Resource Names (URN), RFC 1737*. Technischer Bericht Network Working Group, Dezember 1994.

[SSC95] Stegman, Michael O., Robert Signore und John Creamer: *The ODBC Solution, Open Databse Connectivity in Distributed Environments*. McGrawHill, 1995.

[Ste92] Stevens, Richard W.: *Programmieren von UNIX-Netzen*. Carl Hanser Verlag, München, 1992.

[Ste94] Stevens, Richard W.: *TCP/IP Illustrated Volume 1, The Protocols*. Addison-Wesley, 1994.

[Sun95] Sun Microsystems, Inc.: *NFS Version 3 Protocol Specification, RFC 1813 [Online im Internet]*, Juni 1995.
URL:http://www.internic.net/rfc/rfc1813.txt [Stand: 15.01.1998].

[Tol96] Tolksdorf, Robert: *Die Sprache des Web: HTML 3*. dpunkt, Verlag für digitale Technologie, Heidelberg, 2. Auflage, 1996.

[vR95] Rossum, Guido van: *Extending and Embedding the Python Interpreter*. [Online im Internet], 1995. URL:http://www.python.org [Stand: 21.11.1997].

[vR96a] Rossum, Guido van: *Python Library Reference*. [Online im Internet], 1996. URL:http://www.python.org [Stand: 21.11.1997].

[vR96b] Rossum, Guido van: *Python Reference Manual*. [Online im Internet], 1996. URL:http://www.python.org [Stand: 21.11.1997].

[vR96c] Rossum, Guido van: *Python Tutorial*. [Online im Internet], 1996. URL:http://www.python.org [Stand: 21.11.1997].

[WM96] Wu, Sun und Udi Manber: *AGREP - A Fast aproximate pattern-matching tool*. [Online im Internet], 1996.
URL:ftp://cs.arizona.edu/agrep/agrep.ps.2 [Stand: 15.01.1998].

Anhang A

Die Document Type Definition der Metadokumente

STAND: Juli 1997

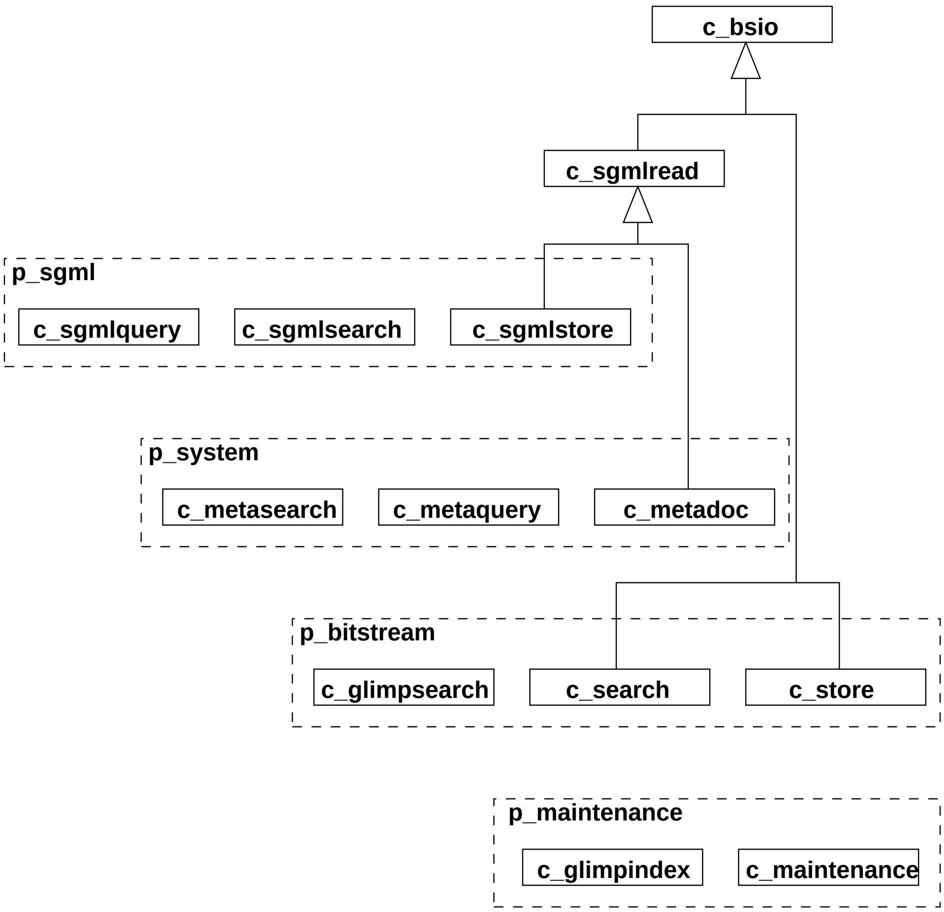
```
<!--

 Meta DTD v0.1
 10/96 Christian Moench

 Document Type Definition for the meta documents of
 the Digital Library. This DTD may become known as public
 entity: "-//DIGLIB//DTD META 0.1//EN".


 Typical usage:

    <!DOCTYPE META SYSTEM "/usr/lib/sgmlcat/meta.dtd">
    <META>
       ...
    </META>

-->

<!--   <!DOCTYPE META [   -->

    <!ELEMENT meta    - - (header, data)>
    <!ATTLIST meta id      ID    #REQUIRED
                  version CDATA #REQUIRED
                  creator CDATA #REQUIRED
                  owner   CDATA #REQUIRED>
```

```
    <!ELEMENT header - O (info?, property, (method)+)>
    <!ELEMENT info   - O CDATA>

    <!ELEMENT property - O EMPTY>
    <!ATTLIST property title  CDATA #REQUIRED
                       author CDATA #REQUIRED
                       owner  CDATA #REQUIRED
                       ident  CDATA #REQUIRED
                       pubkey CDATA #REQUIRED>

    <!ELEMENT method - O EMPTY>
    <!ATTLIST method name CDATA #REQUIRED
                     code CDATA #REQUIRED
                     mime CDATA #REQUIRED>

    <!ELEMENT data - O CDATA>
    <!ATTLIST data storage (coded|extern) "coded">

<!-- ]> -->
```

Anhang B

Klassenhierarchie der Python Programmbibliothek

Die Notation für die Klassenhierarchie ist [RBP+91] entnommen. Die in den gestrichelten Kästen zusammengefaßten Klassen bilden jeweils ein Paket.



Anhang C

Verzeichnis der Programmbibliothek für Dokumentenmethoden

Hier folgt ein Verzeichnis aller in den Paketen realisierten Klassen und der implementierten Basisklassen der Programmbibliothek für Dokumentenmethoden.

Soweit nicht anders angegeben, wird beim Auftreten eines Fehlers während der Ausführung einer Methode -1 als Rückgabewert geliefert.

C.1 Basisklassen

Von den Basisklassen werden einige der Klassen in den Paketen abgeleitet. Die Basisklassen stellen grundlegende Funktionalitäten zum Zugriff auf als Bitstrom vorliegende Dokumente bereit.

C.1.1 Klasse c_bsio

In dieser Klasse werden grundlegende Operationen auf Bitströme (Dateien) realisiert.

Konstruktor(filename,mode) - initialisert Objektvariable, `filename` gibt den Namen der Datei an, `mode` den Zugriffsmodus.

open() - öffnet die Datei.

close() - schließt die Datei.

read(size) - liest `size` Bytes aus der Datei und gibt die gelesene Zeichenkette zurück.

readline - liest solange aus der Datei bis ein Zeilenende erreicht ist und gibt die gelesene Zeichenkette zurück.

write(buff) - schreibt `buff` an der aktuellen Postition der Lese-/Schreibmarke in die Datei.

seek(offset,whence) - setzt die Lese-/Schreibmarke auf die Position `offset`, relativ zu der durch `whence` angebenen Position. Werte für `whence` sind: 0 für Dateianfang, 1 für aktuelle Position, 2 für Dateiende.

tell() - gibt die aktuelle Postition der Lese-/Schreibmarke zurück.

C.1.2 Klasse c_sgmlread

Diese Klasse stellt Methoden zum Einlesen von SGML-Dokumenten aus einer Datei bereit.

Konstruktor(fname) - initialisert Objektvariable, `fname` gibt den Dateinamen des SGML-Dokuments an.

seek_begin() - setzt die Lese-/Schreibmarke auf den Anfang der Datei.

seek_end() - setzt die Lese-/Schreibmarke auf das Ende der Datei.

get_doctype() - gibt das Doctype-Tag des SGML-Dokuments zurück.

get_dtdname() - gibt den Namen der DTD des SGML-Dokuments zurück.

get_dtdfile() - gibt den Dateinamen der DTD zurück.

get_dtd() - gibt die einzelnen Attribute des Doctype-Tags in Form einer Liste zurück.

get_tag(str) - sucht das nächste Tag mit Namen `str` und gibt es zurück.

get_nexttag() - sucht das nächste Tag und gibt es zurück.

get_tagcontents(str) - gibt den Inhalt des nächsten Tags mit Namen `str` zurück.

split_tag(tagstr) - teilt das als `tagstr` übergebene Tag in den Namen des Tags und seine Attribute auf. Zurückgegeben wird ein Dictionary, dessen Schlüssel die Namen der Attribute sind.

C.2 Paket Bitstream (p_bitstream)

Das Paket Bitstream beinhaltet Klassen, die den Zugriff auf den Bitstromspeicher realisieren.

C.2.1 Klasse c_store

Diese Klasse wird benutzt um Dokumente als Bitstrom im Speicher abzulegen.

Konstruktor - übernimmt die Dokumenten-ID, sowie die Dateinamen des Eingabe- und Ausgabebitstroms von der Ablaufumgebung.

open_in() - öffnet den Eingabebitstrom.

open_out() - öffnet den Ausgabebitstrom.

close_in() - schließt den Eingabebitstrom.

close_out() - schließt den Ausgabebitstrom.

read_in(size) - liest `size` Bytes vom Eingabebitstrom.

write_out(buff) - schreibt `buff` in den Ausgabebitstrom

seek_in(offset,whence) - verändert die Position der Lese-/Schreibmarke des Eingabebitstroms, wie unter **seek** der Klasse **c_bsio** beschrieben.

seek_out(offset,whence) - verändert die Position der Lese-/Schreibmarke des Ausgabebitstroms, wie unter **seek** der Klasse **c_bsio** beschrieben.

tell_in() - liefert die aktuelle Position der Lese-/Schreibmarke des Eingabebitstroms.

tell_out() - liefert die aktuelle Position der Lese-/Schreibmarke des Ausgabebitstroms.

register_store() - vermerkt in der Datenbank, daß das Dokument als Bitstrom im Speicher abgelegt wurde.

C.2.2 Klasse c_search

Diese Klasse stellt Methoden zur Volltextsuche in einem bestimmten Dokument bereit.

Konstruktor(dokid) - initialisiert Objektvariable, in `dokid` steht die Dokumenten-ID, des Dokuments, das durchsucht werden soll.

search(searchstr) - führt eine Volltextsuche nach den in `searchstr` gegebene Begriffen durch. Der Aufbau des `searchstr` wurde in Kapitel 5.6 beschrieben.

searchor(searchstr) - führt eine Volltexsuche nach den in `searchstr` gegebene Begriffen auf allen als Bitstrom gespeicherten Dokumenten durch. Läßt jedoch nur Oder-Verknüpfungen zu.

C.2.3 Klasse c_glimpsearch

Die Klasse c_glimpsearch stellt Methoden zur Volltextsuche in den Dokumenten bereit.

Konstruktor(sparams) - initialisiert Objektvariable, in `sparams` stehen Parameter, die bei jedem Suchaufruf an Glimpse übergeben werden.

glimp_search(searchstr) - führt eine Volltextsuche nach den in `searchstr` gegebene Begriffen mittels Glimpse durch. Der Aufbau des `searchstr` wurde in Kapitel 5.6 beschrieben.

glimp_searchor(searchstr) - führt eine Volltextsuche nach den in `searchstr` gegebene Begriffen mittels Glimpse auf allen als Bitstrom gespeicherten Dokumenten durch. Läßt jedoch nur Oder-Verknüpfungen zu.

C.3 Paket SGML (p_sgml)

Das Paket SGML beinhaltet alle Klassen zur Speicherung von, zum Zugriff auf und zum Durchsuchen der in der Datenbank abgelegten SGML-Dokumente.

C.3.1 Klasse c_sgmlstore

Konstruktor - übernimmt die Dokumenten-ID, sowie den Dateinamen des Eingabebitstroms von der Ablaufumgebung.

open_in - öffnet den Eingabebitstrom.

close_in - schließt den Eingabebitstrom.

read_in(size) - liest `size` Bytes vom Eingabebitstrom.

seek_in(offset,whence) - verändert die Position der Lese-/Schreibmarke des Eingabebitstroms, wie unter **seek** der Klasse **c_bsio** beschrieben.

tell_in() - liefert die aktuelle Position der Lese-/Schreibmarke des Eingabebitstroms.

get_emptytags() - liefert alle in der DTD des Dokuments definierten EMPTY-Tags als Liste zurück.

register_sgmlstore() -vermerkt in der Datenbank, daß das Dokument in der Datenbank des Speichers abgelegt wurde.

save_doctype() - legt den Dokumententyp (DTD-Name) in der Datenbank ab.

save_astag(tagstr) - speichert `tagstr` in der Tag-Klasse, Berechungen für den Strukturbaum werden automatisch vorgenommen.

save_astext(textstr) - speichert `textstr` in der Text-Klasse.

save_asbin(urlstr) - speichert das als URL referenzierte binäre Objekt in der Binärklasse.

get_nextelement() - liest das nächste Element aus dem Eingabebitstrom ein und gibt es zurück.

is_tag(tagstr) - ergibt 1, wenn es sich bei `tagstr` um ein Tag handelt, sonst 0.

is_emptytag(tagstr) - ergibt 1, wenn es sich bei `tagstr` um ein Empty-Tag handelt, sonst 0.

C.3.2 Klasse c_sgmlquery

Diese Klasse bietet Methoden zum Durchsuchen eines bestimmten Dokuments nach Suchausdrücken.

Konstruktor(ident) - initialisert Objektvariable, `ident` gibt die Dokumenten-ID an.

sgmlquery_tag(tagname,tagcount) - liefert das `tagcount`-te Tag mit Namen `tagname` zurück.

sgmlquery_tagcontents(tagname,tagcount) - liefert den Inhalt des `tagcount`-ten Tags mit Namen `tagname` zurück.

sgmlquery_structure() - gibt den Strukturbaum des Dokuments zurück.

sgmlquery_doc() - gibt das gesamte Dokument zurück.

sgmlquery_tagcount(tagname) - zählt die Tags mit Namen `tagname` im Dokument.

sgmlquery_splittag(tagstr) - verarbeitet `tagstr` wie unter **sgmlread_splittag** beschrieben.

C.3.3 Klasse c_sgmlsearch

Diese Klasse bietet Methoden zum Durchsuchen aller Dokumente nach Suchausdrücken. Bei der Suche wird der SQL LIKE-Operator verwendet, so daß auch ähnliche Zeichenketten Berücksichtigung finden.

sgmlsearch_tag(searchstr) - durchsucht die Tag-Klasse nach Übereinstimmungen mit `searchstr`.

sgmlsearch_text(searchstr) - durchsucht die Text-Klasse nach Übereinstimmungen mit `searchstr`.

sgmlsearch_bin() - eine virtuelle Methode, die in abgeleiteten Klassen überschrieben werden kann.

sgmlsearch_doc(searchstr) - durchsucht die Tag- und die Text-Klasse nach Übereinstimmungen mit `searchstr`.

sgmlsearch_dtddocs(dtdid) - ermittelt die Dokumenten-ID's aller Dokumente, die der in `dtdid` gegebenen DTD entsprechen. Dabei kann `dtdid` die Public- oder die System-ID einer DTD angeben. Das Ergebnis wird in Form einer Lsite zurückgeliefert.

sgmlsearch_dtdandtags(dtdid,searchstr) - ermittelt die Dokumenten-ID's aller Dokumente, die der durch `dtdid` gegebenen DTD entsprechen und ein Tag enthalten das `searchstr` entspricht. Das Ergebnis wird in Form einer Liste zurückgegeben.

C.4 Paket System (p_system)

Das Paket System realisiert Klassen zur Verarbeitung und den Zugriff auf Informationen aus den Metadokumenten.

C.4.1 Klasse c_metadoc

Hier werden Methoden zur Verfügung gestellt, die es ermöglichen bestimmte Informationen aus einem als Bitstrom vorliegendem Metadokument auszulesen.

Konstruktor(fname) - initialisiert Objektvariable, `fname` ist der Dateiname des Metadokuments.

get_dtd() - liefert eine Liste des DOCTYPE-Tags und seiner Attribute zurück.

get_header() - gibt den Inhalt des HEADER-Tags zurück.

get_info() - gibt den Inhalt des INFO-Tags zurück.

get_property() - gibt ein Dictionary zurück, das das PROPERTY-Tag und seine Attribute enthält. Die Schlüssel des Dictionaries sind die Namen der Attribute.

get_methods() - gibt ein Dictionary zurück, das die Methoden enthält. Dabei sind die Schlüssel des Dictionaries die Namen der Methoden. Jeder Eintrag besteht aus einem Tupel, das den Namen, den URN und den MIME-Typ der Methode beinhaltet.

get_data() - gibt den Inhalt des DATA-Tags zurück.

C.4.2 Klasse c_metaquery

Zum Zugriff auf die Daten aus dem Metadokument eines bestimmten gespeicherten Dokuments, stellt diese Klasse Methoden bereit.

Konstruktor(ident) - initialisiert Objektvariable, `ident` gibt die Dokumenten-ID an.

mquery_info() - liefert das Info-Tag des korrespondierenden Metadokuments zurück.

mquery_property() - liefert das Property-Tag und seine Attribute als Dictionary zurück dessen Schlüssel die Attributnamen sind.

mquery_methods() - liefert die Methoden des Dokuments als Dictionary zurück dessen Schlüssel die Methodennamen sind.

C.4.3 Klasse c_metasearch

Die Klasse c_metasearch durchsucht alle gespeicherten Metadaten nach Begriffen. Bei der Suche in der Datenbank wird der SQL LIKE-Operator benutzt, so daß auch ähnliche Einträge erkannt werden. Es werden jeweils die gefundenen Dokumenten-ID's in einer Liste zurückgegeben

metasearch_ident(searchstr) - durchsucht das Dokumenten-ID Feld.

metasearch_info(searchstr) - durchsucht das Info-Feld.

metasearch_author(searchstr) - durchsucht das Autor-Feld.

metasearch_owner(searchstr) - durchsucht das Eigentümer-Feld.

metasearch_title(searchstr) - durchsucht das Titel-Feld.

C.5 Paket Maintenance (p_maintenance)

Das Paket Maintenance beinhaltet Klassen, die der Wartung des Speichers dienen.

C.5.1 Klasse c_maintenance

Die Klasse c_maintenance stellt Methoden zur Wartung des Speichers zur Verfügung.

vacuum() - reorganisiert die Datenbank und sollte nur aufgerufen werden, wenn alle Speicher angehalten wurden.

delete_bs(ident) - löscht das Dokument mit der Dokumenten-ID `ident` aus dem Bitstromspeicher.

delete_sgml(ident) - löscht das Dokument mit der Dokumenten-ID `ident` aus der Datenbank.

sql_exec(querystr) - führt die SQL-Anweisung `querystr` auf der Datenbank aus und liefert das Ergebnis als ASCII-Text zurück.

C.5.2 Klasse c_glimpindex

Diese Klasse dient der Verwaltung der von Glimpse erstellten Indizes.

remove_index() - löscht die von Glimpse erstellten Indizes.

build_index() - erstellt neue Glimpse Indizes.

C.6 Paket Netz (p_net)

Das Paket Netz stellt einige Klassen zur Netzwerkkommunikation aus Dokumentenmethoden heraus bereit. Ein solches Szenario wurde in Kapitel 2.4.2 vorgestellt.

C.6.1 c_tcp

Eine Klasse, die Methoden zur TCP-Socketkommunikation bereitstellt.

Konstruktor(host,port) - initialisiert Objektvariable und instanziiert einen Socket, dabei geben `host` und `port` die Adresse des Sockets an. Für port=0 wird automatisch ein freier Port zugewiesen.

connect(host,port) - baut eine Netzverbindung zu der durch `host` und `port` angegebenen Adresse auf.

accept() - wartet auf Verbindungsaufbau.

send(buff) - sendet `buff`.

recv(buffsize) - empfängt maximal `buffsize` Bytes.

poll(buffsize,secs) - wie **recv**, jedoch wird nur `secs` Sekunden auf eine eingehende Nachricht gewartet.

close() - baut eine vorhandene Verbindung ab und schließt den Socket.

getaddr() - gibt die Adresse des lokalen Sockets zurück.

getpeeraddr() - gibt die Adresse des Peers bzw. des entfernten Sockets zurück.

C.6.2 c_stopwait1

Diese Klasse realisiert ein einfaches Stop-Wait-Protokoll zur Übertragung von Dateien und langen Zeichenketten.

Konstruktor(host,port,buffsize,tiemout,debug) - initialisiert Objektvariable und instanziiert einen Socket, dabei geben `host` und `port` die Adresse des Sockets an. `buffsize` gibt die maximale Größe eines Blocks an, `timeout` die Zeit, die verstreichen darf bis eine Antwort vom Peer bzw. der Gegenstelle empfangen wird. Wird für `debug` eine 1 übergeben, werden zusätzliche Meldungen erzeugt, die der Fehlersuche dienen sollen.

connect(host,port) - baut eine Netzverbindung zu der durch `host` und `port` angegebenen Adresse auf.

accept() - wartet auf Verbindungsaufbau.

send_str(sendstr) - sendet `sendstr`.

send_file(fname) - sendet die Datei `fname`.

recv_str() - empfängt eine Zeichenkette.

recv_file(fname) - empfängt die Datei `fname`.

C.6.3 c_send

Die Methoden dieser Klasee ermöglichen es, Teile eines bestimmten Dokuments über eine zuvor aufgebaute Netzverbidung zu versenden.

Konstruktor(ident,host,port,blocksize,timeout) - initialisiert Objektvariable und baut die Netzverbindung auf, dabei ist `ident` die Dokumenten-ID, `host` und `port` die Netzadresse, `blocksize` die maximale Größe eines Übertragunsblocks und `timeout` die Zeit in Sekunden, die auf eine Bestätigung gewartet wird.

send_bs(start,end) - sendet einen Teil des Dokuments, `start` und `end` geben dabei die Byte-Grenzen innerhalb des Dokuments an.

send_tagcontents(tagname,tagcount) - sendet den Inhalt des `tagcount`-ten Tags mit Namen `tagname`.

send_tag(tagname,tagcount) - sendet das `tagcount`-te Tag mit Namen `tagname`.

send_structure() - sendet den Strukturbaum des Dokuments.

send_doc() - sendet das gesamte Dokument.

Anhang D

Quellcode des Pythonmoduls zur Anbindung der PostgreSQL Datenbank

Zur Installation des Python-Moduls zur PostgreSQL-Datenbankanbindung, muß sowohl Python als auch PostgreSQL installiert sein. Innerhalb des Python Sourcetrees, muß in die Datei Modules/Setup.local folgende Zeile eingetragen werden:

```
pgpy pgpylib.o -I<PostgreSQL-Include-Pfad> \
               -L<PostgreSQL-Library-Pfad> -lpq
```

Danach einfach make, der neue Pythoninterpreter erlaubt es dann mittels: `import pgpy` auf das Modul zuzugreifen. Die Methoden der drei Objekte (pgconn, pgres, pgblob) sind den von der PostgreSQL C-Library bereitgestellten Funktionen nachempfunden. Bei Unsicherheiten können Informationen zur jeweiligen Methode über die `.__doc__` Eigenschaft der Methoden abgerufen werden. Beispiele für den Einsatz finden sich in ausreichender Anzahl in Anhang E.

D.1 pgpylib.h

```
/* Python Postgres Interface Library
   (C) by Hans Matzen, 1997, Frankfurt, Germany

*/

/* this is pgpylib.h */
```

```
#include <stdlib.h>
#include "Python.h"
#include "libpq-fe.h"

static PyObject *ErrorObject;

#define Py_Try(BOOLEAN) { if (!(BOOLEAN)) return NULL;}


/* ------------------------------------------------------------- */
/* everything for pgresult, the postgres result handle object     */
/* ------------------------------------------------------------- */

/* the definitions for the python object */
typedef struct {
  PyObject_HEAD
  PGresult *res;
} pgres;

staticforward PyTypeObject pgres_obj;

/* all that help stuff */


static char pgres_obj__doc__[]=
   "Postgres Result Handle Object";

static char pgpy_pgres__doc__[]=
   "pgres(res)=pgres";

static char pgres_get_tuple_count__doc__[]=
   "returns the number of tuples.";
static char pgres_get_field_count__doc__[]=
   "returns the number of fields.";
static char pgres_get_fieldname_byindex__doc__[]=
   "returns the fieldname.";
static char pgres_get_fieldindex_byname__doc__[]=
   "returns the fieldname.";
static char pgres_get_fieldtype__doc__[]=
   "returns the fieldtype.";
static char pgres_get_fieldsize__doc__[]=
   "returns the fieldsize.";
static char pgres_get_fieldvalue__doc__[]=
```

```
   "returns the fieldvalue.";
static char pgres_fieldlength__doc__[]=
   "returns the fieldlength.";
static char pgres_commandstatus__doc__[]=
   "returns the commandstatus.";
static char pgres_objectstatus__doc__[]=
   "returns the objectstatus.";
static char pgres_get_resultlist__doc__[]=
   "prints the query result.";

/* forward declarations for methods */

staticforward pgres   * new_pgres(pgres *init);
staticforward void      free_pgres(pgres *self);
staticforward
   PyObject* pgpy_pgres(PyObject *self,PyObject *args);
staticforward
   PyObject* pgres_getattr(pgres *self,char *name);
staticforward PyObject* pgres_get_tuple_count(pgres *self);
staticforward PyObject* pgres_get_field_count(pgres *self);
staticforward
   PyObject* pgres_get_fieldname_byindex(pgres *self,PyObject *args);
staticforward
   PyObject* pgres_get_fieldindex_byname(pgres *self, PyObject *args);
staticforward
   PyObject* pgres_get_fieldtype(pgres *self, PyObject *args);
staticforward
   PyObject* pgres_get_fieldsize(pgres *self,PyObject *args);
staticforward
   PyObject* pgres_get_fieldvalue(pgres *self,PyObject *args);
staticforward
   PyObject* pgres_get_fieldlength(pgres *self, PyObject * args);
staticforward PyObject* pgres_commandstatus(pgres *self);
staticforward PyObject* pgres_objectstatus(pgres *self);
staticforward
   PyObject* pgres_get_resultlist(pgres *self,PyObject *args);


/* method table for class pgres */

static struct PyMethodDef pgres_methods[]={
  {"get_tuple_count"      ,(PyCFunction)pgres_get_tuple_count       },
  {"get_field_count"      ,(PyCFunction)pgres_get_field_count       },
  {"get_fieldname_byindex",(PyCFunction)pgres_get_fieldname_byindex },
  {"get_fieldindex_byname",(PyCFunction)pgres_get_fieldindex_byname },
```

```
  {"get_fieldtype"        ,(PyCFunction)pgres_get_fieldtype        },
  {"get_fieldsize"        ,(PyCFunction)pgres_get_fieldsize        },
  {"get_fieldvalue"       ,(PyCFunction)pgres_get_fieldvalue       },
  {"get_fieldlength"      ,(PyCFunction)pgres_get_fieldlength      },
  {"commandstatus"        ,(PyCFunction)pgres_commandstatus        },
  {"objectstatus"         ,(PyCFunction)pgres_objectstatus         },
  {"get_resultlist"       ,(PyCFunction)pgres_get_resultlist       },
  {NULL,NULL}
};


/* pgres object table  */

static PyTypeObject pgres_obj= {
  PyObject_HEAD_INIT(&PyType_Type) 0,
  "pgres",
  sizeof(pgres),
  0,
  (destructor)free_pgres,
  (printfunc)0,
  (getattrfunc)pgres_getattr,
  (setattrfunc)0,
  (cmpfunc)0,
  (reprfunc)0,
  0,0,0,
  (hashfunc)0,
  (ternaryfunc)0,
  (reprfunc)0,
  0L,0L,0L,0L,
  pgres_obj__doc__
};


/* ------------------------------------------------------------- */
/* everything for pgconn, the postgres connection object         */
/* ------------------------------------------------------------- */

typedef struct {
  PyObject_HEAD
  PGconn *conn;
  int trace;
} pgconn;

staticforward PyTypeObject pgconn_obj;
```

```
/* all that help stuff */

static char pgpy_module_documentation[]=
   "Postgres95 Python Interface";

static char pgconn_obj__doc__[]=
   "Postgres95 Connection Object";

static char pgpy_pgconn__doc__[]=
   "pgconn(host,port,dbname) = pgconn";

static char pgconn_get_dbname__doc__[]=
   "returns the name of the database you are connected to.";
static char pgconn_get_host__doc__[]=
   "returns the hostname you are connected to.";
static char pgconn_get_port__doc__[]=
   "returns the port number you are connected to.";
static char pgconn_get_options__doc__[]=
   "returns the options which were passed to the postmaster.";
static char pgconn_get_tty__doc__[]=
   "returns the tty where the postmaster logs to.";
static char pgconn_connstatus__doc__[]=
   "returns 0 if the connectio is ok., or 1 otherwise.";
static char pgconn_get_error__doc__[]=
   "returns the error that occures last.";
static char pgconn_connreset__doc__[]=
   "reset the connection to the postmaster.";
static char pgconn_switch_trace__doc__[]=
   "switch trace mode on or off.";
static char pgconn_exec_query__doc__[]=
   "executes the query passed as parameter.";


/* forward declarations for methodes */

staticforward
   pgconn  * new_pgconn(char *host, char *port, char* dbname);
staticforward void      free_pgconn(pgconn *self);
staticforward
   PyObject* pgpy_pgconn(PyObject *self,PyObject *args);
staticforward PyObject* pgconn_getattr(pgconn *self,char *name);
staticforward PyObject* pgconn_get_dbname(pgconn *self);
staticforward PyObject* pgconn_get_host(pgconn *self);
staticforward PyObject* pgconn_get_port(pgconn *self);
staticforward PyObject* pgconn_get_options(pgconn *self);
```

```
staticforward PyObject* pgconn_get_tty(pgconn *self);
staticforward PyObject* pgconn_get_error(pgconn *self);
staticforward PyObject* pgconn_connstatus(pgconn *self);
staticforward
   PyObject* pgconn_exec_query(pgconn *self,PyObject *args);
staticforward PyObject* pgconn_switch_trace(pgconn *self);
staticforward PyObject* pgconn_connreset(pgconn *self);


/* method table for class pgconn */

static struct PyMethodDef pgconn_methods[]={
  {"get_dbname"  ,(PyCFunction)pgconn_get_dbname  },
  {"get_host"    ,(PyCFunction)pgconn_get_host    },
  {"get_port"    ,(PyCFunction)pgconn_get_port    },
  {"get_options" ,(PyCFunction)pgconn_get_options },
  {"get_tty"     ,(PyCFunction)pgconn_get_tty     },
  {"connstatus"  ,(PyCFunction)pgconn_connstatus  },
  {"get_error"   ,(PyCFunction)pgconn_get_error   },
  {"connreset"   ,(PyCFunction)pgconn_connreset   },
  {"switch_trace",(PyCFunction)pgconn_switch_trace},
  {"exec_query"  ,(PyCFunction)pgconn_exec_query  },
  {NULL,NULL}
};


/* pgconn object table  */

static PyTypeObject pgconn_obj= {
  PyObject_HEAD_INIT(&PyType_Type) 0,
  "pgconn",
  sizeof(pgconn),
  0,
  (destructor)free_pgconn,
  (printfunc)0,
  (getattrfunc)pgconn_getattr,
  (setattrfunc)0,
  (cmpfunc)0,
  (reprfunc)0,
  0,0,0,
  (hashfunc)0,
  (ternaryfunc)0,
  (reprfunc)0,
  0L,0L,0L,0L,
  pgconn_obj__doc__
```

```
};



/* ------------------------------------------------------------------ */
/* everything for pgblob, the postgres binary large object interface  */
/* ------------------------------------------------------------------ */

typedef struct {
  PyObject_HEAD
  pgconn *conn;
        int fd;
        Oid blobid;
        char *buff;
} pgblob;

staticforward PyTypeObject pgblob_obj;

/* all that help stuff */


static char pgblob_obj__doc__[]=
   "Postgres95 Binary Large Object";

static char pgpy_pgblob__doc__[]=
   "pgblob(pgconn)=pgblob";

static char pgblob_bopen__doc__[]=
   "open BLOB.";
static char pgblob_bclose__doc__[]=
   "close BLOB.";
static char pgblob_bread__doc__[]=
   "read from BLOB.";
static char pgblob_bwrite__doc__[]=
   "write to BLOB.";
static char pgblob_bseek__doc__[]=
   "seek position in BLOB.";
static char pgblob_btell__doc__[]=
   "tells current position in BLOB.";
static char pgblob_bimport__doc__[]=
   "import from file to BLOB.";
static char pgblob_bexport__doc__[]=
   "export from BLOB to file.";
static char pgblob_bcreate__doc__[]=
   "create new BLOB.";
```

```
static char pgblob_bunlink__doc__[]=
   "unlink from BLOB Oid.";


/* forward declarations for methodes */

staticforward pgblob  * new_pgblob(pgconn *init);
staticforward void      free_pgblob(pgblob *self);
staticforward
   PyObject* pgpy_pgblob(PyObject *self,PyObject *args);
staticforward PyObject* pgblob_getattr(pgblob *self,char *name);
staticforward
   PyObject* pgblob_bopen(pgblob *self,PyObject *args);
staticforward PyObject* pgblob_bclose(pgblob *self);
staticforward
   PyObject* pgblob_bread(pgblob *self,PyObject *args);
staticforward
   PyObject* pgblob_bwrite(pgblob *self, PyObject *args);
staticforward
   PyObject* pgblob_bseek(pgblob *self, PyObject *args);
staticforward
   PyObject* pgblob_btell(pgblob *self);
staticforward
   PyObject* pgblob_bimport(pgblob *self,PyObject *args);
staticforward
   PyObject* pgblob_bexport(pgblob *self, PyObject *args);
staticforward
   PyObject* pgblob_bcreate(pgblob *self, PyObject *args);
staticforward PyObject* pgblob_bunlink(pgblob *self);



/* method table for class pgblob */

static struct PyMethodDef pgblob_methods[]={
  {"bopen"   ,(PyCFunction)pgblob_bopen   },
  {"bclose"  ,(PyCFunction)pgblob_bclose  },
  {"bread"   ,(PyCFunction)pgblob_bread   },
  {"bwrite"  ,(PyCFunction)pgblob_bwrite  },
  {"bseek"   ,(PyCFunction)pgblob_bseek   },
  {"btell"   ,(PyCFunction)pgblob_btell   },
  {"bimport" ,(PyCFunction)pgblob_bimport },
  {"bexport" ,(PyCFunction)pgblob_bexport },
  {"bcreate" ,(PyCFunction)pgblob_bcreate },
  {"bunlink" ,(PyCFunction)pgblob_bunlink },
```

```
  {NULL,NULL}
};


/* pgblob object table  */

static PyTypeObject pgblob_obj= {
  PyObject_HEAD_INIT(&PyType_Type) 0,
  "pgblob",
  sizeof(pgblob),
  0,
  (destructor)free_pgblob,
  (printfunc)0,
  (getattrfunc)pgblob_getattr,
  (setattrfunc)0,
  (cmpfunc)0,
  (reprfunc)0,
  0,0,0,
  (hashfunc)0,
  (ternaryfunc)0,
  (reprfunc)0,
  0L,0L,0L,0L,
  pgblob_obj__doc__
};


/* pgpylib init table, lets stick it together */

static struct PyMethodDef pgpy_methods[]={
  {"pgconn",(PyCFunction)pgpy_pgconn,1,pgpy_pgconn__doc__},
  {"pgres" ,(PyCFunction)pgpy_pgres,1,pgpy_pgres__doc__},
  {"pgblob",(PyCFunction)pgpy_pgblob,1,pgpy_pgblob__doc__},
  {NULL,NULL}
};
```

D.2 pgpylib.c

```
/* Python Postgres Interface Library
   (C) by Hans Matzen, 1997, Frankfurt, Germany

*/

/* this is pgpylib.c */

#include <stdio.h>
#include "pgpylib.h"

/* pgres constructor and destructor */

static pgres *new_pgres(pgres *init) {
  pgres *self;

  Py_Try((self=PyObject_NEW(pgres, &pgres_obj)));
  self->res=init->res;

  return self;
}

static void free_pgres(pgres *self) {
  PQclear(self->res);
  PyMem_DEL(self);
}

/* pgres methods */

static PyObject* pgres_getattr(pgres *self,char *name) {
  return Py_FindMethod(pgres_methods, (PyObject *) self,name);
}

/* returns the number of records in the result */
static PyObject* pgres_get_tuple_count(pgres *self) {
  int result;

  result=PQntuples(self->res);
  return Py_BuildValue("i",result);
 }

/* returns the number of fileds per record in the result */
static PyObject* pgres_get_field_count(pgres *self) {
  int result;
```

```
  result=PQnfields(self->res);
  return Py_BuildValue("i",result);
}

/* returns fieldname in column fieldindex */
static PyObject*
   pgres_get_fieldname_byindex(pgres *self,PyObject *args) {
  char *result;
  int fieldindex;

  Py_Try(PyArg_Parse(args,"i",&fieldindex));
  result=PQfname(self->res,fieldindex);
  return Py_BuildValue("s",result);
}

/* returns column number of fieldname */
static PyObject*
   pgres_get_fieldindex_byname(pgres *self, PyObject *args) {
  int result;
  char *fieldname;

  Py_Try(PyArg_Parse(args,"s",&fieldname));
  result= PQfnumber(self->res,fieldname);
  return Py_BuildValue("i",result);
}

/*  returns type of field with index fieldindex */
static PyObject*
   pgres_get_fieldtype(pgres *self, PyObject *args) {
  int result;
  int fieldnum;

  Py_Try(PyArg_Parse(args,"i",&fieldnum));
  result=PQftype(self->res,fieldnum);
  return Py_BuildValue("i",result);
}

/*  returns size i bytes of field with index fieldindex */
static PyObject*
   pgres_get_fieldsize(pgres *self, PyObject *args) {
  int result;
  int fieldindex;

  Py_Try(PyArg_Parse(args,"i",&fieldindex));
```

```
  result=PQfsize(self->res,fieldindex);
  return Py_BuildValue("i",result);
}

/* returns the value of field with index fieldnum
   in record tupnum     */
static PyObject*
   pgres_get_fieldvalue(pgres *self, PyObject *args) {
  char *result;
  int tupnum;
  int fieldnum;

  Py_Try(PyArg_ParseTuple(args,"ii",&tupnum,&fieldnum));
  result=PQgetvalue(self->res,tupnum,fieldnum);
  return Py_BuildValue("s",result);
}

/* returns length in bytes of value in field given
   by fieldnum, tupnum     */
static PyObject*
   pgres_get_fieldlength(pgres *self,PyObject *args) {
  int result;
  int tupnum, fieldnum;

  Py_Try(PyArg_ParseTuple(args,"ii",&tupnum,&fieldnum));
  result=PQgetlength(self->res,tupnum,fieldnum);
  return Py_BuildValue("i",result);
}

/* returns the returnvalue from the databaseserver */
static PyObject* pgres_commandstatus(pgres *self) {
  char *result;

  result=PQcmdStatus(self->res);
  return Py_BuildValue("s",result);
}

/* returns the object status of the resultobject
   helpful for errorchecking  */
static PyObject* pgres_objectstatus(pgres *self) {
 char *result;

  result=PQoidStatus(self->res);
  return Py_BuildValue("s",result);
}
```

```
/* returns an ASCII table of the whole resukt matrix */
static PyObject*
   pgres_get_resultlist(pgres *self,PyObject *args) {

  int fillAllign;
  char * fieldSep;
  int printHeader;
  int quiet;

  Py_Try(PyArg_ParseTuple(args,"isii",&fillAllign,&fieldSep,
     &printHeader,&quiet));
  PQdisplayTuples(self->res,NULL,fillAllign,
     fieldSep,printHeader,quiet);

  return Py_BuildValue("s","EOL");
}



/* pgconn constructor and destructor */

static pgconn *
   new_pgconn( char *host, char *port, char* dbname) {

  pgconn *self;

  Py_Try((self=PyObject_NEW(pgconn, &pgconn_obj)));

  self->trace=0;
  self->conn=PQsetdb(host,port,getenv("PGOPTIONS"),
     getenv("PGTTY"),dbname);

  return self;

}

static void free_pgconn(pgconn *self) {
  PQfinish(self->conn);
  PyMem_DEL(self);
}



/* pgconn methods */
```

```
static char pgconn_getdbname__doc__[]="getdbname(pgconn)";

/* returns the database name /*
static PyObject* pgconn_get_dbname(pgconn *self) {
  char *result;

  result=PQdb(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the hostname of the databaseserver */
static PyObject* pgconn_get_host(pgconn *self) {
  char *result;

  result =PQhost(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the portnumber of the databaseserver */
static PyObject* pgconn_get_port(pgconn *self) {
  char *result;

  result =PQport(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the options set for the connection */
static PyObject* pgconn_get_options(pgconn *self) {
  char *result;

  result =PQoptions(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the tty */
static PyObject* pgconn_get_tty(pgconn *self) {
  char *result;

  result =PQtty(self->conn);
  return Py_BuildValue("s",result);
}

/* returns the connectionstatus
   helpful for errorchekcing */
```

```
static PyObject* pgconn_connstatus(pgconn *self) {
  int result;

  result =PQstatus(self->conn);
  return Py_BuildValue("i",result);
}

/* if an error occured, this gets the errormessage
   from the server    */
static PyObject* pgconn_get_error(pgconn *self) {
  char *result;

  result =PQerrorMessage(self->conn);
  return Py_BuildValue("s",result);
}

/* switches tracemode on/off */
static PyObject* pgconn_switch_trace(pgconn *self) {
  if (self->trace == 1) {
    PQuntrace(self->conn);
    self->trace=0;
  } else {
    PQtrace(self->conn,stdout);
    self->trace=1;
  }

  return Py_BuildValue("i",self->trace);
}

/* resets the connection */
static PyObject* pgconn_connreset(pgconn *self) {

  PQreset(self->conn);
  return Py_BuildValue("i",0);
}

/* executes the query given by qu
   returns a pgres objecthandle   */
static PyObject*
   pgconn_exec_query(pgconn *self, PyObject *args) {

  PyObject *result;
  pgres *reshandle=malloc(sizeof(pgres)+1);
  char *qu;
```

```
  Py_Try(PyArg_Parse(args,"s",&qu));
  reshandle->res = PQexec(self->conn,qu);
  result= (PyObject *) new_pgres(reshandle);
  return result;
}

/* neede by python,  to resolve methodcalls */
static PyObject *pgconn_getattr(pgconn *self, char *name) {
  if (strcmp(name,"trace")==0) {
    return Py_BuildValue("i",self->trace);
  } else {
    return Py_FindMethod(pgconn_methods,
       (PyObject *) self,name);
  }
}


/* pgblob constructor and destructor */

static pgblob *new_pgblob( pgconn *init) {

  pgblob *self;

  Py_Try((self=PyObject_NEW(pgblob, &pgblob_obj)));

  self->blobid=0;
  self->fd=0;
  self->buff=malloc(1030);
  self->conn=init;
  Py_INCREF(init);


  return self;

}

static void free_pgblob( pgblob *self) {
    free(self->buff);
    Py_DECREF(self->conn);
    PyMem_DEL(self);
}



/* pgblob methods */
```

```
/* opens a known blob */
static PyObject* pgblob_bopen(pgblob *self, PyObject *args) {
  int result;
  int mode;

  Py_Try(PyArg_Parse(args,"i",&mode));
  result=lo_open(self->conn->conn,self->blobid,mode);
  self->fd=result;
  return Py_BuildValue("i", result);
}


/* closes a blob */
static PyObject* pgblob_bclose(pgblob *self) {
  int result;

  result =lo_close(self->conn->conn,self->fd);
  return Py_BuildValue("i",result);
}

/* reads len bytes from blob and puts them into buff */
static PyObject* pgblob_bread(pgblob *self,PyObject *args) {
  int result;
  int len;


  Py_Try(PyArg_Parse(args,("i"),&len));
  result =lo_read(self->conn->conn,self->fd,self->buff,len);

  return Py_BuildValue("i",result);
}

/* writes buff to blob returns number of written bytes */
static PyObject*
   pgblob_bwrite(pgblob *self,PyObject *args) {
  int result;
  char *buff;
  int len;

  Py_Try(PyArg_ParseTuple(args,("si"),&buff,&len));

  result =lo_write(self->conn->conn,self->fd,buff,len);

  return Py_BuildValue("i",result);
```

```
}

/* changes the position of the filepointer */
static PyObject* pgblob_bseek(pgblob *self,PyObject *args) {
  int result;
  int offset;
  int whence;

  Py_Try(PyArg_ParseTuple(args,"ii",&offset,&whence));
  result =lo_lseek(self->conn->conn,self->fd,offset,whence);
  return Py_BuildValue("i",result);
}

/* returns the position of the filepointer */
static PyObject* pgblob_btell(pgblob *self) {
  int result;

  result =lo_tell(self->conn->conn,self->fd);
  return Py_BuildValue("i",result);
}

/* export the whole blob to file fname
   be careful using this sometimes causes trouble
   with postgres   */
static PyObject*
   pgblob_bexport(pgblob *self,PyObject *args) {
  int result;
  char *fname;

  Py_Try(PyArg_Parse(args,"s",&fname));
  PQexec(self->conn->conn,"begin ");
  result =lo_export(self->conn->conn,self->blobid,fname);
  PQexec(self->conn->conn,"end;");
  return Py_BuildValue("i",result);
}


/* imports the file given by fname as blob into
   postgres, returns the blob-id */
static PyObject*
   pgblob_bimport(pgblob *self,PyObject *args) {
  Oid result;
  char *fname;

  Py_Try(PyArg_Parse(args,"s",&fname));
```

```
  PQexec(self->conn->conn,"begin ");
  result =lo_import(self->conn->conn,fname);
  PQexec(self->conn->conn,"end;");
  self->blobid=result;
  return Py_BuildValue("i",(unsigned int)result);
}


/* creates a new blob and returns the blob-id  */
static PyObject* pgblob_bcreate(pgblob *self,PyObject *args) {
    int mode;
    Oid result;

    Py_Try(PyArg_Parse(args,"i",&mode));
    result=lo_creat(self->conn->conn,mode);
    self->blobid=result;
    return Py_BuildValue("i",(unsigned int) result);
}


/* removes the blob given by blobid from the database */
static PyObject* pgblob_bunlink(pgblob *self) {
  int result;

  result=lo_unlink(self->conn->conn,self->blobid);
  return Py_BuildValue("i",result);
}

/* this is for python, for resolving methodcalls  */
static PyObject *pgblob_getattr(pgblob *self, char *name) {
  if (strcmp(name,"fd")==0) {
      return Py_BuildValue("i",self->fd);
  } else if (strcmp(name,"blobid")==0) {
      return Py_BuildValue("i",self->blobid);
  } else if (strcmp(name,"buff")==0) {
      return Py_BuildValue("s",self->buff);
  } else {
      return Py_FindMethod(pgblob_methods,
(PyObject *) self,name);
  }
}
```

```
/* ------------------------------------------------------- */
/* pgpy module methods                                     */
/* ------------------------------------------------------- */

static PyObject* pgpy_pgres(PyObject *self,PyObject *args) {

  PyObject *result;
  pgres *p;

  Py_Try(PyArg_ParseTuple(args,"O",p));
  result = (PyObject *) new_pgres(p);

  return result;
}

static PyObject* pgpy_pgconn(PyObject *self,PyObject *args) {

  PyObject *result;
  char *host, *port, *dbname;

  Py_Try(PyArg_ParseTuple(args,"sss",&host,&port,&dbname));
  result = (PyObject *) new_pgconn(host,port,dbname);

  return result;
}

static PyObject* pgpy_pgblob(PyObject *self,PyObject *args) {

  PyObject *result;
  pgconn *c;


  Py_Try(PyArg_ParseTuple(args,"O",&c));
  result = (PyObject *) new_pgblob(c);

  return result;
}


/* ------------------------------------------------------- */
/* module pgpy init                                        */
/* ------------------------------------------------------- */
```

```
void initpgpy() {
  PyObject *m, *d;

/* create module and add functions */
  m=Py_InitModule4("pgpy",pgpy_methods,
pgpy_module_documentation,(PyObject*)NULL,PYTHON_API_VERSION);

/* these are the symbols for error reporting */
d= PyModule_GetDict(m);
ErrorObject=PyString_FromString("pgpy.error");
PyDict_SetItemString(d,"error",ErrorObject);

/* any errors ?? */

if (PyErr_Occurred()) {
  Py_FatalError("unable to initialize module pgpy");
}

}
```

Anhang E

Quellcode des Prototyps

E.1 Schnittstellenkomponente

E.1.1 Beispiel für Konfigurationsdatei

```
# --------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname       : interface.conf
# Datum           : 03.11.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Konfigurationsdatei des Schnittstellenelements
#
# Anmerkungen     :
# --------------------------------------------------------------

# Portnummer fuer eingehende Auftraege
# Syntax: RECEIVE_PORT <portnr>
#
RECEIVE_PORT 6301

# Adressen der Broker
# Syntax: BROKER_ADDRESS <host>,<port>[,<timeout>]
#
BROKER_ADDRESS localhost,6401
```

```
# Python Modul in dem die Filter Routinen stehen
# Syntax: MAP_MODULE <modulname>,<modulname>,...
#
MAP_MODULE filters

# Eintraege, die einem Kommando einen Filter zuordnen
# Syntax: MAP_COMMAND <command>,<name of filterfunction>
#
MAP_COMMAND STORE,store_filter
MAP_COMMAND ACTIVATE,dummyfilter
MAP_COMMAND INFO,dummyfilter

# maximale Anzahl der offenen Auftraege im Schnittstellenobjekt
# Syntax: MAX_ORDERS <count>
#
MAX_ORDERS 10

# Logdatei
# Syntax: LOG_FILE <filename>
#
LOG_FILE ../log/ifobject.log
```

E.1.2 Hauptprogramm der Schnittstellenkomponente

```
#!/usr/users2/diplom/hans/python/python/bin/python
# --------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : ifobject.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Dieses Programm realisiert die
#                    Schnittstellenkomponente des Speichersystems
#
# Anmerkungen      :
#
# --------------------------------------------------------------

# globale Variable
CONFIG_FILE="interface.conf"

# imports
import sys
import string
import signal

# Klasse zum parsen der Konfigurationsdatei
import confparse
# Klasse zur Verwaltung einer Socketmenge
import c_multiconn
# Klasse um in ein Logfile zu schreiben
import c_log
# Klasse zur Auftragsverwaltung (Zuordnung Socket <-> Auftrag)
import c_orgalists


# Verarbeitung  Kommandozeilenparameter
# pruefe parameter
if len(sys.argv)<2:
      print "Usage: ifobject.py <name> [-f <configfile>]"
      sys.exit(1)

# hole Kommandozeilenparameter
```

```
ifobjname=sys.argv[1]
try:
      CONFIG_FILE=sys.argv[3]
except:
      pass

# installiere Signalhandler
doit=1

def sig_int(a,b):
      global doit
      # Abbruchbedingung fuer Mainloop
      doit=0

signal.signal(signal.SIGINT,sig_int)



# instanziiere Socketmengenobjekt
conn=c_multiconn.c_multiconn()

# Verarbeite Konfigurationsfile
params=confparse.confparse(CONFIG_FILE)

brlist=[]
rcvlist=[]
modlist=[]
mapdict={}
lfile="./if.log"
i=0

# durchlaufe alle Parameter und speichere sie in
# der dafür vorgesehenen Liste
while i<len(params):
      el=params[i]
      if el[0]=="RECEIVE_PORT":
        rport=eval(el[1])
        rcvlist.append(conn.open_sock("",rport))
      elif el[0]=="BROKER_ADDRESS":
        baddr=string.split(el[1],",")
        brlist.append(baddr)
      elif el[0]=="MAP_MODULE":
        mmod=el[1]
        modlist=string.split(mmod,",")
      elif el[0]=="MAP_COMMAND":
```

```
        map=string.split(el[1],",")
        mapdict[map[0]]=map[1]
      elif el[0]=="MAX_ORDERS":
        maxorders=eval(el[1])
      elif el[0]=="LOG_FILE":
        lfile=el[1]

      i=i+1


# Oeffne Logfile
log=c_log.c_log("Schnittstelle "+ifobjname,lfile)

log.log("Gestartet")

# init. Variablen fuer Auftragslistenverwaltung
# instanziiere Speicherliste und Jobliste
rcv_connlist=[]
jobs=c_orgalists.c_joblist()

# Schnittstellenelement-Sockets vorbereiten
i=0
log.log("Verbindung fuer Klienten unter:")
while (i<len(rcvlist)):
      log.log(str(conn.get_localaddr(rcvlist[i])))
      conn.listen(rcvlist[i])
      i=i+1

# ----------------------------------------------------------------
# Main Loop
# ----------------------------------------------------------------
doit=1
while doit==1:
      log.log("Warte auf Auftraege")
      active_socks=[]
      # Warte auf Auftraege
      active_socks=conn.wait_event()

      # Ereignisse abarbeiten
      i=0
      while i<len(active_socks):
        actsock=active_socks[i]

        if actsock in rcvlist:
          log.log("Ein Klient bittet um Verbindung.")
```

```
  # Klient baut verbindung auf
  newfd=conn.accept(actsock)
  rcv_connlist.append(newfd)
elif actsock in rcv_connlist:
  # Nachricht ueber bestehende Verbindung
  log.log("Ein␣Klient␣bittet␣um␣Gehoer.")
  msg=""
  msg=conn.read_sock(actsock)
  if msg=="":
    log.log("Oops,␣der␣hat␣einfach␣aufgelegt␣["+
     str(actsock)+"].")
    conn.close_sock(actsock)
    del rcv_connlist[rcv_connlist.index(actsock)]
  else:
    # Verarbeitung des auftrages
    msg_parts=string.split(msg,"␣")
    # Nachricht filtern/konvertieren
    if mapdict.has_key(msg_parts[0]):
      # aufruf der filterroutine
      statement="import␣"+modlist[0]
      exec statement
      statement="msgnew="+modlist[0]+"."+
       mapdict[msg_parts[0]]+"(msg)"
      exec statement

    #verbindung zum broker aufbauen und auftrag
    # abschicken
    newfd=conn.open_sock("",0)
    conn.connect(newfd,brlist[0][0],
      eval(brlist[0][1]))
    conn.write_sock(newfd,msgnew)
    # und Auftrag in Auftragsliste speichern
    jobs.addjob(actsock,"c",0,newfd,"c")

else:
  # Nachricht ueber bestehende Verbindung
  log.log("Ein␣Broker␣bittet␣um␣Gehoer.")
  msg=conn.read_sock(actsock)
  if msg=="":
    log.log("Oops,␣der␣hat␣einfach␣aufgelegt␣["+
     str(actsock)+"].")
    conn.close_sock(actsock)
    del rcv_connlist[rcv_connlist.index(actsock)]
  else:
    # zuordnung der antwort zum klienten
```

```
            clifd=jobs.get_job_by_repfd(actsock)
            tuple=jobs.get_jobdata(clifd)
            # job aus liste loeschen
            jobs.remjob(clifd)
            # verbindung zum broker abbauen
            conn.close_sock(actsock)
            # antwort an client schicken
            conn.write_sock(tuple[2],msg)
            # verbindung zum klienten beenden
            conn.close_sock(tuple[2])
            # verbindung aus liste loeschen
            del rcv_connlist[rcv_connlist.index(tuple[2])]

        i=i+1

# löschen des Verbindungsobjekts
del conn
```

E.1.3 Filterroutinen

```
# ----------------------------------------------------------------
# Projekt           : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1997
# Dateiname         : filter.py
# Datum             : 03.11.1997
# letzte Änderung :
# Autor             : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache           : Python v1.4
# Beschreibung      : Hier werden Funktionen definiert, die die
#                     eingehenden Aufträge an das interne Protokoll
#                     des Speichersystems anpassen.
# ----------------------------------------------------------------


# ----------------------------------------------------------------
# Name              : dummyfilter
# Datum             : 03.11.1997
# letzte Änderung :
# Typ               : Funktion
# Argumente         : msg - ein Textstring
# Rückgabewert      : msg - ein Textstring
# ----------------------------------------------------------------
# Beschreibung      : Da diese Funktion nur einen Dummyfilter
#                     realisiert, wird lediglich das übergebene
#                     Argument unverändert zurückgegeben.
# ----------------------------------------------------------------

def dummyfilter(msg):
        return msg



# ----------------------------------------------------------------
# Name              : store_filter
# Datum             : 03.11.1997
# letzte Änderung :
# Typ               : Funktion
# Argumente         : msg - ein Textstring
# Rückgabewert      : msg - ein Textstring
# ----------------------------------------------------------------
```

```
# Beschreibung    : Ersetzt das STORE Kommando durch ein
#                   PUT Kommando
#                   Diese Routine dient nur Demonstrations-
#                   und Testzwecken
# -------------------------------------------------------------

def store_filter(msg):
      return "PUT"+ msg[5:]
```

E.2 Broker

E.2.1 Beispiel für Konfigurationsdatei

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : broker.conf
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Konfigurationsdatei fuer Broker
# Anmerkungen      :
# -------------------------------------------------------------
# Portnummer fuer eingehende Auftraege von Schnittstellenobjekten
# Syntax: RECEIVE_PORT <portnr>
#
RECEIVE_PORT 6401

# Portnummer fuer Verbindungsaufnahme der Speicher
# CONNECT_PORT <portnr>
#
CONNECT_PORT 6433

# Portnummer fuer Monitor/Steurungswerkzeug
# Syntax: CONTROL_PORT <portnr>
#
CONTROL_PORT 6444

# maximale Anzahl der offenen Auftraege im Broker
# Syntax: MAX_ORDERS <count>
#
MAX_ORDERS 50

# Logdatei
# Syntax: LOG_FILE <filename>
#
LOG_FILE ./../log/broker.log
```

E.2.2 Hauptprogramm des Brokers

```
#!/usr/users2/diplom/hans/python/python/bin/python
# ----------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : broker.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hauptprogramm zum Broker
# Anmerkungen      :
#
# ----------------------------------------------------------------

# globale Variable
CONFIG_FILE="broker.conf"

# imports
import sys
import string
import signal
import confparse
import c_multiconn
import c_log


#
# Hilfsfunktionen
#
# Loescht das Element mit dem Wert x aus der Liste Y
def dellist(y,x):
      erg=[]
      for i in range(len(y)):
        if y[i]!=x:
          erg.append(y[i])
      return erg


# Verarbeitung  Kommandozeilenparameter
# pruefe parameter
if len(sys.argv)!=2:
```

```
    print "Usage:␣broker.py␣<name>␣[-f␣<configfile>]"
    sys.exit(1)

# hole Kommandozeilenparameter und setzte Statusinformationen
# des Brokers
brokername=sys.argv[1]
brokerstate="run"
brokerqueue=0
brokercounter=0
# Wie wird verteilt automatisch=1, manuell=0
brokerdistmethod=1
# installiere Signalhandler
doit=1
def sighup(a,b):
    global brokerstate
    if brokerstate=="run":
      brokerstate="paused"
    else:
      brokerstate="run"

def sigint(a,b):
    global doit
    doit=0


signal.signal(signal.SIGHUP ,sighup)
signal.signal(signal.SIGINT ,sigint)

# instanziiere Socketmengenobjekt
conn=c_multiconn.c_multiconn()

# Verarbeite Konfigurationsfile
params=confparse.confparse(CONFIG_FILE)

# setze defaultwerte
maxorders=10
ifobj_connlist=[]
ctrl_connlist=[]
rep_connlist=[]
lfile="./broker.log"
i=0

# setzte Parameter aus Konfigurationsfile
while i<len(params):
    el=params[i]
```

```
      if el[0]=="RECEIVE_PORT":
        rport=eval(el[1])
        ifobj_connlist.append(conn.open_sock("",rport))
      elif el[0]=="CONNECT_PORT":
        cport=eval(el[1])
        rep_connlist.append(conn.open_sock("",cport))
      elif el[0]=="CONTROL_PORT":
        ctrlport=eval(el[1])
        ctrl_connlist.append(conn.open_sock("",ctrlport))
      elif el[0]=="MAX_ORDERS":
        maxorders=eval(el[1])
      elif el[0]=="LOG_FILE":
        lfile=el[1]

      i=i+1

# Oeffne Logfile
log=c_log.c_log("Broker␣"+str(brokername),lfile)
log.log("Gestartet")

# init. Variablen fuer Auftragslistenverwaltung
ifobj_list=[]
ctrl_list=[]
rep_list=[]


# instanziiere Speicherliste und Jobliste
import c_orgalists
connected_reps=c_orgalists.c_replist()
jobs=c_orgalists.c_joblist()

# initialisere Sockets
# Schnittstellenkomponenten-Sockets vorbereiten
i=0
log.log("Verbindung␣fuer␣Schnittstellenobjekte␣unter:")
while (i<len(ifobj_connlist)):
      log.log(str(conn.get_localaddr(ifobj_connlist[i])))
      conn.listen(ifobj_connlist[i])
      i=i+1

# Speicher-Sockets vorbereiten
i=0
log.log("Verbindung␣fuer␣Speicher␣unter:")
while (i<len(rep_connlist)):
      log.log(str(conn.get_localaddr(rep_connlist[i])))
```

```
      conn.listen(rep_connlist[i])
      i=i+1

# Steuerungs-Sockets vorbereiten
i=0
log.log("Verbindung fuer Managementobjekte unter:")
while (i<len(ctrl_connlist)):
      log.log(str(conn.get_localaddr(ctrl_connlist[i])))
      conn.listen(ctrl_connlist[i])
      i=i+1

# ------------------------------------------------------------
# Mainloop
# ------------------------------------------------------------
doit=1
while doit==1:
      log.log("Warte auf Auftraege")
      active_socks=[]
      # auf Ereignissen an Sockets warten
      active_socks=conn.wait_event()

      # ereignisse abarbeiten
      i=0
      while i<len(active_socks):
        actsock=active_socks[i]

        # ist der broker angehalten ?
        if brokerstate=="paused":
          for h in active_socks:
            try:

                conn.write_sock("Der Broker ist angehalten.\n")
            except:
              pass
          # while schleife verlassen
          break

        #
        # Verbindungsaufbau der verschiedenen Objekte
        #
        if actsock in rep_connlist:
          log.log("Ein Speicher bittet um Verbindung.")
          # Speicher baut verbindung auf
          newfd=conn.accept(actsock)
          rep_list.append(newfd)
```

```
if actsock in ifobj_connlist:
  log.log("Eine␣Schnittstelle␣bittet␣um␣Verbindung."
    )
  # Schnittstellenobjekt baut verbindung auf
  newfd=conn.accept(actsock)
  ifobj_list.append(newfd)

if actsock in ctrl_connlist:
  log.log("Eine␣Steuerinstanz␣bittet␣um␣Verbindung."
    )
  # M/S Werkzeug baut verbindung auf
  newfd=conn.accept(actsock)
  ctrl_list.append(newfd)

#
# Verarbeitung verschiedener Nachrichten
#

# Nachricht von einem Speicher
if actsock in rep_list:
  # Nachricht ueber bestehende Verbindung
  log.log("Ein␣Speicher␣bittet␣um␣Gehoer.")
  msg=""
  msg=conn.read_sock(actsock)
  log.log("Nachricht␣:␣"+msg[0:40])
  if msg=="":
    log.log("Oops,␣der␣hat␣einfach␣aufgelegt␣["+
      str(actsock)+"].")
    conn.close_sock(actsock)
    rep_list=dellist(rep_list,actsock)
  else:
    reply=""
    parts=string.split(msg,"␣")
    # Anmeldung eines Speichers
    if parts[0]=="CMD_CONN":
      if connected_reps.addrep(parts[1],
         parts[2], parts[3])!=-1:
        reply="CMD_CONNCONFIRM"
      else:
        reply="CMD_CONNDENY"
    else:
      # Auftragsrueckmeldung empfangen
      jobid=jobs.get_job_by_repfd(actsock)
      njob=jobs.get_jobdata(jobid)
```

```
      # Rueckmeldung an Schnittstellenobjekt
      # schicken
      conn.write_sock(njob[2],msg)
      # Speicher als idle markieren
      connected_reps.set_idle(njob[6])
      # Auftrag aus Jobliste loeschen
      jobs.remjob(jobid)


    if reply!="":
      conn.write_sock(actsock,reply)
      log.log("Sende Antwort "+reply[0:40])

# Nachricht von einer Schnittstelle
if actsock in ifobj_list:
  # Nachricht ueber bestehende Verbindung
  log.log("Eine Schnittstelle bittet um Gehoer.")
  msg=conn.read_sock(actsock)
  log.log("Nachricht : "+msg[0:40])
  if msg=="":
    log.log("Oops, der hat einfach aufgelegt ["+
     str(actsock)+"].")
    conn.close_sock(actsock)
    ifobj_list=dellist(ifobj_list,actsock)
  else:
    reply=""
    freerep=connected_reps.get_idlerep()
    if freerep[0]==-1:
      reply="Alle Speicher ausgelastet."
    else:
      # neuen auftrag erfassen und weiterleiten
      brokercounter=brokercounter + 1
      ns=conn.open_sock("",0)
      rep_list.append(ns)
      conn.connect(ns, freerep[2],
        eval(freerep[3]))
      jobs.addjob(actsock,"w",freerep[0],ns)
      conn.write_sock(ns,msg)

    if reply!="":
      conn.write_sock(actsock,reply)

# Nachricht von einem Managementobjekt
if actsock in ctrl_list:
  # Nachricht ueber bestehende Verbindung
```

```
log.log("Eine␣Managementinstanz␣bittet␣um␣Gehoer."
   )
msg=conn.read_sock(actsock)
log.log("Nachricht␣:␣"+msg[0:40])
if msg=="":
  log.log("Oops,␣der␣hat␣einfach␣aufgelegt␣["+
    str(actsock)+"].")
  conn.close_sock(actsock)
  ctrl_list=dellist(ctrl_list,actsock)
else:
  reply=""
  if msg=="CMD_CONFGET":
    # erstelle Konfigurationsstring
    brokerqueue=len(jobs.get_runningjobs())
    brstr="{("+ brokername + ","+
      brokerstate + ","+ str(brokercounter) +
      ","+ str(brokerqueue) +")}"
    reply=brstr + ";"+
      str(connected_reps.get_replist()) + ";"+
      str(jobs.get_runningjobs())

  elif msg=="CMD_EXIT":
    # Broker anhalten
    brokerstate="paused"
    # Alle Speicher beenden
    for i in connected_reps.get_replist():
      conn.send(i[2],"CMD_EXIT")
    # Abbruchbedingung fuer Mainloop setzen
    doit=0
  elif msg=="CMD_QUIT":
    # Notausstieg
    del conn
    raise SystemExit

  elif msg[:8]=="CMD_CHGST":
    # zustand einens speichers veraendern
    mlst=string.split(msg,"␣")

      chgid=connected_reps.get_repbyname(mlst[1])
    if chgid!=-1:
      if mlst[2]=="paused":
        # anhalten
        connected_reps.set_paused(chgid)
      else:
        # fortsetzen
```

```
                connected_reps.set_idle(chgid)
          elif msg[:9]=="CMD_MAPJOB":
            # Auftrag zu Speicher zuordnen und
            # abschicken
            mlst=string.split(msg," ")

              freerep=connected_reps.get_repinfo(mlst[2])
            # neuen auftrag erfassen und weiterleiten
            brokercounter=brokercounter + 1
            ns=conn.open_sock("",0)
            rep_list.append(ns)
            conn.connect(ns, freerep[2],
              eval(freerep[3]))
            jobs.set_repstate(actsock,"w")
            conn.write_sock(ns,msg)

          elif msg=="CMD_PAUSE":
            brokerstate="paused"
          elif msg=="CMD_CONT":
            brokerstate="run"
          elif msg="CMD_AUTO":
            brokerdistmethod=1
          elif msg="CMD_MAN":
            brokerdistmethod=0
          if reply!="":
            conn.write_sock(actsock,reply)

      i=i+1
del conn
```

E.3 Monitor-/Steuerungswerkzeug

E.3.1 Hauptprogramm des Monitor-/Steuerungswerkzeug

```
#!/usr/local/bin/python
# ------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname       : monitor.py
# Datum           : 03.11.1997
# letzte Änderung : 08.01.1998
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Managementobjekt mit kombinierter
#                   Monitor-/Steuerungsfunktionalität
# Anmerkungen     :
#
# ------------------------------------------------------------


# imports fuer Tk, Socketkommunikation und Stringverarbeitung
from Tkinter import *
import socket
import string


#
# Klasse des Monitor-/Steuerungswerkzeuges
#
class moni:
      #
      # Konstruktor
      #
      def __init__(self,master):
        # Variablen setzen und Socket erzeugen
        self.master=master
        self.br_anz=0
        self.sp_anz=0
        self.if_anz=0
        self.upint=5
        self.widgets={}
        self.host="localhost"
```

```
        self.port=6444
        self.connected=0
        self.conn=socket.socket(socket.AF_INET,
          socket.SOCK_STREAM)
        self.confstr=""

        # GUI aufbauen
        self.create_everything()


    #
    # Zeichnet die Oberflaeche und ordnet
    # die Ereignisse zu
    #
    def create_everything(self):
        #
        # Menues
        #
        self.mbar=Frame(self.master,relief=RAISED, bd=2)
        self.mbar.pack(side=TOP, fill=X)
        # menuebuttons einrichten
        self.m_verbindung =Menubutton(self.mbar,
          text="Verbindung",underline=0)
        self.m_broker =Menubutton(self.mbar,text="Broker",
          underline=0)
        self.m_options =Menubutton(self.mbar,text='Options',
          underline=0)
        self.m_info =Menubutton(self.mbar,text="Info",
          underline=0)

        self.m_verbindung.pack(side=LEFT)
        self.m_broker.pack(side=LEFT)
        self.m_options.pack(side=LEFT)
        self.m_info.pack(side=RIGHT)
        # menueeintraege erzeugen
        self.m1=Menu(self.m_verbindung,tearoff=NO)
        self.m1.add_command(label="Aufbauen",underline=0,
          command=self.br_connect)
        self.m1.add_command(label="Abbauen",underline=0,
          command=self.br_disconnect)
        self.m1.add_command(label="Beenden",underline=0,
          command=self.ende)

        self.m2=Menu(self.m_broker,tearoff=NO)
        self.m2.add_command(label="Anhalten",underline=0,
```

```
    command=self.br_stop)
self.m2.add_command(label="Fortsetzen",underline=0,
    command=self.br_cont)
self.m2.add_command(label="Manuelle␣Verteilung",
    underline=0,command=self.br_man)
self.m2.add_command(label="Automatische␣Verteilung",
    underline=0,command=self.br_auto)


self.m3=Menu(self.m_info,tearoff=NO)
self.m3.add_command(label="Info",underline=0,
    command=self.info1)

self.m4=Menu(self.m_options,tearoff=NO)
self.m4.add_command(label="Update",underline=0,
    command=self.brupdate)
self.m4.add_command(label="Parameter",underline=0,
    command=self.br_parms)
# menubuttons mit menues verbinden
self.m_verbindung.config(menu=self.m1)
self.m_broker.config(menu=self.m2)
self.m_options.config(menu=self.m4)
self.m_info.config(menu=self.m3)
# frame mit menubuttons als menueleiste einrichten
self.mbar.tk_menuBar(self.m_verbindung, self.m_broker,
    self.m_options,self.m_info)

# diagramm bereich einrichten
self.f2=Frame(self.master)
self.f2.pack()
self.draw=Canvas(self.f2, bg='Black', height=400,
    width=500,scrollregion=(0,0,1000,400))
self.draw.pack()
self.dsbx=Scrollbar(self.f2,orient=HORIZONTAL)
self.dsbx.pack(fill=X,side=BOTTOM,expand=YES)
self.draw.config(xscrollcommand=self.dsbx.set)
self.dsbx.config(command=self.draw.xview)

self.draw.bind("<Double-Button-1>",self.action)

# updateintervall selektor montieren
self.f4=Frame(self.master)
self.f4.pack(fill=X)

self.f3=Frame(self.f4,relief=SUNKEN,bd=3)
```

```
        self.f3.pack(side=LEFT,fill=X)
        self.sc1=Scale(self.f3,{'from':0,'to':300},
          label="Updateintervall␣(sek.)",showvalue=YES,
          orient=HORIZONTAL,length=300,tickinterval=50,
          command=self.setupint)
        self.sc1.pack(side=LEFT)
        self.sc1.set(self.upint)

        # Status Ampel
        from ampel import ampel
        self.f5=Frame(self.f4,relief=SUNKEN,bd=3)
        self.f5.pack(side=RIGHT)
        self.amp=ampel(self.f5,20)
        self.amp.pack("side=RIGHT")
        self.amp.switch(3)
        self.amp.bind("<1>",self.connchange)
        self.lamp=Label(self.f5,text="Nicht␣verbunden")
        self.lamp.pack(side=LEFT)

        # buttons anbringen
        self.f1=Frame(self.master)
        self.f1.pack(side=RIGHT,fill=X)
        self.b4= Button(self.f1, text='Quit',
          command=self.ende)
        self.b4.pack(side=RIGHT)

# --------------------------- Ereignismethoden  --------------------

    #
    # Beendet das M/S Werkzeug
    #
    def ende(self):
        # wenn wir noch mit dem Broker verbunden sind
        if self.connected==1:
            # beim Broker abmelden
            self.br_disconnect()
        raise SystemExit

    #
    # Wertet einen Mausklicks auf die Ampel aus
    # und ruft die entsprechenden Methoden auf
    #
    def connchange(self,event):
        # ermitteln welches Ampellicht angeklickt wurde
        typ=self.amp.which(event.x,event.y)
```

```
    # gruen wurde geklickt, wenn wir noch nicht mit
    # dem Broker verbunden sind dann Verbindung aufbauen
    if typ==3:
      if self.connected==0:
        self.br_connect()
    # gelb wurde angeklickt, siehe oben
    elif typ==2:
      if self.connected==0:
        self.br_connect()
      self.brupdate()
    # rot wurde angeklickt, wenn wir mit dem Broker
    # verbunden sind dann trenne Verbindung
    else:
      if self.connected==1:
        self.br_disconnect()

  # setzt neues updateintervall, immer dann wenn
  # der schieberegler veraendert wurde
  def setupint(self,x):
    self.upint=self.sc1.get()

  #
  # ermittelt auf welche Instanz im Hauptfenster ein
  # Doppelklick ausgefuehrt wurde und zeigt das
  # entsprechende Konfigurationsfenster an.
  #
  def action(self,event):
    x=event.x
    y=event.y
    #ermittle objektname
    for i in self.widgets.keys():
      a=self.widgets[i]
      xo=a[8][0]
      yo=a[8][1]
      if (x >=xo) and (x<=xo+100) and (y>=yo) and (y<=yo+
       100):
        obj=i
    # suche objekt in configstring
    self.olist=[]
    clist=string.split(self.confstr,";")
    # ist es der Broker
    if string.split(clist[0][2:-2],",")[0]==obj:
      list=string.split(clist[0][2:-2],",")
      self.olist=["Broker",list[0],self.host,
         str(self.port),list[1],list[2],list[3]]
```

```
else:
  spl=eval(clist[1])
  for j in spl.keys():
    if spl[j][1]==obj:
      self.olist=["Speicher",spl[j][1],spl[j][2],
         spl[j][3],spl[j][4],"0","0"]
if self.olist==[]:
   afl=eval(clist[2])
  for j in afl.keys():
    if afl[j][1]==obj:
      self.olist=["Auftrag",afl[j][1],afl[j][2],
         afl[j][3],afl[j][4],"0","0"]



# fenster mit der konfiguration des objekts anzeigen
self.iw=Tk()
 if1=Frame(self.iw)
if1.pack(fill=X)
if2=Frame(self.iw)
if2.pack(side=BOTTOM)
ml1=Label(if1,text="Objekttyp␣␣:␣"+self.olist[0])
ml2=Label(if1,text="Objektname␣:␣"+self.olist[1])
ml3=Label(if1,text="Hostname␣␣␣:␣"+self.olist[2])
ml4=Label(if1,text="Port␣␣␣␣␣␣␣:␣"+self.olist[3])

self.objstate=self.olist[4]
ml5=Frame(if1)

self.vcb1=StringVar()
rb1=Radiobutton(ml5,text="Laufend",value="r",
   variable=self.vcb1,command=self.sv_1)
rb2=Radiobutton(ml5,text="Angehalten",value="s",
   variable=self.vcb1,command=self.sv_2)
rb3=Radiobutton(ml5,text="Terminiert",value="t",
   variable=self.vcb1,command=self.sv_3)
rb1.pack(side=LEFT)
rb2.pack(side=LEFT)
rb3.pack(side=LEFT)
if self.objstate=="run":
  rb1.select()
  self.vcbt="run"
elif self.objstate=="paused":
  rb2.select()
  self.vcbt="paused"
```

```
  else:
    rb3.select()

  ml6=Label(if1,text="Aufträge␣␣␣:␣"+self.olist[5])
  if self.olist[0]=="Broker":
    ml7=Label(if1,text="offene␣Auftr.␣:␣"+
      self.olist[6])
  elif self.olist[0]=="Speicher":
    ml7=Label(if1,text="Auslastung␣␣␣␣:␣"+
      self.olist[6])
  else:
   ml7=Label(if1,text="")

  bl1=Button(if2,text="OK",width=10,
     command=self.setobj_parms)
  bl2=Button(if2,text="Cancel",width=10,
     command=self.iw.destroy)
  ml1.pack(anchor="w")
  ml2.pack(anchor="w")
  ml3.pack(anchor="w")
  ml4.pack(anchor="w")
  ml5.pack(anchor="w")
  ml6.pack(anchor="w")
  ml7.pack(anchor="w")
  bl1.pack(side=LEFT)
  bl2.pack(side=LEFT)
  self.iw.wait_window(self.iw)
  del self.iw


#
# Drei Hilfsroutinen zum auslesen
# der Parameter des Konfigurationsfensters
#
def sv_1(self):
  self.vcbt="run"
def sv_2(self):
  self.vcbt="paused"
def sv_3(self):
  self.vcbt="term"

#
# sendet eine Zustandsaenderung an den Broker
#
def setobj_parms(self):
```

```
  # commando erstellen
  self.vcb1=self.vcbt
  self.objstate= self.vcb1
  commandstr="CMD_CHGST␣"+self.olist[1]+"␣"+
   self.objstate
  # Kommando an Broker senden
  if self.connected==1:
    self.conn.send(commandstr)
  self.iw.destroy()
  del self.olist, self.objstate, self.vcb1, self.vcbt

#
# aktualisiert die graphische Anzeige
#
def refresh(self,confstr):
  # alte widgets löschen
  for i in self.widgets.keys():
    for j in range(0,8):
      w=self.widgets[i][j]
      self.draw.delete(w)

  self.widgets={}
  self.sp_anz=0
  self.br_anz=0
  self.if_anz=0
  self.master.update_idletasks()
  # neue widgets aufbauen
  clist=string.split(confstr,";")
  # Broker erzeugen
  brlist=string.split(clist[0][2:-2],",")
  brstr="X,"+brlist[0]+","+self.host+","+str(self.port)+
   ","+brlist[1]+","+brlist[2]+","+brlist[3]
  self.broker(brstr)
  # Speicher erzeugen
  spdict=eval(clist[1])
  for i in spdict.keys():
    spl=spdict[i]
    spstr=str(spl[0])+","+spl[1]+","+spl[2]+","+spl[3]+
     ","+spl[4]+",0,0"
    self.speicher(spstr)

  ifdict=eval(clist[2])
  for i in ifdict.keys():
    ifl=ifdict[i]
```

```
#
# Zeichnet eine Auftragsinstanz
#
def auftrag(self,confstr=""):
  # Koordinaten berechnen
  y=10
  x=self.if_anz*105+5

  if confstr!="":
    clist=string.split(confstr,",")
  else:
    clist=["␣","␣","␣","␣","␣","␣","␣"]
  n=str(clist[1])
  h=str(clist[2])
  p=str(clist[3])
  s=str(clist[4])
  c=str(clist[5])

  # Auftrag zeichnen
  s0=self.draw.create_rectangle(x,y,x+100,y+100,
     fill='lightGreen')
  s1=self.draw.create_text(x+1,y+10,text="Name:"+n,
     anchor='w',width=95)
  s2=self.draw.create_text(x+1,y+25,text="Host:"+h,
     anchor='w',width=95)
  s3=self.draw.create_text(x+1,y+40,text="Port:"+p,
     anchor='w',width=95)
  s4=self.draw.create_text(x+1,y+55,text="State:"+s,
     anchor='w',width=95)
  s5=self.draw.create_text(x+1,y+70,text="Count:"+c,
     anchor='w',width=95)
  self.if_anz=self.if_anz+1
  s6=""
  s7=self.draw.create_line(x+50,y+100,self.br_anz*50+5,
     150,fill="yellow",width="5",arrow=BOTH)
  s8=(x,y)
  # Widgets merken damit man weiter darauf zugreifen
  # kann
  self.widgets[n]=(s0,s1,s2,s3,s4,s5,s6,s7,s8)

#
# zeichnet eine Speicherinstanz
#
def speicher(self,confstr=""):
  # Koordinaten berechnen
```

```
    y=290
    x=self.sp_anz*105+5

    if confstr!="":
      clist=string.split(confstr,",")
    else:
      clist=[" "," "," "," "," "," "," "]
    n=str(clist[1])
    h=str(clist[2])
    p=str(clist[3])
    s=str(clist[4])
    c=str(clist[5])
    l=str(clist[6])
    # Speicher zeichnen
    s0=self.draw.create_rectangle(x,y,x+100,y+100,
      fill='moccasin')
    s1=self.draw.create_text(x+1,y+10,text="Name :"+n,
      anchor='w',width=90)
    s2=self.draw.create_text(x+1,y+25,text="Host :"+h,
      anchor='w',width=90)
    s3=self.draw.create_text(x+1,y+40,text="Port :"+p,
      anchor='w',width=90)
    s4=self.draw.create_text(x+1,y+55,text="State:"+s,
      anchor='w',width=90)
    s5=self.draw.create_text(x+1,y+70,text="Count:"+c,
      anchor='w',width=90)
    s6=self.draw.create_text(x+1,y+85,text="Load :"+l,
      anchor='w',width=90)
    self.sp_anz=self.sp_anz+1
    s7=self.draw.create_line(x+50,y,self.br_anz*50+5,250,
      fill="white",width="5",arrow=BOTH)
    s8=(x,y)

    # Widgets merken damit man weiter darauf zugreifen
    # kann
    self.widgets[n]=(s0,s1,s2,s3,s4,s5,s6,s7,s8)

  #
  # zeichnet eine Brokerinstanz
  #
  def broker(self,confstr=""):
    # berechne Koordinaten
    y=150
    x=self.br_anz*105+5
```

```
    if confstr!="":
      clist=string.split(confstr,",")
    else:
      clist=["␣","␣","␣","␣","␣","␣","␣"]
    n=str(clist[1])
    h=str(clist[2])
    p=str(clist[3])
    s=str(clist[4])
    c=str(clist[5])
    q=str(clist[6])
    # Zeichne Broker
    s0=self.draw.create_rectangle(x,y,x+100,y+100,
       fill='Red')
    s1=self.draw.create_text(x+1,y+10,text="Name␣␣:␣"+n,
       anchor='w')
    s2=self.draw.create_text(x+1,y+25,text="Host␣␣:␣"+h,
       anchor='w')
    s3=self.draw.create_text(x+1,y+40,text="Port␣␣:␣"+p,
       anchor='w')
    s4=self.draw.create_text(x+1,y+55,text="State␣:␣"+s,
       anchor='w')
    s5=self.draw.create_text(x+1,y+70,text="Count␣:␣"+c,
       anchor='w')
    s6=self.draw.create_text(x+1,y+85,text="Queued:␣"+q,
       anchor='w')
    self.br_anz=self.br_anz+1
    s7=""
    s8=(x,y)
    # Widgets merken damit man weiter darauf zugreifen
    # kann
    self.widgets[n]=(s0,s1,s2,s3,s4,s5,s6,s7,s8)

  #
  # Zeigt das About an
  #
  def info1(self):
    iw=Tk()
    if1=Frame(iw)
    if1.pack()
    m1=Message(if1,text="
       Monitor␣V0.99\n\n␣(c)␣by␣Hans␣Matzen,␣1997,\n
     Germany, Frankfurt/M\n email:
        hans@tm.informatik.uni-frankfurt.de",
     width="10c")
    b1=Button(if1,text="OK",command=iw.destroy)
```

```
    m1.pack()
    b1.pack()
    iw.wait_window(iw)
    del iw

  #
  # zeigt das Konfigurationsfenster fuer die Brokerparameter
  # an
  #
  def br_parms(self):
    # fenster aufbauen
    self.iw=Tk()
    self.if1=Frame(self.iw)
    self.if1.pack()
    self.if2=Frame(self.iw)
    self.if2.pack(side=BOTTOM)
    self.ml1=Label(self.if1,text="Hostname:")
    self.ml2=Entry(self.if1,width=30)
    self.ml3=Label(self.if1,text="Port:")
    self.ml4=Entry(self.if1,width=30)
    self.bl1=Button(self.if2,text="OK",width=10,
      command=self.setbr_parms)
    self.bl2=Button(self.if2,text="Cancel",width=10,
      command=self.iw.destroy)
    self.ml1.pack()
    self.ml2.pack()
    self.ml3.pack()
    self.ml4.pack()
    self.bl1.pack(side=LEFT)
    self.bl2.pack(side=LEFT)
    self.ml2.insert(END,self.host)
    self.ml4.insert(END,str(self.port))
    # warten bis Fenster geschlossen wird
    self.iw.wait_window(self.iw)


  #
  # setzt die im konfigurationsfenster gesetzten werte
  #
  def setbr_parms(self):
    self.host=self.ml2.get()
    self.port=eval(self.ml4.get())
    self.iw.destroy()
    del self.iw
```

```
#
# baut die Verbindung zum Broker auf
#
def br_connect(self):
  # pruefe ob bereits verbunden
  if self.connected==0:
    # versuche verbindung aufzubauen
    try:
      # connect durchfuehren
      self.conn.connect(self.host,self.port)
      # anzeige aktualisieren
      self.lamp.config(text="Verbunden")
      self.amp.switch(1)
      self.connected=1
      # automatische aktualisierung der Statusdaten
      # starten
      self.master.after(self.upint*1000,
        self.brupdate)
    except:
      # wenns nicht klappt
      # zeige Fehlermeldung
      import time
      self.lamp.config(text="Verbindungsfehler")
      self.amp.switch(2)
      self.master.update_idletasks()
      time.sleep(3)
      self.lamp.config(text="Nicht verbunden")
      self.amp.switch(3)

#
# fordert beim Broker die aktuellen Statusdaten an
# und zeigt sie an
#
def brupdate(self):
  # sind wir verbunden ?
  if self.connected==1:
    # wenn ja fordere statusdaten beim broker an
    self.conn.send("CMD_CONFGET")
    # warte auf antwort vom Broker
    self.confstr=self.conn.recv(8192)
    # anzeigen aktualisieren
    self.refresh(self.confstr)
    self.amp.switch(1)
    self.master.update_idletasks()
```

```
    # automatischen update starten
    self.master.after(self.upint*1000,self.brupdate)


#
# trennt die Verbindung zum Broker
#
def br_disconnect(self):
  # sind wir verbunden ?
  if self.connected==1:
    # wenn ja sende Meldung an Broker
    self.conn.send("QUIT")
    self.conn.close()
    del self.conn
    # hole neuen Socket fuer naechste Verbindung
    self.conn=socket.socket(socket.AF_INET,
      socket.SOCK_STREAM)
    self.connected=0
    # aktualisiere Anzeige
    self.lamp.config(text="Nicht␣verbunden")
    self.amp.switch(3)


#
# hält den Broker an
#
def br_stop(self):
  if self.connectd==1:
    self.conn.send("CMD_PAUSE")

#
# startet den broker wieder
#
def br_cont(self):
  if self.connectd==1:
    self.conn.send("CMD_CONT")

#
# sendet einen Zuordnungsauftrag an den Broker
#
def br_mapcommand(self,job,spname):
  if self.connectd==1:
    self.conn.send("CMD_MAPJOB␣"+str(job)+"␣"+
      str(spname))
#
```

```
        # schaltet auf manuelle verteilung um
        #
        def br_man(self):
          if self.connectd==1:
            self.conn.send("CMD_MAN")

        #
        # schaltet auf automatische verteilung um
        #
        def br_auto(self):
          if self.connectd==1:
            self.conn.send("CMD_AUTO")


#
# Hauptprogramm
#
root=Tk()
to=moni(root)
root.mainloop()
```

E.3.2 Verbindungsindikator-Widgetklasse

```
#!/usr/users2/diplom/hans/python/python/bin/python
# -----------------------------------------------------------------
# Projekt           : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1997
# Dateiname         : ampel.py
# Datum             : 03.11.1997
# letzte Änderung : 11.11.1997
# Autor             : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache           : Python v1.4
# Beschreibung      : Eine Ampelklasse zur Benutzung in graphischen
#                     Oberflaechen, die mit Tk realisiert wurde
# Anmerkungen       :
# -----------------------------------------------------------------

# imports
from Tkinter import *
import time


class ampel:

      #
      # Konstruktor
      #
      def __init__(self,master,width):
        # Variablen setzen
        self.master=master
        self.height=width*3
        self.width=width
        # Ampel aufbauen
        self.create_everything()

      #
      # Zeichnet die Ampel
      #
      def create_everything(self):
        self.pic=Canvas(self.master,background="Black",
           width=self.width,height=self.height,borderwidth=0)
        self.pic.create_rectangle(1,1,self.width+1,self.height+
          1,fill="Grey")
```

```
  ypos1 = self.width
  ypos2 = 2 * self.width
  ypos3 = 3 * self.width

  xwidth=ypos1
  if self.height<xwidth:
    xwidth=self.height

  xpos1=1
  xpos2=self.width

  self.pic.create_oval(xpos1, 1,xpos2,ypos1,tag="l1",
     fill="black")
  self.pic.create_oval(xpos1,ypos1,xpos2,ypos2,tag="l2",
     fill="black")
  self.pic.create_oval(xpos1,ypos2,xpos2,ypos3,tag="l3",
     fill="black")


#
# Ein Wrapper fuer die Standar-Tk-Pack-Function
#
def pack(self,args=""):
  self.pic.pack(side=RIGHT)

#
# Ein Wrapper fuer die Standar-Tk-Bind-Function
#
def bind(self,event,func):
  self.pic.bind(event,func)

#
# Ermittelt unter Angabe der Koordinaten
# auf welches Ampellicht geklickt wurde
#
def which(self,x,y):
  if y>2*self.width:
    self.switch(1)
    return 3
  elif y>self.width:
    self.switch(2)
    return 2
  else:
    self.switch(3)
    return 1
```

```
#
# Schaltet die Ampel um
# num =0 --> Ampel aus
# num =1 --> Ampel gruen
# num =2 --> Ampel gelb
# num =3 --> Ampel rot
#
def switch(self,num):
  if num<0 or num>3:
    return -1
  else:
    tagstr="l"+str(num)
    if num==0:
      col1="black"
      col2="black"
      col3="black"
    if num==1:
      col1="green"
      col2="black"
      col3="black"

    if num==2:
      col1="black"
      col2="yellow"
      col3="black"

    if num==3:
      col1="black"
      col2="black"
      col3="red"

    self.pic.itemconfig("l1",fill=col3)
    self.pic.itemconfig("l2",fill=col2)
    self.pic.itemconfig("l3",fill=col1)

    return 0
```

E.4 Aktiver Speicher

E.4.1 Beispiel für Konfigurationsdatei

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : speicher.conf
# Datum            : 03.11.1997
# letzte Änderung  : speicher.conf
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Konfigurationsdatei fuer Speicher
# Anmerkung        :
# -------------------------------------------------------------

#
# Portnummer fuer eingehende Auftraege von Brokern
# Syntax: RECEIVE_PORT <portnr>
#
RECEIVE_PORT 6501


#
# Portnummer fuer Verbindungsaufnahme der Broker
# BROKER_ADDRESS <hostname>,<portnr>
#
BROKER_ADDRESS localhost,6433


#
# Portnummer fuer Managementkomponente
# Syntax: CONTROL_PORT <portnr>
#
CONTROL_PORT 6544


#
# Python Modul in dem die Filter Routinen stehen
# Syntax: MAP_MODULE <modulname>
#
MAP_MODULE filters
```

```
#
# Eintraege, die einem Kommando einen Filter zuordnen
# Syntax: MAP_COMMAND <command>,<filterfunction>
#
MAP_COMMAND PUT,dummyfilter
MAP_COMMAND ACTIVATE,dummyfilter
MAP_COMMAND INFO,dummyfilter


#
# Logdatei
# Syntax: LOG_FILE <filename>
#
LOG_FILE ../log/speicher.log
```

E.4.2 Hauptprogramm des Speichers

```
#!/usr/users2/diplom/hans/python/python/bin/python
# --------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : speicher.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Dies ist das Hauptprogramm der Speicher-
#                    komponente
#
# Anmerkungen      :
#
# --------------------------------------------------------------


#
# imports
#
from sc_globals import *
import time
import select
import c_bsio
import regex
import string
import c_metadoc
import urllib
import os
import posix
import class_funcs
import pgpy
import sys
import socket
import signal
import c_log
import signalfuncs
import confparse
# Ablaufumgebungen importieren
from py_abl import *
```

```
# setzte Defaultwerte
MAXBUFF=32768
CONFIG_FILE="speicher.conf"

# verabeitung  Kommandozeilenparameter
# pruefe parameter
if len(sys.argv)<2:
      print "Usage:␣speicher.py␣<name>␣[-f␣<configfile>]"
      sys.exit(1)

# hole kommandozeilenparameter
spobjname=sys.argv[1]
try:
      CONFIG_FILE=sys.argv[3]
except:
      pass

# installiere Signalhandler
msg=""
def sig_int(a,b):
      global msg
      msg="QUIT"
signal.signal(signal.SIGUSR1,signalfuncs.sig_int)


# Verarbeite Konfigurationsfile
params=confparse.confparse(CONFIG_FILE)

brlist=[]
rcvlist=[]
ctrlist=[]
modlist=[]
mapdict={}
lfile="./sp.log"
i=0

while i<len(params):
      el=params[i]
      if el[0]=="RECEIVE_PORT":
        rport=eval(el[1])
        rcvlist.append(rport)
      elif el[0]=="BROKER_ADDRESS":
        baddr=string.split(el[1],",")
        brlist.append(baddr)
      elif el[0]=="CONTROL_PORT":
```

```
        cport=eval(el[1])
        ctrlist.append(cport)
      elif el[0]=="LOG_FILE":
        lfile=el[1]
      elif el[0]=="MAP_MODULE":
        mmod=el[1]
        modlist=string.split(mmod,",")
      elif el[0]=="MAP_COMMAND":
        map=string.split(el[1],",")
        mapdict[map[0]]=map[1]
      i=i+1


# Oeffne Logfile
log=c_log.c_log("Speicher␣"+spobjname,lfile)

log.log("Gestartet")

host="localhost"
port=rcvlist[0]

# instanziiere Auftragssocket
conn=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
try:
      conn.bind(host,port)
      conn.listen(5)
except:
      log.log("Auftragssocketadresse␣bereits␣in␣Benutzung.")
      log.log("Speicher␣beendet␣sich.")
      raise SystemExit

# Anmeldung beim Broker
conn1=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
# Verbindung aufbauen
try:
      conn1.connect(brlist[0][0],eval(brlist[0][1]))
except:
      log.log("Broker␣nicht␣verfuegbar")
      log.log("Speicher␣beendet␣sich")
      raise SystemExit

# Anmeldung schicken
conn1.send("CMD_CONN␣"+spobjname+"␣"+conn.getsockname()[0]+"␣"+
  str(conn.getsockname()[1]))
brmsg=conn1.recv(500)
```

```
# Wurde die Anmdeldung akzeptiert?
if brmsg=="CMD_CONNDENY":
      log.log("Der␣Broker␣verweigert␣die␣Anmeldung")
      raise SystemExit
else:
      log.log("Anmeldung␣beim␣Broker␣erfolgreich␣abgeschlossen."
        )


# init recv buffer da steht immer
# die letzte empfangene Nachricht drin
msg=""

# solange wir nicht beenden sollen werden Auftraege verarbeitet
while msg!="QUIT"and msg!="CMD_EXIT":
      log.log("Warte␣auf␣Auftraege␣an␣Adresse"+
       str(conn.getsockname()))
      # warten auf Nachrichten
      activ=select.select([conn.fileno()],[],[])
      # Verbindung aufbauen
      news,newsaddr=conn.accept()
      log.log("Verbindung␣aufgebaut␣mit"+str(newsaddr))

      # auf Auftrag warten
      msg=""
      msg=news.recv(MAXBUFF)
      log.log("Habe␣folgenden␣Auftrag␣empfangen:")
      log.log(msg)
      log.log("----End␣of␣message---")

      # konvertiere NAchricht gemaess der definierten Filter
      # im Konfigurationsfile
      msg_parts=string.split(msg,"␣")
      if mapdict.has_key(msg_parts[0]):
        statement="import␣"+modlist[0]
        exec statement
        statement="msgnew="+modlist[0]+"."+
         mapdict[msg_parts[0]]+"(msg)"
        exec statement

      # verarbeite eingegangene Nachricht
      # Auftrag parsen (Variable: msg)
      cmd=""
      params=""
```

```
tmpfname=""
tmpfile=""
pos1=regex.search(" ",msg)
if pos1>=0:
  cmd=string.upper(msg[:pos1])
else:
  cmd=string.upper(msg)

if cmd=="PUT":
  pos2=regex.search("</META>",msg)
  tmpfile=msg[pos1+1:pos2+7]
  tmpfname=class_funcs.get_uniquename()
  fd=c_bsio.c_bsio(tmpfname,"a+")
  fd.open()
  fd.write(tmpfile)
  fd.close()
  params=msg[pos2+8:]
else:
  params=msg[pos1+1:]


# ----------------------------------------------------------
# fuehre Kommando aus
# ----------------------------------------------------------

# ----------------------------------------------------------
# ein neues Dokument soll gespeichert werden
# ----------------------------------------------------------
if cmd=="PUT":
  # speichern der Metadokument Informationen
  # instanziiere Objekt zum Zugriff auf das Metadokument
  meta=c_metadoc.c_metadoc(tmpfname)
  # hole neue Dokumenten-ID
  newdocid=class_funcs.get_newdocid()
  # baue Verbindung zur Datenbank auf
  main_conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
  # ist die DTD des Dokuments schon erfasst?
  dtdnum=class_funcs.get_dtdnum(meta.get_dtd())

  # lese Metadaten aus dem Metadokument
  infostr =meta.get_info()
  infostr =infostr[string.find(infostr,">")+
   1:string.rfind(infostr,"</")]
  propdict=meta.get_property()
  methods=meta.get_methods()
```

```
# bereite SQL Query vor
dbclass="doc_description"
query="insert␣into␣"+dbclass+"␣values␣("+newdocid+",0,"

query=query + "'"+ propdict["title"] + "',"
query=query + "'"+ propdict["author"] + "',"
query=query + "'"+ propdict["owner"] + "',"
query=query + "'"+ propdict["ident"] + "',"
query=query + "'"+ propdict["pubkey"] + "',"
query=query + "'"+ infostr + "',"+str(dtdnum)+",0,0);"


# fuehre SQL Query aus (speichere Metadaten in der DB)
main_res=main_conn.exec_query(query)
erg=main_res.commandstatus()
# hat alles geklappt?
if str(erg)[0:6]!="INSERT":
  log.log("Fehler␣beim␣Speichern␣der␣Metadaten")

# Methoden in der DB speichern
for j in methods.keys():
  # SQL Query vorbereiten
  dbclass="doc_methods"
  query="insert␣into␣"+dbclass+"␣values␣("+newdocid+
   ","
  query=query + "'"+ methods[j][0] + "',"
  query=query + "'"+ methods[j][1] + "',"
  query=query + "'"+ methods[j][2] + "');"
  main_res=main_conn.exec_query(query)
  # und abschicken (eine Methode in DB sspeichern)
  erg=main_res.commandstatus()
  # hat ales geklappt ?
  if str(erg)[0:6]!="INSERT":
    log.log("Fehler␣beim␣Speichern␣der␣Methode:",
      j)

# -----------------------------
# Metadaten speichern ENDE
# -----------------------------

# Dokumenteninhalt in temporaere Datei ablegen
datafname=urllib.urlretrieve(meta.get_data()) [0]

#
# aktiviere Methode
```

```
  #
  # ermittle URL der Methode die in zum Aufruf in
  # params uebergeben werden
  if methods.has_key(params):
      methurl=methods[params][1][regex.search("URL:",
         methods[params][1])+4:]

  # hole methode
  methtmpfile,methheader=urllib.urlretrieve(methurl)

  # lasse Methode von Ablaufumgebung ausfuehren
  exec_erg=exec_mc(newdocid,methtmpfile,params,
     datafname)

  # pruefe auf fehler und sende ergebnis
  if exec_erg[0:6]=="RESULT":
    log.log("Methode␣ohne␣Fehler␣ausgefuehrt.")
  else:
    log.log("Fehler␣beim␣Ausfuehren␣der␣Methode.")
  news.send(exec_erg)



# ---------------------------------------------------------
# eine Dokumentenmethode soll aktiviert werden
# ---------------------------------------------------------
if cmd=="ACTIVATE":
  # parse Parameter
  import string
  ident=string.split(params,"␣")[0]
  method=string.split(params,"␣")[1]
  mparams=string.split(params,"␣")[2]
  # baue Verbindung zur DB auf und pruefe, ob die
  # Methode exisitiert
  pgc=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
  query="select␣*␣from␣doc_methods␣where␣docnr="+
   str(ident)+"␣and␣"
  query=query+"␣meth_class='"+method+"';"

  res=pgc.exec_query(query)
  if res.get_tuple_count()!=0:
    # ermittle Methodenname aus query Ergebnis
    methurl=res.get_fieldvalue(0,2)
    log.log("die␣Methode␣steht␣in␣",methurl)
```

```
    # hole methode
    methtmpfile,methheader=urllib.urlretrieve(methurl)

    # lasse Methode von Ablaufumgebung ausfuehren
    exec_erg=exec_mc(ident,methtmpfile,mparams)

    # pruefe auf fehler und sende ergebnis
    if exec_erg[0:6]=="RESULT":
      log.log("Methode␣ohne␣Fehler␣ausgefuehrt.")
    else:
      log.log("Fehler␣beim␣Ausfuehren␣der␣Methode.")
    news.send(exec_erg)

  else:
    news.send("Die␣Methode␣"+method+
     "␣wurde␣fuer␣das␣Dokument␣mit␣der␣ident␣"+str(ident)+
     "␣nicht␣gefunden.")


# ------------------------------------------------------------
# es sollen Informationen ueber die Programmbibliothek
# geliefert werden
# ------------------------------------------------------------
if cmd=="INFO":
  import c_info
  info=c_info.c_info()
  erg=info.get_info()
  log.log("Sende␣Programmbibliothek␣Informationen.")
  news.send(erg)

# ------------------------------------------------------------
# der Speicher soll sich beenden
# ------------------------------------------------------------
if cmd=="QUIT":
  log.log("Habe␣Terminierungskommando␣erhalten.")
  news.send("Speicher␣beendet␣sich.")
  msg="QUIT"

# ------------------------------------------------------------
# Unbekanntes Kommando empfangen
# ------------------------------------------------------------
if not (cmd in ["PUT","INFO","ACTIVATE","QUIT"]):
  log.log("Kommando␣nicht␣erkannt.")
  log.log("Sende␣Fehlermeldung.")
  news.send("Kommando␣nicht␣erkannt.")
```

```
        log.log("Nachricht␣wurde␣verarbeitet.")

        time.sleep(2)
        log.log("Schliesse␣Socket")
        news.close()
        del news
        del newsaddr



log.log("---Speicher␣normal␣beendet---")
```

E.4.3 Filterroutinen

```
#!/usr/local/bin/python
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : filter.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hier werden Funktionen definiert, die die
#                    eingehenden Aufträge an das interne Protokoll
#                    des Speichers anpassen.
#
# Anmerkungen      :
# ---------------------------------------------------------------


# ---------------------------------------------------------------
# Name             : dummyfilter
# Datum            : 03.11.1997
# letzte Änderung :
# Typ              : Funktion
# Argumente        : msg - ein Textstring
# Rückgabewert     : msg - ein Textstring
# ---------------------------------------------------------------
# Beschreibung     : Da diese Funktion nur einen Dummyfilter
#                    realisiert, wird lediglich das übergebene
#                    Argument unverändert zurückgegeben.
# ---------------------------------------------------------------
# Anmerkungen      :
# ---------------------------------------------------------------
def dummyfilter(msg):
        return msg
```

E.4.4 Ablaufumgebung

```
#!/usr/local/bin/python
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : py_abl.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     :
#
# Anmerkungen      :
#
# ---------------------------------------------------------------
def exec_mc(docid,methodtmpfile,params,datafname=""):
      exectmpfile=class_funcs.getuniquename()
      fd1=c_bsio.c_bsio(exectmpfile,"a+")
      fd1.open()

      # schreibe skript header
      fd1.write(DL_EXECPYTHON+"\n")
      fd1.write("#␣Dieses␣Skript␣wurde␣automatisch␣erstellt.\n")
      fd1.write("#␣Digitale␣Bibliotheken␣Projekt,
        ␣Uni-Frankfurt\n")
      fd1.write("#␣Lehrstuhl␣Telematik,␣1997\n\n")
      fd1.write("from␣sc_globals␣import␣*\n\n")
      # schreibe globale systemvariablen
      # docid und datasource setzen
      if datafname!="":
        fd1.write("CURRENT_INFILE=\'"+datafname+"\'\n")
      fd1.write("DOCID="+str(docid)+"\n")
      fd1.write("
        #␣---------------␣auto␣header␣ende␣-------------\n")

      # die Methode dranhaengen
      fd2=c_bsio.c_bsio(methodtmpfile,"r")
      fd2.open()
      buff="␣"
      while buff!="":
        buff=fd2.read(512)
```

```
  fd1.write(buff)
fd2.close()
fd1.close()

# Methode ausfuehren, aufraeumen und Rueckmeldung schicken
posix.chmod(exectmpfile,0755)

if os.system(exectmpfile+"␣"+params+">"+TMPPATH+
 "output.dali")==0:
    log.log("Methode␣ohne␣Fehler␣ausgefuehrt.")
    # hole Methoden Output
    fdd=open(DL_TMPPATH+"output.dali","r")
    erg=""
    buff="␣"
    while buff!="":
      buff=fdd.readline()
      erg=erg+buff

    fdd.close()

    ret="RESULT␣"+str(docid)+"\n"+erg
    del erg,buff,fdd
else:
  log.log("Fehler␣beim␣Ausfuehren␣der␣Methode.")
  ret="ERROR␣"+str(docid)+"\n"+erg

# temporaere Dateien loeschen
os.system("rm␣"+"-f␣"+methtmpfile)
os.system("rm␣"+"-f␣"+exectmpfile)
os.system("rm␣"+"-f␣"+DL_TMPPATH+"output.dali")
# und ergebnis zurueckgeben
return ret
```

E.4.5 Klasse zur Ausführung eines INFO-Kommandos

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : c_info.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse die Informationen ueber alle
#                    verfuegbaren Klassen der Programmbibliothek
#                    liefert
# Anmerkungen      :
#
# -------------------------------------------------------------

import os
import string
from sc_globals import *



class c_info:
      #
      # Konstruktor
      #
      def __init__(self):
        self.info=""

      #
      # Liefert Informationen ueber alle Klassen im
      # Pfad der Programmbibliothek
      #
      def get_info(self):
        found=[]
        found = os.listdir(DL_MIPATH)
        miclass=[]
        h1=0
        # stelle Liste der Dateien zusammmen
        # und speichere sie in miclass
        while h1<len(found)-1:
          if found[h1][-4:-1]!=".py":
```

```
        miclass.append(found[h1])
        h1=h1+1

    # Durchlaufe alle Klassen
    while miclass !=[]:
        print "Scanne␣Klasse:␣",miclass[0]
        fd=open(DL_MIPATH+miclass[0],"r")
        zeile="␣"
        # Zeile fuer Zeile durchsuchen
        while zeile!="":
          zeile=fd.readline()
          # eine neue Klasse ?
          if string.lstrip(zeile)[0:6]=="class␣":
            self.info=self.info+zeile+"\n"
          # ein neue Methode ?
          if string.lstrip(zeile)[0:4]=="def␣":
            self.info=self.info+"␣␣␣"+zeile[:-2]+
              "\n"


        fd.close()
        del miclass[0]
        self.info=self.info+"\n\n"

    return self.info
```

E.5 Hilfsklassen

E.5.1 Logfile-Klasse

```
# ----------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : c_log.py
# Datum            : 03.11.1997
# letzte Änderung  :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Eine einfache Logfile Klasse
# Anmerkungen      :
# ----------------------------------------------------------------
# import Standard Dateizugriffsklasse
import c_bsio

class c_log(c_bsio.c_bsio):
      #
      # Konstruktor
      #
      def __init__(self,pname,fname):
        import posix
        # erzeuge Dateiobjekt
        c_bsio.c_bsio.__init__(self,fname,"a+")
        # Merke Logdateizeilentrailer
        self.trailer=pname+" ("+str(posix.getpid())+"): "
        # oeffne Logdatei
        self.open()
      #
      # Destruktor
      #
      def __del__(self):
        # schliesse Logdatei
        self.close()
      #
      # schreibt eine Zeile in das Logfile
      #
      def log(self,line):
        self.write(self.trailer+line+"\n")
        self.fd.flush()
```

E.5.2 Klasse zur parallelen Verwaltung einer Socketmenge (c_multiconn)

```
# -------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
# Dateiname       : c_multiconn.py
# Datum           : 03.11.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Klasse zur gleichzeitigen Verwaltung
#                   mehrerer TCP/IP Verbindungen
# Anmerkungen     :
#
# -------------------------------------------------------------

# imports
import socket
import select
import ERRNO
import errno


class c_multiconn:

      #
      # Konstruktor
      #
      def __init__(self):
        # initialisiere Liste der Filedeskriptoren
        self.fdlist=[]
        # initialisiere Dictionary fuer Sockets
        self.sockdict={}

      #
      # Gibt alle vorgehaltenen Informationen ueber
      # die Sockets aus. (for debugging)
      #
      def print_lists(self):
        print "---------------------------------"
        print self.fdlist
        print self.sockdict
```

```
    print "-----------------------------------"


  #
  # erzeugt einen neuen Socket mit der durch host, port
  # gegebene Adresse, werden host, port nicht uebergeben
  # wird eine freie Adresse benutzt
  #
  def open_sock(self,host="",port=0):
    conns=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    try:
      conns.bind(host,port)
      fd=conns.fileno()
      self.fdlist.append(fd)
      self.sockdict[fd]=conns
      return fd
    except:
      return -1

  #
  # schliesst den Socket mit Nummer socknr
  # und traegt ihn aus den Listen aus
  #
  def close_sock(self,socknr):
    sock=self.sockdict[socknr]
    try:
      sock.shutdown(1)
    except:
      sock.close()

    del sock
    self.fdlist.remove(socknr)
    del self.sockdict[socknr]
    return 1

  #
  # wartet auf allen Sockets auf Leseereignisse
  # und gibt eine Liste der aktiven Sockets zurueck
  #
  def wait_event(self,sec=0):
    active=([],[],[])
    if sec==0:
      try:
        active=select.select(self.fdlist,[],[])
      except select.error:
```

```
      pass
    else:
      active=select.select(self.fdlist,[],[],sec)
    return active[0]

  #
  # schreibt eine Nachricht auf den Socket mit Nummer socknr
  # und gibt die Anzahl der geschriebenen Bytes zurueck
  #
  def write_sock(self,socknr,sendstr):
    sock=self.sockdict[socknr]
    return sock.send(sendstr)

  #
  # liest eine Nachricht vom Socket mit Nummer socknr
  # und gibt sie zurueck
  #
  def read_sock(self,socknr):
    sock=self.sockdict[socknr]
    return sock.recv(64000)


  def poll_sock(self,socknr,secs):
    pass

  #
  # baut fuer Socket mit Nummer socknr eine Verbindung
  # mit der Adresse host, port auf
  #
  def connect(self,socknr,host,port):
    sock=self.sockdict[socknr]
    return sock.connect(host,port)

  def listen(self,socknr):
    sock=self.sockdict[socknr]
    return sock.listen(5)

  #
  # wartet an Socket mit Nummer socknr auf
  # ein connect und gibt das neue Socketobjet zurueck
  #
  def accept(self,socknr):
    sock=self.sockdict[socknr]
    conns, addr =sock.accept()
    fd=conns.fileno()
```

```
    self.fdlist.append(fd)
    self.sockdict[fd]=conns
    return fd

  #
  # Gibt die lokale Adresse des Sockets mit
  # der Nummer socknr zurueck
  #
  def get_localaddr(self,socknr):
    sock=self.sockdict[socknr]
    return sock.getsockname()

  #
  # Destruktor
  #
  def __del__(self):
    # schliesse alle offenen Sockets
    for i in self.fdlist:
      self.close_sock(i)
```

E.5.3 Klassen zur Verwaltung der Speicher und der Auftraege (c_joblist, c_orgalists)

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        :
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Hier werden zwei Klassen definiert, die
#                    die Verwaltung der angemeldeten Speicher und
#                    der laufenden Auftraege uebernehmen
# Anmerkungen      :
#
# ------------------------------------------------------------


# ------------------------------------------------------------
# Klasse zur Verwaltung der beim Broker angemeldeten Speicher
# Dabei wird zu jedem Speicher eine laufende Nummer, der Name
# des Speichers, die Adresse fuer zu sendende Auftraege
# (host,port) und der Zustand des Speichers gespeichert.
# ------------------------------------------------------------
class c_replist:

      #
      # Konstruktor
      #
      def __init__(self):
        # Variablen initialisieren
        self.count=0
        self.repdict={}
        self.IDLE="idle"
        self.BUSY="busy"
        self.PAUSED="paused"

        # Eintrag in repdict:
        # ( lfdnr, repname, hostname, port, state)

      #
      # fuegt einen neuen Speicher mit Namen name, der unter
```

```
  # host, port auf Auftraege wartet in die Liste ein.
  # Gibt die laufende Nummer innerhalb der Liste zurueck
  #
  def addrep(self,name,host,port):
    self.count=self.count+1
    self.repdict[self.count]=(self.count,name,host,port,
       self.IDLE)
    return self.count

  #
  # loescht den Speicher mit laufender Nummer repnr aus
  # der Liste
  #
  def removerep(self,repnr):
    if self.repdict.has_key(repnr):
      del self.repdict[repnr]
      return repnr
    else:
      return -1

  #
  # liefert die Nummer eines Speichers zurueck, der momentan
  # keinen
  # Auftrag verarbeitet und setzt den Zustand des Speichers
  # auf busy
  #
  def get_idlerep(self):
    i=0
    erg=(-1,"","",-1,-1)
    while i<=self.count:
      if self.repdict.has_key(i):
        nr,n,h,p,s=self.repdict[i]

        if s==self.IDLE:
          erg=(nr,n,h,p,s)
          del self.repdict[i]
          self.repdict[i]=(nr,n,h,p,self.BUSY)
          i=len(self.repdict)+1
      i=i+1
    return erg

  #
  # setzt den Zustand des Speichers mit der laufenden
  # Nummer repnr auf idle
  #
```

```
    def set_idle(self,repnr):
      if self.repdict.has_key(repnr):
        nr,n,h,p,s=self.repdict[repnr]
        del self.repdict[repnr]
        self.repdict[nr]=(nr,n,h,p,self.IDLE)
        return repnr
      else:
        return -1

    #
    # setzt den Zustand des Speichers mit der laufenden
    # Nummer repnr auf angehalten
    #
    def set_paused(self,repnr):
      if self.repdict.has_key(repnr):
        nr,n,h,p,s=self.repdict[repnr]
        del self.repdict[repnr]
        self.repdict[nr]=(nr,n,h,p,self.PAUSED)
        return repnr
      else:
        return -1

    #
    # gibt eine Liste alle in der Liste befindlichen
    # Speicher zurueck
    #
    def get_replist(self):
      return self.repdict

    #
    # sucht den Speicher mit Namen nam und gibt die repnr
    # zur"uck
    #
    def get_repbyname(self,nam):
      for i in self.repdict.keys():
        if self.repdict[i][1]==nam:
          return self.repdict[i][0]
      return -1


# imports fuer c_joblist
import string
# ----------------------------------------------------------------
# Klasse zur Verwaltung der offenen Auftraege.
```

```
# Die Groesse der Liste mu bei der Instanziierung mit angegeben
# werden, wenn sie nicht 50 sein soll.
# Die Klasse ermoeglicht die Zurordnung der Auftraege zu den
# bearbeitenden oder auftraggebenden Instanzen ueber Sockets
# -----------------------------------------------------------
class c_joblist:
      #
      # Konstruktor
      #
      def __init__(self,maxjobs=50):
        # initialisiere leere Listen und Dictionaries
        self.maxjobs=maxjobs
        self.ifdict={}
        self.repdict={}
        self.repnrdict={}
        #
        # e = empty, u =used
        #
        self.slots ="X"+ "e"* maxjobs
        #
        # e=empty, c=connected, w=waiting for reply, f=failed,
        # d=disconnected
        #
        self.ifstates ="X"+ "e"* maxjobs
        self.repstates ="X"+ "e"* maxjobs
        # clearing fd dictionaries
        for i in range(1,maxjobs+1):
          self.ifdict[i] = -1
          self.repdict[i] = -1
          self.repnrdict[i] = -1

        self.count=0

      #
      # fueht einen neuen Auftrag ein
      # if_fd   - Fildeskriptor des Sockets der
      # Schnittstellenkomponente
      # ifstate - unbenutzt
      # repnr   - laufende Nummer des Speichers aus der
      # Speicherliste
      # rep_fd  - Fildeskriptor des Sockets des Speichers der
      # den
      #           Auftrag ausfuehrt
      # repstate- Zustand des Speichers
      #
```

```
# Bei Fehler wird -1 zurueckgegeben, sonst die
# Auftragsnummer
#
def addjob(self,if_fd,ifstate,repnr,rep_fd=-1,repstate=""
  ):
  self.count=self.count+1
  if self.count>self.maxjobs:
    self.count=self.maxjobs
    return -1
  else:
    jnr=string.find(self.slots,"e")
    self.slots=self.slots[:jnr]+"u"+self.slots[jnr+1:]

    self.ifdict[jnr]=if_fd
    self.repdict[jnr]=rep_fd
    self.ifstates=self.ifstates[:jnr]+ifstate+
     self.ifstates[jnr+1:]
    self.repstates=self.repstates[:jnr]+repstate+
     self.repstates[jnr+1:]
    self.repnrdict[jnr]=repnr
    return jnr

#
# setzt den Zustand der Schnittstellenkomponente, der der
# Auftrag jobnr
# zugeordnet ist auf newstate (for debugging only)
#
def set_ifstate(self,jobnr,newstate):
  self.ifstates=self.ifstates[:jobnr]+newstate+
   self.ifstates[jobnr+1:]
  return 0

#
# setzt den Zustand des Speichers, dem der Auftrag jobnr
# zugeordnet ist auf newstate
#
def set_repstate(self,jobnr,newstate):
  self.repstates=self.repstates[:jobnr]+newstate+
   self.repstates[jobnr+1:]
  return 0

#
# loescht den Auftrag mit Nummer jobnr aus der Liste
#
def remjob(self,jobnr):
```

```
    self.slots=self.slots[:jobnr]+"e"+self.slots[jobnr+1:]

  #
  # Ermittelt den Auftrag, der der Verbindung mit
  # Filedeskriptor
  # if_fd zugeordnet ist und gibt die Nummer des Auftrags
  # zurueck.
  #
  def get_job_by_iffd(self,if_fd):
    erg=-1
    for i in self.ifdict.keys():
      if self.ifdict[i]==if_fd:
        erg=i
    return erg

  #
  # Ermittelt den Auftrag, der der Verbindung mit
  # Filedeskriptor
  # rep_fd zugeordnet ist und gibt die Nummer des Auftrags
  # zurueck.
  #
  def get_job_by_repfd(self,rep_fd):
    erg=-1
    for i in self.repdict.keys():
      if self.repdict[i]==rep_fd:
        erg=i
    return erg

  #
  # gibt ein dictionary zurueck das alle relevanten
  # Auftragsdaten
  # beinhaltet.
  #
  def get_joblist(self):
    i=0
    ergdict={}
    for i in range(1,self.maxjobs+1):
      ergdict[i]=(self.slots[i],self.ifdict[i],
         self.ifstates[i],self.repdict[i],
         self.repstates[i],self.repnrdict[jobnr])
    return ergdict

  #
  # gibt ein Dictionary zurueck, das alle laufenden
  # Auftraege beinhaltet
```

```
# in der Queue befindliche oder gestoppte Auftraege werden
# nicht
# beruecksichtigt
#
def get_runningjobs(self):
  i=0
  ergdict={}
  for i in range(1,self.maxjobs+1):
    if self.slots[i]!="e":
      ergdict[i]=(self.ifdict[i],self.ifstates[i],
         self.repdict[i],self.repstates[i])
  return ergdict

#
# gibt zu einer gegebenen jobnr alle relevanten
# Informationen
# zurueck
#
def get_jobdata(self,jobnr):
  return (jobnr,self.slots[jobnr],self.ifdict[jobnr],
     self.ifstates[jobnr],self.repdict[jobnr],
     self.repstates[jobnr],self.repnrdict[jobnr])
```

E.5.4 Funktionen zum Parsen der Konfigurationsdateien

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : confparse.py
# Datum            : 03.11.1997
# letzte Änderung  :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Funktion zum parsen der Konfigurationsdateien
#                    gibt eine Liste von Listen der gueltigen
#                    Zeilen der Datei zurueck
# Anmerkungen      :
#
# ---------------------------------------------------------------
def confparse(fname):
      import string
      parameters=[]

      # durchlaufe alle Zeilen der Konfigurationsdatei fname
      for line in open(fname,"r").readlines():
        # sortiere Kommentare und leere Zeilen aus
        if line[0]!="#"and len(line)>1:
          # jede Zeile in einer Liste ablegen
          llist=string.split(line)
          # und an die Parameterliste anhaengen
          parameters.append(llist)
      # Ergebnisliste zurueckgeben
      return parameters
```

E.6 Programmbibliothek für Dokumentenmethoden

E.6.1 Basisklassen und Funktionen

Klasse c_bsio

```
#!/usr/local/bin/python
# --------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname       : c_bsio.py
# Datum           : 03.11.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Eine einfache Klasse, die Methoden für
#                   Dateizugriffe bereitstellt
# Anmerkungen     :
#
# --------------------------------------------------------------


class c_bsio:
      # Dateiname und Filedekriptor init.
      fname=""
      fd=0

      #
      # Konstruktor
      # Initialisiert die Variablen
      # filename - Dateiname
      # mode     - Modus wie unter 'man fopen' beschrieben
      #
      def __init__(self,filename,mode="rw"):
        self.fname=filename
        self.mode=mode


      #
      # Öffnet die Datei
```

```
# gibt den Filedeskriptor
# der geöffneten Datei zurück
#
def open(self):
  self.fd=open(self.fname,self.mode)
  return self.fd


#
# Schliesst die Datei
#
def close(self):
  return self.fd.close()



#
# liest size Bytes aus der Datei und gibt sie zurueck
#
def read(self,size):
  return self.fd.read(size)


#
# Liest solange aus der Datei bis entweder
# ein Zeilenumbruch oder das Dateiende erreicht ist
#
def readline(self):
  return self.fd.readline()


#
# Schreibt einen String in die Datei
# und gibt die Anzahl der  geschriebenen Bytes zurueck

def write(self,buff):
  return self.fd.write(buff)


#
# Setzt den Dateizeiger auf eine neu Position
# Argumente       : offset - die Anzahl der Bytes um die
# der
# Dateizeiger bzgl. der von whence bestimmten
# Position verschoben werden soll.
```

```
# whence - 0 für den Anfang der Datei,
#          1 für die aktuelle Position
#          2 für das Ende der Datei

def seek(self,offset,whence):
  return self.fd.seek(offset,whence)


#
# Ermittelt die aktuelle Position des Dateizeigers
#

def tell(self):
  return self.fd.tell()
```

Klasse c_sgmlread

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_sgmlread.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse zum komfortablen Zugriff auf die
#                    Elemente eines als Bitstrom vorliegenden
#                    SGML-Dokuments
# Anmerkungen      : Diese Klasse wird von der Klasse c_bsio
#                    abgeleitet.
# -------------------------------------------------------------

# imports
import regex
import string
import c_bsio

class c_sgmlread(c_bsio.c_bsio):
      #
      # Konstruktor
      # fname gibt den Dateinamen des SGML-Dokuments an
      #
      def __init__(self,fname):
        # Konstruktor der Basisklasse
        c_bsio.c_bsio.__init__(self,fname,"r")
        # Oeffnen der Eingabedatei
        self.open()

      #
      # Destruktor
      #
      def __del__(self):
        # Datei schliessen
        self.close()
      #
      # Lese-/Schreibmarke auf den Dateianfang setzen
      #
      def seek_begin(self):
```

```
      self.fd.seek(0,0)

    #
    # Lese-/Schreibmarke auf das Dateiende setzen
    #
    def seek_end(self):
      self.fd.seek(0,2)

    #
    # liest das Doctype Tag des SGML-Dokuments aus
    # und gibt es zurueck
    #
    def get_doctype(self):
      self.oldpos=self.tell()
      self.seek(0,0)
      buff=" "
      erg=""
      while regex.search("<!DOCTYPE",buff)==-1 and buff!="":
        buff=self.fd.readline()

        if buff =="":
          erg= ""
        else:
          self.fd.seek(-len(buff),1)
          buff=""
          while buff!=">"and buff!="[":
            buff=self.read(1)
            erg=erg+buff

        if len(erg)>0 and erg[len(erg)-1]=="[":
          erg=erg[0:-2]
          erg=erg+">"
        self.seek(self.oldpos,0)
        return erg

    #
    # liefert den DTD Bezeichner des SGML-Dokuments zurueck
    #
    def get_dtdname(self):
      dtyp=self.get_doctype()
      return string.split(dtyp[1:-1])[1]
    #
    # liefert den Filenamen der DTD zurueck
    #
    def get_dtdfile(self):
```

```
    dtyp=self.get_doctype()
    return string.split(dtyp[1:-1])[3][1:-1]
  #
  # gibt eine Liste alle Attribute des Doctype-Tags zurueck
  #
  def get_dtd(self):
    import string
    doctypetag=self.get_doctype()
    doctypetag=doctypetag[2:-1]
    return string.split(doctypetag," ")

  #
  # sucht ausgehend von der aktuellen Position der
  # Lese-/Schreibmarke das naechste Tag mit Namen str
  #
  def get_tag(self,str):
    buff=" "
    while regex.search("<"+str+"*",buff)==-1 and
       regex.search("</"+str+"*",buff)==-1 and buff!="":
      buff=self.fd.readline()
      if buff!="":
        self.fd.seek(-len(buff)+regex.search("<"+str,
           buff)-1,1)
        if regex.search("</"+str+" ",buff)!=-1 or
           regex.search("</"+str+">",buff)!=-1:
          self.fd.seek(1,1)
          erg=""
          buff=""
          while buff!=">":
            buff=self.fd.read(1)
            erg=erg+buff
      else:
        erg=""

    return erg

  #
  # sucht ausgehend von der aktuellen Position der
  # Lese-/Schreibmarke das naechste Tag
  #
  def get_nexttag(self):
    buff=" "
    while regex.search("<",buff)==-1 and buff!="":
      buff=self.fd.readline()
```

```
    if buff!="":
      self.fd.seek(-len(buff)+regex.search("<",buff),1)
      # lese Taginhalt
      erg=""
      buff=""
      while buff!=">":
        buff=self.fd.read(1)
        erg=erg+buff
    else:
      erg=""
    return erg

  #
  # sucht ausgehend von der aktuellen Position der
  # Lese-/Schreibmarke das naechste Tag mit Namen str
  # und liefert den Inhalt des Tags zurueck
  #
  def get_tagcontents(self,str):
    erg=self.get_tag(str)
    if erg=="</"+str+">":
      erg=self.get_tag(str)
    erg1=""
    buff="␣"
    while regex.search("</"+str+"␣",buff)==-1 and
       regex.search("</"+str+">",buff)==-1 and buff!="":
      buff=self.fd.readline()
      erg1=erg1+buff

    erg1=erg1[:regex.search("</"+str+">",erg1)+len(str)+3]
    return erg+erg1

  #
  # teilt ein gegebenes Tag (tagstr) in den Tagnamen
  # und die einzelnen Attribute auf und liefert ein
  # Dictionaries zurueck, dessen Schluessel die
  # Attributnamen sind.
  #
  def split_tag(tagstr):
    ergdict={}

    # tagklammern entfernen
    tagstr=tagstr[1:-1]

    #hole tagname
    pos=string.find(tagstr,"␣")
```

```
tag=tagstr[:pos]
tagstr=tagstr[pos:]

# tagname in dictionary einfuegen
ergdict["TAG"]=tag

# hole attribute
while len(tagstr)!=0:

  # whitespace entfernen
  tagstr=string.strip(tagstr)

  # attributname holen
  pos1 = string.find(tagstr,"=")
  attrib = string.strip( tagstr[:pos1] )
  tagstr = tagstr[pos1+1:]

  # whitespace entfernen
  tagstr=string.strip(tagstr)

  # hole werte
  pos3 = string.find(tagstr,"␣")
  if pos3 == -1:
    pos3=len(tagstr)
  wert = string.strip( tagstr[:pos3] )
  tagstr = tagstr[pos3+1:]

  # attribute werte paar in dictionary einfuegen
  ergdict[attrib]=wert

return ergdict
```

Verwaltungsfunktionen

```
# ---------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1998
# Dateiname       : class_funcs.py
# Datum           : 09.12.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Hier stehen verschiedene Funktionen, die von
#                   den Paketen verwendet werden.
# Anmerkungen     :
#
# ---------------------------------------------------------------

# standard imports
import time
# globale Variable holen
from sc_globals import *

# Postgres interface laden
import pgpy

#
# gibt eine neue noch nicht benutzte lokale docid zurueck
#
def get_newdocid():
        conn = pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        if conn.connstatus() ==0:
            query="select docnr from local_doc_ids;"
            res = conn.exec_query(query)
            docid = res.get_fieldvalue(0,0)
            newdid = eval(docid) + 1
            query1="update local_doc_ids set docnr="+
              str(newdid)+";"
            res1 = conn.exec_query(query1)
            return docid
        elif conn.connstatus() == 1:
            return -1
#
# gibt eine neue dtd idnr zurueck
#
```

```
def get_newdtdid():
        conn = pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        if conn.connstatus() ==0:
            query="select dtdnr from local_doc_ids;"
            res = conn.exec_query(query)
            docid = res.get_fieldvalue(0,0)
            newdid = eval(docid) + 1
            query1="update local_doc_ids set dtdnr="+
             str(newdid)+";"
            res1 = conn.exec_query(query1)
            return docid
        elif conn.connstatus() == 1:
            return -1


# erzeugt einen eindeutigen temporaeren Dateinamen
def get_uniquename():
        time.sleep(1)
        return str(int(time.time()))

#
# ermittelt die dtdnr einer gegebenen doctype beschreibung,
# exisitert die dtd nicht in der db, so wird sie angelegt
#
def get_dtdnum(lst):
      dtdid=lst[1]
      dtdref=lst[2]
      dtddesc=lst[3]

      conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
      query="select dtdnr from dtd where dtdid='"+dtdid+"';"
      res=conn.exec_query(query)

      if res.get_tuple_count()==0:
        newid=get_newdtdid()
        query="insert into dtd values ("+newid+",'"+dtdid+"',"
        if dtdref=="SYSTEM":
          query=query+"'','"+dtddesc+"');"
        else:
          query=query+"'"+dtddesc+"','');"

        res=conn.exec_query(query)
        if res.commandstatus()[0:6] != "INSERT":
          print "FEHLER  bei Insert in dtd."
```

```
        return newid
      else:
        return res.get_fieldvalue(0,0)

#
# ermittelt den dtd filename einer gegebenen doctype beschreibung,
# exisitert die dtd nicht in der db, so wird -1 zurueck gegeben
#
def get_dtdfile(lst):
      dtdid=lst[1]
      dtdref=lst[2]
      dtddesc=lst[3]

      conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
      query="select * from dtd where dtdid='"+dtdid+"';"

      res=conn.exec_query(query)


      if res.get_tuple_count()!=0:
         return res.get_fieldvalue(0,3)
      else:
        return -1
```

E.6.2 Paket Bitstream (p_bitstream.py)

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_bitstream.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket Bitstream, fasst alle Bitstrompaket
#                    Klassen zusammen
#
# Anmerkungen      :
#
# -------------------------------------------------------------
import c_store
import c_bsio
import c_search
import c_glimpsearch
```

Klasse c_store

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_store.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse c_store bietet Methoden, um
#                    Dokumente in Form eines Bitstroms im
#                    Speicher abzulegen.
# Anmerkungen      : Die Variablen CURRENT_INFILE und NEWDOCID
#                    werden von der Ablaufumgebung gesetzt.
#
# -------------------------------------------------------------
```

```
#imports

import sc_globals
import class_funcs
CURRENT_INFILE="␣"
NEWDOCID=0

class c_store:

        infile=""
        outfile=""
        docid=0
        fd_in=0
        fd_out=0
        #
        # Konstruktor, setzt alle notwendigen Variablen
        # CURRENT_INFILE und NEWDOCID, werden von der
        # Ablaufumgebung gesetzt
        #
        def __init__(self):
            self.infile=sc_globals.DL_INPATH+
             CURRENT_INFILE
            #self.docid=class_funcs.get_newdocid()
            self.docid=NEWDOCID
            if self.docid== -1:
                print "Bekomme␣keine␣neue␣Doc␣Id."
                raise KeyboardInterrupt
            self.outfile=sc_globals.DL_BSOUTPATH+
             str(self.docid)+".bs"

        #
        # oeffnet den Eingabebitstrom
        #
        def open_in(self):
            self.fd_in=open(self.infile,"r")

        #
        # oeffnet den Ausgabebitstrom
        #
        def open_out(self):
            self.fd_out=open(self.outfile,"a+")
        #
        # schliesst den Eingabebitsrom
        #
        def close_in(self):
```

```
    self.fd_in.close()
#
# schliesst den Ausgabebitstrom
#
def close_out(self):
    self.fd_out.close()
#
# liest size Bytes aus dem Eeingabebitstrom
#
def read_in(self,size):
    return self.fd_in.read(size)
#
# schreibt outstr in den Ausgabebitstrom
#
def write_out(self,outstr):
    self.fd_out.write(outstr)

#
# setzt die Position der Lese-/Schreibmarke
# des Eingabebitstroms Syntax: wie Python seek Befehl
#
def seek_in(self,offset,whence):
    return self.fd_in.seek(offset,whence)

#
# setzt die Position der Lese-/Schreibmarke
# des Ausgabebitstroms Syntax: wie Python seek Befehl
#
def seek_out(self,offset,whence):
    return self.fd_out.seek(offset,whence)
#
# liefert die Position der Lese-/Schreibmarke
# des Eingabebitstroms zurueck
#
def tell_in(self):
    return self.fd_in.tell()

#
# liefert die Position der Lese-/Schreibmarke
# des Ausgabebitstroms zurueck
#
def tell_out(self):
    return self.fd_out.tell()

#
```

```
# vermerkt in den Metadaten des Dokuments, dass es als
# Bitstrom im Speicher abgelegt wurde.
#
def register_store(self):
  query="
    update doc_description set bs_stored='t' where docnr="
  query=query+str(self.docid)+";"
  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "Fehler bei Update in register_sgmlstore"

    return -1
  else:
    return 0
```

Klasse c_search

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_search.py
# Datum            : 09.12.1997
# letzte Änderung : 15.01.98
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse zum Volltextsuche in einem bestimmten
#                    Dokument
# Anmerkungen      :
#
# ---------------------------------------------------------------

class c_search:
      #
      # Konstruktor, initialisert Variable
      #
      def __init__(self,docid):
        from sc_globals import *
        self.dir=DL_FULLTEXT_DIR
        self.docid=docid

      #
      # Durchsucht das Dokument nach dem durch searchstr
      # gegebenen und/oder-verkn"upften␣Begriffen
      # Dabei steht ";␣fuer␣oder" und "&␣fuer␣und"
      # Ergebnis ist eine Liste von Zeilen, in denen
      # der Ausdruck vorkommt
      #
      def search(self,searchstr):
        slist=string.split(searchstr,"&")
        erglist=[]
        for i in slist:
          erglist.append(self.searchor(i))

        erg=[]
        for i in erglist[0]:
          flag=1
          for j in erglist[1:]:
            if not (i in j):
```

```
          flag=0
          break
      if flag==1:
        erg.append(i)

    return erg

  #
  # Wei search nur werden lediglich veroderter
  # Ausdruecke verabreitet
  #
  def searchor(self,searchstr):
    slist=string.split(searchstr,";")
    fd=open(self.dir+self.docid,"r")
    buff="␣"
    while buff!="":
      buff=fd.readline()
      for i in slist:
        if string.find(buff,i):
          erg.append(buff)
    fd.close()
    return erg
```

Klasse c_glimpsearch

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_glimpsearch.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Eine Klasse um mit glimpse eine Volltext-
#                    recherche durchzufuehren.
# Anmerkungen      :
#
# -------------------------------------------------------------

class c_glimp_search:
      #
      # Konstruktor, initialisert Variable
      #
      def __init__(self,sparams):
        import string
        from sc_globals import *
        self.dir=DL_FULLTEXT_DIR
        self.params=sparams
        self.resfile=DL_FULLTEXT_RESULTS

      #
      # Durchsucht alle als Bitstrom gespeicherten Dokumente
      # nach dem durch searchstr
      # gegebenen und/oder-verkn"upften␣Begriffen
      # Dabei steht ";␣fuer␣oder" und "&␣fuer␣und"
      # Ergebnis ist eine Liste von Zeilen, in denen
      # der Ausdruck vorkommt
      #
      def glimp_search(self,searchstr):
        slist=string.split(searchstr,"&")
        erglist=[]
        for i in slist:
          erglist.append(self.glimp_searchor(i))

        erg=[]
        for i in erglist[0]:
```

```
      flag=1
      for j in erglist[1:]:
        if not (i in j):
          flag=0
          break
      if flag==1:
        erg.append(i)

    return erg

  #
  # Wie glimp_search es werden aber lediglich veroderter
  # Ausdruecke verarbeitet
  #
  def glimp_searchor(self,searchstr):
    import os
    erg=[]
    os.system("glimpse␣-y␣-H␣"+self.dir+"␣"+self.params+"␣"
       +searchstr+"␣>"+self.resfile)
    fd=open(self.resfile,"r")
    buff="␣"
    while buff!="":
      buff=fd.readline()
      erg.append(buff[:-1])

    fd.close()
    os.unlink(self.resfile)
    return erg[:-1]
```

E.6.3 Paket SGML (p_sgml.py)

```
# -------------------------------------------------------------
# Projekt           : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname         : p_sgml.py
# Datum             : 03.11.1997
# letzte Änderung :
# Autor             : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache           : Python v1.4
# Beschreibung      : Paket SGML, fasst alle SGML-Paket Klassen
#                     zusammen
#
# Anmerkungen       :
#
# -------------------------------------------------------------
import c_sgmlquery
import c_sgmlread
import c_sgmlsearch
import c_sgmlstore
```

Klasse c_sgmlquery

```
# -------------------------------------------------------------
# Projekt           : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1998
# Dateiname         : c_sgmlquery.py
# Datum             : 09.12.1997
# letzte Änderung :
# Autor             : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache           : Python v1.4
# Beschreibung      : Eine Klasse die SGML spezifische Abfragen auf
#                     einem bestimmten Dokument durchfuehrt.
# Anmerkungen       :
# -------------------------------------------------------------

# imports
from sc_globals import *
import pgpy
```

```
class c_sgmlquery:
      #
      #  Konstruktor
      #
      def __init__(self,ident):
        self.ident=ident
        # Datenbankverbindungsobjekt instanziieren
        self.pgconn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)

      #
      # liefert das tagcount'e Tag mit Namen tagname zurueck
      #
      def sgmlquery_tag(self,tagname,tagcount):
        query="select␣*␣from␣doc_tags␣where␣docnr="+
         str(self.ident)
        query=query + "␣and␣contents␣ *␣'"+tagname+
         "*'␣order␣by␣elementnr;"

        res=self.pgconn.exec_query(query)
        if res.get_tuple_count()>0:
          return [ res.get_fieldvalue(tagcount-1,
             res.get_fieldindex_byname("contents")),
             res.get_fieldvalue(tagcount-1,
             res.get_fieldindex_byname("elementnr")) ]
        else:
          return ["",0]
      #
      # liefert den Inhalt des tagcount'en Tags
      # mit Namen tagname zurueck
      #
      def sgmlquery_tagcontents(self,tagname,tagcount):
        pos1=self.sgmlquery_tag(tagname,tagcount)[1]
        query1="select␣contents,
           elementnr␣from␣doc_tags␣where␣docnr="+str(self.ident)
        query1=query1 + "␣and␣elementnr>"+str(pos1)
        query1=query1 +"and␣contents␣ *␣'</"+tagname+
         "'␣order␣by␣elementnr;"
        res1=self.pgconn.exec_query(query1)
        if res1.get_tuple_count>0:
          pos2=res1.get_fieldvalue(0,1)

          query2= "select␣contents,
             elementnr␣from␣doc_elements*␣where␣docnr="
          +str(self.ident)
```

```
      query2=query2+"␣and␣elementnr>="+str(pos1)
      query2=query2+"␣and␣elementnr<="+str(pos2)+
        "␣order␣by␣elementnr;"

      res2=self.pgconn.exec_query(query2)
      if res2.get_tuple_count>0:
        j=res2.get_tuple_count()
        i=0
        erg=""
        while i < j:
          erg=erg+str(res2.get_fieldvalue(i,0))+"\n"
          i=i+1

        return erg[:-1]
      else:
        return ""
    else:
      return ""

  #
  # liefert den Strukturbaum als ASCII Text zurueck
  #
  def sgmlquery_structure(self):
    query="select␣contents,
      elementnr␣from␣doc_tags␣where␣docnr="+ str(self.ident) +
      "␣order␣by␣elementnr;"

    res=self.pgconn.exec_query(query)
    j=res.get_tuple_count()
    i=0
    erg=""
    while i < j:
      erg=erg+str(res.get_fieldvalue(i,0))+"\n"
      i=i+1
    return erg[:-1]

  #
  # gibt das gesamte Dokument zurueck
  #
  def sgmlquery_doc(self):
    query="select␣contents,
      elementnr␣from␣doc_elements*␣where␣docnr="+str(self.ident)+
     "␣order␣by␣elementnr;"

    res=self.pgconn.exec_query(query)
```

```
  j=res.get_tuple_count()
  i=0
  erg=""
  while i < j:
    erg=erg+str(res.get_fieldvalue(i,0))+"\n"
    i=i+1
  return erg

#
# gibt die Anzahl der Tags mit Namen tagname zurueck
#
def sgmlquery_tagcount(self,tagname):
  query="select␣contents␣from␣doc_tags␣where␣docnr="+
   str(self.ident)+"␣and␣contents␣ *␣'"+tagname+
   "';"
  res=self.pgconn.exec_query(query)
  return res.get_tuple_count()

#
# teilt ein gegebenes Tag (tagstr) in den Tagnamen
# und die einzelnen Attribute auf und liefert ein
# Dictionaries zurueck, dessen Schluessel die
# Attributnamen
# sind.
#
def sgmlquery_splittag(self,tagstr):
  import string

  ergdict={}
  t=""
  for j in range(len(tagstr)):
    if tagstr[j]!="\012":
      t=t+tagstr[j]

  tagstr=t

  # tagklammern entfernen
  tagstr=tagstr[1:-1]

  #hole tagname
  pos=string.find(tagstr,"␣")
  tag=tagstr[:pos]
  tagstr=tagstr[pos:]

  # tagname in dictionary einfuegen
```

```
ergdict["TAG"]=tag

#hole attribute
while len(tagstr)!=0:

  # whitespace entfernen
  tagstr=string.strip(tagstr)

  # attributname holen
  pos1 = string.find(tagstr,"=")
  attrib = string.strip( tagstr[:pos1] )
  tagstr = tagstr[pos1+1:]

  # whitespace entfernen
  tagstr=string.strip(tagstr)

  # hole werte
  if len(tagstr)>0:
    if tagstr[0]=="\'":
      tagstr=tagstr[1:]
      pos3 = string.find(tagstr,"\'")
    else:
      pos3 = string.find(tagstr," ")
  else:
    pos3 = string.find(tagstr," ")

  if pos3 == -1:
    pos3=len(tagstr)
  wert = string.strip( tagstr[:pos3] )

  tagstr = tagstr[pos3+1:]

  # attribute werte paar in dictionary einfuegen
  ergdict[attrib]=wert

return ergdict
```

Klasse c_sgmlsearch

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_sgmlsearch.py
# Datum            : 09.12.1997
# letzte Änderung : 15.01.1998
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Dies Klasse realisiert ein Suchinterface fuer
#                    in der Datenbank abgelegte SGML Dokumente
# Anmerkungen      :
#
# ------------------------------------------------------------

# imports
import pgpy

class c_sgmlsearch:
      #
      # Konstruktor
      #
      def __init__(self):
        # Verbindungsobjekt zur Datenbank instanziieren
        self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)

      #
      # Durchsucht alle in der Datenbank gespeicherten
      # Tags nach searchstr und findet auch aehnliche
      # Ausdruecke.
      # Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
      # zurueckgegeben
      #
      def sgmlsearch_tag(self,searchstr):
        # SQL Abfrage zusammenbauen
        query="select␣*␣from␣doc_tags␣where␣contents␣ *"+
         searchstr+";"
        # Abfrage ausfuehren
        res=self.conn.exec_query(query)
        # Fehler aufgetreten ?
        if res.get_tuple_count()==0:
          # wenn ja, -1 zurueckgeben
```

```
    return -1
  else:
    #wenn nein Ergebnis zurueckgeben
    erg=res.get_resultlist(1,"|",1,0)
    return erg

#
# Durchsucht die Text Klasse der Datenbank
# nach searchstr und findet auch aehnliche
# Ausdruecke.
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def sgmlsearch_text(self,searchstr):
  # SQL Abfrage zusammenbauen
  query="select * from doc_text where contents  *"+
   searchstr+";"
  # Abfrage ausfuehren
  res=self.conn.exec_query(query)
  # Fehler aufgetreten ?
  if res.get_tuple_count()==0:
    # wenn ja, -1 zurueckgeben
    return -1
  else:
    #wenn nein Ergebnis zurueckgeben
    erg=res.get_resultlist(1,"|",1,0)
    return erg

#
# virtuelle Methode, muss in einer abgeleiteten
# Klasse ueberschrieben werden und ist zum Durchsuchen
# von Binaerdaten gedacht ;-)
#
def sgmlsearch_bin(self):
      pass

#
# Durchsucht die Tag- und die Text- Klasse der Datenbank
# nach searchstr und findet auch aehnliche
# Ausdruecke.
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben.
#
def sgmlsearch_doc(self,searchstr):
  erg=[]
```

```
  erg1=self.sgmlsearch_tag(searchstr)
  erg2=self.sgmlsearch_text(searchstr)
  if erg1!=-1:
    erg.append(erg1)
  if erg2!=-1:
    erg.append(erg2)
  return erg

#
# ermittelt die Dokumenten-ID's aller Dokumente, die der
# in dtdid gegebenen DTD entsprechen. Dabei kann dtdid
# die Public- oder die System-ID einer DTD angeben.
# Das Ergebnis wird in Form einer Lsite zur"uckgeliefert.
#
def sgmlsearch_dtddocs(self,dtdid):
  # DTD-Nummer ermitteln
  query="select dtdnr from dtd where publicname ="+
     dtdid + "or systemname ="+dtdid+";"
  # Abfrage ausfuehren
  res=self.conn.exec_query(query)
  # Fehler aufgetreten ?
  if res.get_tuple_count()==0:
    # wenn ja, -1 zurueckgeben
    return []
  else:
    #wenn nein Ergebnis zurueckgeben
    dtdnr=res.get_fieldvalue(1,0)

  # Doc-ID's holen
  query="select docnr from doc_description where dtdnr="
     + str(dtdnr)+";"
  # Abfrage ausfuehren
  res=self.conn.exec_query(query)
  # Fehler aufgetreten ?
  if res.get_tuple_count()==0:
    # wenn ja, -1 zurueckgeben
    return []
  else:
    #wenn nein Ergebnis zurueckgeben
    erg=res.get_resultlist(1,"|",1,0)
    return erg
#
# ermittelt die Dokumenten-ID's aller Dokumente, die
# der durch dtdid gegebenen DTD entsprechen und ein Tag
# enthalten das searchstr entspricht. Das Ergebnis wird
```

```
# in Form einer Liste zur"uckgegeben.
#
def sgmlsearch_dtdandtags(self,dtdid,searchstr):
  # ermittle Teilergebnisse
  erg1=self.sgmlsearch_tag(searchstr)
  erg2=self.sgmlsearch_dtddocs(dtdid)

  erg=[]
  for i in erg1:
    if i in erg2:
      erg.append(i)

  return erg
```

Klasse c_sgmlstore

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_sgmlstore.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Diese Klasse stellt Methoden bereit, um als
#                    Bitstrom vorliegende SGML-Dokumente in der
#                    Datenbank zu speichern
# Anmerkungen      : Diese Klasse wird von der Klasse c_sgmlread
#                    abgeleitet. Die Varibalen DL_CURRENT_INFILE
#                    und DL_NEW_DOCID werden von der Ablauf-
#                    umgebung gesetzt.
# ---------------------------------------------------------------

# imports
from sc_globals import *
import class_funcs
import pgpy
import c_sgmlread
import urllib
import regsub


class c_sgmlstore(c_sgmlread.c_sgmlread):
      # Variablen
      infile=""
      docid=0
      fd_in=0
      emptytags=[]
      dtdfname=""
      # Variablen fuer Strukturbaumberechnungen
      atag,ntag,atyp,ntyp="","","",""
      anum,nnum=0,0
      stack=[]
      element_counter=0
      ls=("",0)
```

```
#
# Konstruktor
#
def __init__(self,DL_CURRENT_INFILE,DL_NEW_DOCID):
  # uebernehme Parameter
  self.infile=DL_CURRENT_INFILE
  self.docid=DL_NEW_DOCID

  # Konstruktur der Basisklasse
  c_sgmlread.c_sgmlread.__init__(self,self.infile)

  # hole Liste der EMPTY Tags
  self.dtdfname=class_funcs.get_dtdfile(self.get_dtd())
  self.emptytags_lst=self.get_emptytags(self.dtdfname)

  # baue Verbindung zu Postgres auf
  self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
  if self.conn.connstatus()!=0:
    print "
      Postgresverbindung konnte nicht aufgebaut werden."
    raise KeyboardInterrupt

#
# oeffnet den eingabebitstrom
#
def open_in(self):
  self.fd_in=open(self.infile,"r")
#
# schliesst den Eingabebitstrom
#
def close_in(self):
  self.fd_in.close()
#
# liest size Bytes vom Eingabebitstrom
#
def read_in(self,size):
  return self.fd_in.read(size)

#
# setzt die aktuelle Position der Lese-/Schreibmarke
# des Eingabebitstroms (Syntax wie Python Befehl seek)
#
#
def seek_in(self,offset,whence):
  self.fd_in.seek(offset,whence)
```

```
#
# liefert die aktuelle Position der Lese-/Schreibmarke
# des Eingabebitstroms zurueck
#
def tell_in(self):
  return self.fd_in.tell()

#
# filtert in der Datenbank abzulegende Text, um sie an
# die PostgreSQL Syntax anzupassen
#
def pgfilter(self,fstr):
  erg=""
  for i in range(0,len(fstr)):
    # ersetze anfuehrungszeichen
    if fstr[i]=="'":
      erg=erg+"\\"

    erg=erg+fstr[i]
  return erg


#
# liefert eine Liste alle in der DTD fname definierten
# EMPTY-Tags zurueck
#
def get_emptytags(self,fname):
  import c_bsio
  import string

  fd=c_bsio.c_bsio(fname,"r")
  erglst=[]
  fd.open()
  buff = "␣"
  while buff!="":
    buff=fd.readline()
    if
       string.upper(string.strip(buff)[:9])=="<!ELEMENT":
      if string.find(buff,"EMPTY")!=-1:
        erglst.append(string.upper(
           string.strip(buff)[10:string.find(
           string.strip(buff),"-")-1]))
  fd.close()
  return erglst
```

```
#
# vermerkt in der Datenbank, dass das Dokument in der
# Datenbank
# gespeichert wurde
#
def register_sgmlstore(self):
  query="
    update␣doc_description␣set␣sgml_stored='t'␣where␣docnr="
  query=query+str(self.docid)+";"
  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "Fehler␣bei␣Update␣in␣register_sgmlstore"
    return -1
  else:
    return 0

#
# vermerkt die DTD der das Dokument angehoerig ist in
# der Datenbank
#
def save_doctype(self):
  dtdtup=self.get_dtd()
  dtdnum=class_funcs.get_dtdnum(dtdtup)

  query="update␣doc_description␣set␣dtdnr="
  query=query+str(res.get_fieldvalue(1,0))
  query=query+"␣where␣docnr="+str(self.docid)+";"

  res=self.conn.exec_query(query)
  if res.commandstatus[0:6]!="UPDATE":
    print "
      Fehler␣c_sgmlstore.save_doctype␣beim␣Eintragen␣der␣DTDNR"
#
# speichert tagst in der Tag-Klasse der Datenbank
# und berechnet die Strukturbaumverweise automatisch
#
def save_astag(self,tagstr):
  dbclass="doc_tags"
  self.element_counter=self.element_counter+1

  self.nnum=self.element_counter
  self.ntag=tagstr
  isempty=self.is_emptytag(self.ntag)
```

```
# Berechnungen fuer Strukturbaum

if self.ntag[0:2]!="</":
  self.ntyp="S"
  self.stack.append((self.ntag,self.nnum))
else:
  self.ntyp="E"
  self.ls=self.stack[-1]
  self.stack=self.stack[:-1]

if self.ntyp=="S"and self.atyp=="S":
  # atag ist Vater von ntag
  query="update doc_tags set son="+str(self.nnum)
  query=query+" where docnr="+str(self.docid)
  query=query+" and elementnr="+str(self.anum)+";"

  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "FEHLER in c_sgmlstore.save_astag Query 1"


if self.ntyp=="S"and self.atyp=="E":
  # ntag ist rechter Bruder von anum
  query="update doc_tags set brother="+
    str(self.nnum)
  query=query+" where docnr="+str(self.docid)
  query=query+" and elementnr="+str(self.anum)+";"

  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "
      FEHLER in c_sgmlstore.save_astag Query 2"

if self.ntyp=="E"and self.atyp=="S":
  # atag ist ein Blatt
  query="update doc_tags set brother="+
    str(self.nnum)
  query=query+" where docnr="+str(self.docid)
  query=query+" and elementnr="+str(self.anum)+";"

  res=self.conn.exec_query(query)
  if res.commandstatus()[0:6]!="UPDATE":
    print "
      FEHLER in c_sgmlstore.save_astag Query 3"
```

```
    if self.ntyp=="E"and self.atyp=="E":
      # atag ist ein Blatt
      query="update␣doc_tags␣set␣brother="+
       str(self.nnum)
      query=query+"␣where␣docnr="+str(self.docid)
      query=query+"␣and␣elementnr="+str(self.ls[1])+";"

      res=self.conn.exec_query(query)
      if res.commandstatus()[0:6]!="UPDATE":
        print "
          FEHLER␣in␣c_sgmlstore.save_astag␣Query␣4"


    # letztes  Tag und Typ merken
    self.atag=self.ntag
    self.anum=self.nnum
    if isempty==0:
      self.atyp=self.ntyp
    elif isempty==1:
      self.atyp="E"
      self.stack=self.stack[:-1]

    # neuen Tagdatensatz in DB eintragen
    tagstr=self.pgfilter(tagstr)
    query="insert␣into␣"+dbclass+"␣values␣("+
     str(self.docid)+","
    query=query+str(self.nnum)+",0,0,'"+tagstr+"',"
    query=query+str(0)+");"

    res=self.conn.exec_query(query)
    if res.commandstatus()[0:6]!="INSERT":
      print "FEHLER␣in␣c_sgmlstore.save_astag␣Query␣5"

  #
  # speichert textstr in der Text-Klasse der Datenbank
  #
  def save_astext(self,textstr):
    dbclass="doc_text"
    self.element_counter=self.element_counter+1
    textstr=self.pgfilter(textstr)
    query="insert␣into␣"+dbclass+"␣values␣("+
     str(self.docid)+","
    query=query+str(self.element_counter)+",0,0,'"+textstr+
     "');"
```

```
  res=self.conn.exec_query(query)
  erg=res.commandstatus()
  if str(erg)[0:6]!="INSERT":
    print "Fehler bei INSERT in Postgres"
    raise KeyboardInterrupt
  return 0
#
# speichert das unter der URL urlstr abrufbare Objekt
# als binaere Daten in der Datenbank
#
def save_asbin(self,urlstr):
  dbclass="doc_bin"
  self.element_counter=self.element_counter+1
  binstr=urllib.urlretrieve(urlstr)
  query="insert into "+dbclass+" values ("+
   str(self.docid)+","
  query=query+str(self.element_counter)+",0,0,'"+
   binstream+"');"

  res=self.conn.exec_query(query)
  erg=res.commandstatus()
  if str(erg)[0:6]!="INSERT":
    print "Fehler bei INSERT in Postgres"
    raise KeyboardInterrupt
  return 0
#
# liest das naechste Tag aus dem Eingabebitstrom und
# gibt es zurueck
#
def get_nextelement(self):
  # leerzeilen ueberspringen
  test1=""
  test1=self.read(1)
  while test1=="\n":
    test1=self.read(1)

  # position des Lese-/Schreibmarke korrigieren
  if test1!="":
    if self.tell()>1:
      self.seek(-1,1)

      # sicherstellen das nicht mitten im Tag gesucht
      # wird
      if test1=="<":
        return self.get_nexttag()
```

```
    else:
      erg=""
      buff=" "
      while buff!="<"and buff!="":
        buff=self.read(1)
        erg=erg+buff

      # letztes Zeichen wegwerfen
      if len(erg)>1:
        erg=erg[0:-1]
      # position des Lese-/Schreibmarke korrigieren
      self.seek(-1,1)
      return erg
  else:
    return ""
#
# ermittelt, ob es sich bei tagstr um ein Tag handelt
# gibt 1 zureuck wenn ja, 0 wenn nein und -1 bei Fehler
#
def is_tag(self,tagstr):
  if len(tagstr)!=0:
    if tagstr[0]=="<"and tagstr[len(tagstr)-1]==">":
      return 1
    else:
      return 0
  else:
    return -1
#
# ermittelt, ob es sich bei tagstr um ein Empty-Tag
# handelt
# gibt 1 zureuck wenn ja, 0 wenn nein und -1 bei Fehler
#
def is_emptytag(self,tagstr):
  import string
  if len(tagstr)!=0:
    if tagstr[1:string.find(tagstr," ")] in
       self.emptytags_lst:
      return 1
    else:
      return 0
  else:
    return -1
```

E.6.4 Paket System (p_system.py)

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_system.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket System, fasst alle Systempaket Klassen
#                    zusammen
#
# Anmerkungen      :
#
# ------------------------------------------------------------

import c_metadoc
import c_metasearch
import c_metaquery
```

Klasse c_metadoc

```
# ------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_metadoc.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Die klasse c_metadoc stellt Methoden zur
#                    Verfuegung um Informationen aus unseren Meta
#                    Docs zu extrahieren
# Anmerkungen      : Diese Klasse wird von der Klasse
#                    c_sgmlread abgeleitet.
# ------------------------------------------------------------

# imports
```

```
import c_sgmlread
import regex
import string
import regsub
from sc_globals import *


class c_metadoc(c_sgmlread.c_sgmlread):
      #
      # Konstruktor, oeffnet das Metadokument
      #
      def __init__(self,fname):
        # Konstruktor der Basisklasse aufrufen
        c_sgmlread.c_sgmlread.__init__(self,fname)
        self.open()

      #
      # liefert eine Liste der Attribute des DOCTYPE
      # Tags zurueck
      #
      def get_dtd(self):
        import string
        self.seek_begin()
        doctypetag=self.get_doctype()
        doctypetag=doctypetag [2:-1]
        return string.split(doctypetag," ")
      #
      # liefert den Inhalt des kompletten
      # HEADER Tags zurueck
      #
      def get_header(self):
        self.seek_begin()
        return self.get_tagcontents("HEADER")
      #
      # liefert den Inhalt des INFO Tags zurueck
      #
      def get_info(self):
        self.seek_begin()
        return self.get_tagcontents("INFO")
      #
      # liefert ein Dictionary mit allen Attributen und ihren
      # Werten des PROPERTY Tags zurueck
      #
      def get_property(self):
        self.seek_begin()
```

```
    erg={}
    buff = self.get_tag("PROPERTY")

    pos1=regex.search("title=*\'",buff)
    pos2=regex.search("author=*\'",buff)
    pos3=regex.search("owner=*\'",buff)
    pos4=regex.search("ident=*\'",buff)
    pos5=regex.search("pubkey=*\'",buff)
    title =string.strip(buff[pos1:pos2])
    author=string.strip(buff[pos2:pos3])
    owner =string.strip(buff[pos3:pos4])
    ident =string.strip(buff[pos4:pos5])
    pubkey=string.strip(buff[pos5:])

    title =title[string.find(title,"\'")+
     1:string.rfind(title,"\'")]
    author=author[string.find(author,"\'")+
     1:string.rfind(author,"\'")]
    owner =owner[string.find(owner,"\'")+
     1:string.rfind(owner,"\'")]
    ident =ident[string.find(ident,"\'")+
     1:string.rfind(ident,"\'")]
    pubkey=pubkey[string.find(pubkey,"\'")+
     1:string.rfind(pubkey,"\'")]

    erg["title"] =title
    erg["author"]=author
    erg["owner"] =owner
    erg["ident"] =ident
    erg["pubkey"]=pubkey
    return erg
  #
  # liefert ein Dictionary alle dem Dokument zugeordneten
  # Methoden und ihrer Namen, sowie MIME-Types zurueck
  #
  def get_methods(self):
    self.seek_begin()
    erg={}
    buff = self.get_tag("METHOD")
    while buff!="":
      pos1=regex.search("name=*\'",buff)
      pos2=regex.search("code=*\'",buff)
      pos3=regex.search("mime=*\'",buff)
      name=string.strip(buff[pos1:pos2])
      code=string.strip(buff[pos2:pos3])
```

```
      mime=string.strip(buff[pos3:])

      name=name[string.find(name,"\'")+
       1:string.rfind(name,"\'")]
      code=code[string.find(code,"\'")+
       1:string.rfind(code,"\'")]
      mime=mime[string.find(mime,"\'")+
       1:string.rfind(mime,"\'")]

      erg[name]=[name,code,mime]
      buff = self.get_tag("METHOD")

    return erg
  #
  # liefert den Inhalt des Data-Tags zurueck
  #
  def get_data(self):
    self.seek_begin()
    buff= self.get_tagcontents("DATA")
    return string.strip(buff[string.find(buff,">")+
     1:string.rfind(buff,"</")])

  #
  # Destruktor, schliesst das Metadokument
  #
  def __del__(self):
    self.close()
```

Klasse c_metasearch

```
# ---------------------------------------------------------------
# Projekt           : Digitale Bibliotheken Projekt
#                      Uni-Frankfurt/M, Professur Telematik und
#                      verteilte Systeme, Prof. O. Drobnik
#                      Diplomarbeit, Matzen,Hans, 1998
# Dateiname         : c_metasearch.py
# Datum             : 09.12.1997
# letzte Änderung   :
# Autor             : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache           : Python v1.4
# Beschreibung      : Diese Klasse realisiert ein einfaches
#                     Suchinterface ueber alle gespeicherten
#                     Metadaten
# Anmerkungen       :
# ---------------------------------------------------------------
# imports
import pgpy

class c_metasearch:
      #
      # Konstruktor
      #
      def __init__(self):
        # Verbindungsobjekt zur Datenbank instanziieren
        self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)

      #
      # Durchsucht die in der Datenbank gespeicherten
      # Metadaten nach searchstr und betrachtet dabei
      # jeweils nur das durch field gegeben Feld in der
      # doc_description Klasse. Es werden auch aehnliche
      # Ausdruecke gefunden.
      # Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
      # zurueckgegeben
      #
      def metasearch(self,field,searchstr):
        # SQL Abfrage zusammenbauen
        query="select * from doc_description where "+field+" *"
          +searchstr+";"
        # Abfrage ausfuehren
        res=self.conn.exec_query(query)
        # Fehler aufgetreten ?
        if res.get_tuple_count()==0:
```

```
    # wenn ja, -1 zurueckgeben
    return -1
  else:
    #wenn nein Ergebnis zurueckgeben
    erg=res.get_resultlist(1,"|",1,0)
    return erg
#
# sucht nach ident aehnlichen Dokumenten-ID's
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_ident(self,ident):
  return self.metasearch("ident",searchstr)
#
# sucht nach searchstr aehnlichen INFO-Feldern
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_info(self,searchstr):
  return self.metasearch("info",searchstr)
#
# sucht nach searchstr aehnlichen AUTHOR-Feldern
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_author(self,searchstr):
  return self.metasearch("author",searchstr)
#
# sucht nach searchstr aehnlichen OWNER-Feldern
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_owner(self,searchstr):
  return self.metasearch("owner",searchstr)
#
# sucht nach searchstr aehnlichen TITLE-Feldern
# Als Ergebnis wird eine ASCII Liste von Dokumenten ID's
# zurueckgegeben
#
def metasearch_title(self,searchstr):
  return self.metasearch("title",searchstr)
```

Klasse c_metaquery

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_metaquery.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse fuer Abfragen der Metadaten eines
#                    bestimmten Dokuments
# Anmerkungen      :
#
# ---------------------------------------------------------------

# imports
from sc_globals import *
import pgpy

class c_metaquery:
      #
      # Konstruktor, setzt Variable
      # ident gibt die Dokuent-ID des Dokuments an
      #
      def __init__(self,ident):
        # Verbidungsobjekt zur Datenbank instanziieren
        self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        self.ident=ident

      #
      # liefert den Inahlt des INFO Tags des gespeicherten
      # Dokuments zurueck
      #
      def mquery_info(self):
        query="select info from doc_description where ident="+
          str(self.ident)+";"
        res=conn.exec_query(query)
        return res.get_fieldvalue(0,0)

      #
      # liefert ein Dictionary mit den Attributen und ihren
      # Werten
```

```
  # des PROPERTY Tags zurueck
  #
  def mquery_property(self):
    query="select * from doc_description where ident="+
      str(self.ident)+";"
    res=conn.exec_query(query)
    erg={'author':'','ident':'','owner':'','pubkey':'',
      'title':''}
    erg['author']=res.get_fieldvalue(0,3)
    erg['ident'] =res.get_fieldvalue(0,5)
    erg['owner'] =res.get_fieldvalue(0,4)
    erg['pubkey']=res.get_fieldvalue(0,6)
    erg['title'] =res.get_fieldvalue(0,2)

    return erg

  #
  # liefert ein Dictionary mit allen zu den Methoden
  # gespeicherten Informationen (MIME-Typ, Name, URL)
  # zurueck
  #
  def mquery_methods(self):
    erg={}

    query="select * from doc_methods where ident="+
      str(self.ident)+";"
    res=conn.exec_query(query)

    count=res.get_tuple_count()
    j=0
    while j<count:
      mclass=res.get_fieldvalue(j,1)
      mloc  =res.get_fieldvalue(j,2)
      mtyp  =res.get_fieldvalue(j,3)

      erg[mclass]= [mclass,mloc,mtyp]
      j=j+1

    return erg
```

E.6.5 Paket Maintenance (p_maintenance.py)

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_maintenance.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket Maintenance, fasst alle Maintenance-Paket
#                    Klassen zusammen
#
# Anmerkungen      :
#
# ---------------------------------------------------------------
import c_maintenance
import c_glimpindex
```

Klasse c_maintenace

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_maintenance.py
# Datum            : 09.12.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Diese Klasse realisiert ein paar Methoden
#                    zur Pflege des Systems
# Anmerkungen      :
# ---------------------------------------------------------------

#imports
from sc_globals import *
import pgpy
```

```
class c_maintenance:
      #
      # Reorganisert die Postgresdatenbank
      #
      def vacuum(self):
        self.conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        res=self.conn.exec_query("vacuum;")
        erg=res.commandstatus()
        if str(erg)[0:6]!="VACUUM":
          return -1
        else:
          return 0

      #
      # loescht ein als Bitstrom gespeichertes Dokument
      #
      def delete_bs(self,ident):
        import os
        os.system("rm␣"+"-f␣"+DL_BSOUTPATH+str(ident)+".bs")
        conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        res=self.conn.exec_query(
          "
            update␣doc_description␣set␣bs_stored='f'␣where␣docnr="+
          str(ident)+";")
        erg=res.commandstatus()
        if str(erg)[0:6]!="UPDATE":
          return -1
        else:
          return 0


      #
      # loescht ein Dokument aus dem Speicher
      #
      def delete_doc(self,ident):
        # erstmal als Bitstrom loeschen
        self.delete_bs(ident)
        # dann die Datenbank leeren
        conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
        res=self.conn.exec_query(
          "delete␣from␣doc_elements*␣where␣docnr="+
            str(ident)+";")
        res=self.conn.exec_query(
          "delete␣from␣doc_methods␣where␣docnr="+
            str(ident)+";")
```

```
    res=self.conn.exec_query(
      "
        update␣doc_description␣set␣sgml_stored='f'␣where␣docnr="+
      str(ident)+";")
    erg=res.commandstatus()
    if str(erg)[0:6]!="UPDATE":
      return -1
    else:
      return 0
  #
  # fuehrt ein SQL Statement auf der Datenbank aus
  #
  def sql_exec(self,querystr):
    conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)
    res=self.conn.exec_query(querystr)
     # Fehler aufgetreten ?
    if res.get_tuple_count()==0:
      # wenn ja, -1 zurueckgeben
      return -1
    else:
      #wenn nein Ergebnis zurueckgeben
      erg=res.get_resultlist(1,"|",1,0)
      return erg

    return erg
  #
  # Reinitialisiert die Datenbank
  # VORSICHT alle gespeicherten Daten gehen verloren
  #
  def dali_reinit(self):
    import time

    print "
      -----------------------------------------------------------"
    print "
      Sie␣sind␣dabei␣die␣komplette␣DALI␣Datenbank␣zu␣loeschen."
    print ""
    print "␣Ich␣warte␣jetzt␣mal␣20␣Sekunden,
      ␣wenn␣Sie␣es␣sich␣anders"
    print ""
    print "ueberlegt␣haben,
      ␣sollten␣sie␣ein␣paarmal␣CTRL-C␣druecken."
    print "
      ------------------------------------------------------------"
    time.sleep(20)
```

```
# Verbindung zu Postgres aufbauen
conn=pgpy.pgconn(DL_PGHOST,DL_PGPORT,DL_DBNAME)

# Alle Klassen entfernen
res=self.conn.exec_query("drop␣table␣doc_description;"
  )
res=self.conn.exec_query("drop␣table␣doc_elements;")
res=self.conn.exec_query("drop␣table␣doc_bin;")
res=self.conn.exec_query("drop␣table␣doc_tags;")
res=self.conn.exec_query("drop␣table␣doc_text;")
res=self.conn.exec_query("drop␣table␣doc_methods;")
res=self.conn.exec_query("drop␣table␣dtd;")
res=self.conn.exec_query("drop␣table␣local_doc_ids;")

# neue leere Klassen erzeugen
res=self.conn.exec_query("
  CREATE␣TABLE␣local_doc_ids␣(docnr␣int4,␣dtdnr␣int4)␣archive␣=␣none;")

res=self.conn.exec_query("
  CREATE␣TABLE␣doc_elements␣(docnr␣int4,␣elementnr␣int4,␣brother␣int4,
  ␣son␣int4,␣contents␣text)␣archive␣=␣none;")

res=self.conn.exec_query("CREATE␣TABLE␣dtd␣(dtdnr␣int4,
  ␣dtdid␣text,publicname␣text,
  ␣systemname␣text)␣archive␣=␣none;")

res=self.conn.exec_query("
  CREATE␣TABLE␣doc_tags␣()␣inherits␣(␣doc_elements)␣archive␣=␣none;")

res=self.conn.exec_query("
  CREATE␣TABLE␣doc_text␣()␣inherits␣(␣doc_elements)␣archive␣=␣none;")

res=self.conn.exec_query("
  CREATE␣TABLE␣doc_bin␣()␣inherits␣(␣doc_elements)␣archive␣=␣none;")

res=self.conn.exec_query("
  CREATE␣TABLE␣doc_description␣(docnr␣int4,␣dtdnr␣int4,␣title␣text,
  ␣author␣text,␣owner␣text,␣ident␣text,␣pubkey␣text,
  ␣info␣text,␣metadtdnr␣int4,␣bs_stored␣bool,
  ␣sgml_stored␣bool)␣archive␣=␣none;")

res=self.conn.exec_query("
  CREATE␣TABLE␣doc_methods␣(docnr␣int4,␣meth_class␣text,
  ␣meth_location␣text,␣meth_mimetype␣text)␣archive␣=␣none;")
```

```
    # Initialen Zaehlerstand einfuegen
    res=self.conn.exec_query("
      INSERT␣INTO␣local_doc_ids␣VALUES␣(100,1);")

    self.read_sgmlcat(self)
  #
  # traegt alle auf dem System verf"ugbaren␣DTD's␣in␣die
  # DTD Klasse der Datenbnak ein
  #
  def read_sgmlcat(self):
    import c_bsio
    import class_funcs
    import string

    fname=DL_SGMLCATPATH+DL_SGMLCATNAME
    fd=c_bsio.c_bsio(fname,"r")
    fd.open()
    buff="␣"
    while buff!="":
      buff=fd.readline()
      if buff[0:6]=="PUBLIC":
        newdtdid=class_funcs.get_newdtdid()
        publictup=string.split(buff,"\'")

        dtdid=string.strip(publictup[2])
        publicname=string.strip(publictup[1])
        systemname=DL_SGMLCATPATH+dtdid

        query="insert␣into␣dtd␣values␣("+str(newdtdid)+
          ","
        query=query+"'"+dtdid+"',"
        query=query+"'"+publicname+"',"
        query=query+"'"+systemname+"');"

        res=conn.exec_query(query)
        if res.commandstatus()[0:6]!="INSERT":
          print "
            Fehler␣bei␣INSERT␣in␣c_maintenance.read_sgmlcat"

    fd.close()
```

Klasse c_glimpindex

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : c_glimpindex.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Klasse für den Zugriff auf glimpseindex
# Anmerkungen      :
# -------------------------------------------------------------

class c_glimp_index:
      #
      # Konstruktor
      #
      def __init__(self):
        from sc_globals import *
        # setzen des Pfades in dem alle als
        # Bitstrom gespeicherten Dokumente liegen
        self.dir=DL_FULLTEXT_DIR
      #
      # Loescht die von glimpse erstellten Indizes
      #
      def remove_index(self):
        import os
        try:
          # loesche Index Files
          os.unlink(self.dir+".glimpse_*")
        except:
          # da waren wohl gar keine da
          pass
      #
      # Erstellt einen neuen Index fuer das
      # Bistromverzeichnis
      #
      def build_index(self):
        import os
        os.system("glimpseindex -f -H "+self.dir+" "+self.dir+
          " >/dev/null")
```

E.6.6 Paket Net (p_net.py)

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
#
# Dateiname        : p_net.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Paket Netz, fasst alle Netzpaket-Klassen
#                    zusammen
#
# Anmerkungen      :
#
# ---------------------------------------------------------------
import c_tcp
import c_stopwait1
import c_send
import c_retrieve
```

Klasse c_tcp

```
# ---------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1997
# Dateiname        : c_tcp.py
# Datum            : 03.11.1997
# letzte Änderung :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Eine einfache TCP Connection Klasse
#
# Anmerkungen      :
#
# ---------------------------------------------------------------
```

```
# imports
import socket
import time

class c_tcpconn:

      #
      # Konstruktor, wird kein Port mit uebergeben
      # dann wollen wir die Verbindung aufbauen(Client)
      #
      def __init__(self,host="",port=0):
        self.sock=socket.socket(socket.AF_INET,
          socket.SOCK_STREAM)

        if port !=0:
          try:
            self.sock.bind(host,port)
            self.typ="server"
          except socket.error,msg:
            print "---",msg,"---"

        else:
          self.typ="client"


      #
      # baut eine Verbindung mit host,port auf
      #
      def connect(self,host,port):
        self.sock.connect(host,port)

      #
      # wartet auf Verbindungsaufbau
      #
      def accept(self):
        self.sock.listen(5)
        self.peerconn,self.peeraddr = self.sock.accept()

      #
      # schreibt buff in den Socket
      #
      def send(self,buff):
        if self.typ=="client":
          self.sock.send(buff)
```

```
    else:
      self.peerconn.send(buff)

  #
  # wartet auf eine Nachricht die max. buffsize bytes lang
  # ist
  #
  def recv(self,buffsize):
    if self.typ=="client":
      return self.sock.recv(buffsize)
    else:
      return self.peerconn.recv(buffsize)

  #
  # wie recv, wartet aber nur secs Sekunden
  #
  def poll(self,buffsize,secs=1):
    polltime=secs
    erg=""
    if self.typ=="client":
      self.sock.setblocking(0)
    else:
      self.peerconn.setblocking(0)

    while erg==""and polltime>0:
      try:
        if self.typ=="client":
          erg=self.sock.recv(buffsize)
        else:
          erg=self.peerconn.recv(buffsize)
      except socket.error:
        erg=""
        time.sleep(1)
        polltime=polltime-1

    if self.typ=="client":
      self.sock.setblocking(1)
    else:
      self.peerconn.setblocking(1)

    return erg

  #
  # baut eine evtl. bestehende Verbindung ab
  # und schliesst die Sockets
```

```
#
def close(self):
  if self.typ=="server":
    self.sock.close()
    self.peerconn.shutdown(1)
  else:
    self.sock.shutdown(1)

  return


#
# liefert die lokale Adresse zurueck
#
def getaddr(self):
  return [socket.gethostname(),self.sock.getsockname()]

#
#liefert die Remote-Adresse zurueck
#
def getpeeraddr(self):
  if self.typ=="client":
    return self.sock.getpeername()
  else:
    return self.peerconn.getpeername()
```

Klasse c_stopwait1

```
# -------------------------------------------------------------
# Projekt         : Digitale Bibliotheken Projekt
#                    Uni-Frankfurt/M, Professur Telematik und
#                    verteilte Systeme, Prof. O. Drobnik
#                    Diplomarbeit, Matzen,Hans, 1998
# Dateiname       : c_stopwait1.py
# Datum           : 09.12.1997
# letzte Änderung :
# Autor           : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache         : Python v1.4
# Beschreibung    : Hier ist eine einfache Stop-Wait Protokoll
#                   Maschine als Klasse implementiert.
# Anmerkungen     : Diese Klasse kann verwendet werden, um
#                   aus Dokumentemethoden separate Netzverbindung
#                   zur Datenuebermittlung aufzubauen. Sie ist
#                   von der Klasse c_tcpconn abgeleitet.
# -------------------------------------------------------------
#

# import
from c_tcp import c_tcpconn
import c_bsio
import select

class c_stopwait1(c_tcpconn):
      #
      # Konstruktor, initialsiert Variable
      # host,port gibt die Adresse des lokalen sockets an
      # buffsize die Paketgroesse bei der Uebertragung
      # timeout die Zeit (in Sekunden) die auf eine Bestaetigung
      # gewartet wird, mittels debug=1 werden zusaetzliche
      # Nachrichten  ueber den Verlauf der uebertragung
      # ausgegeben
      #
      def __init__(self,host="localhost",port=0,buffsize=512,
        timeout=5,debug=0):
        # Konstruktor der Basisklasse aufrufen
        c_tcpconn.__init__(self,host,port)
        self.blocksize=buffsize
        self.timeout=timeout
        self.debug=debug
      #
      # Destruktor, trennt die Verbidung falls sie aufgebaut ist
```

```
#
def __del__(self):
  try:
    c_tcpconn.sock.close()
    c_tcpconn.peerconn.close()
  except:
    pass
#
# fuehrt einen TCP connect an  die durch
# host,port gegebene Adresse aus
#
def connect(self,host,port):
  c_tcpconn.connect(self,host,port)
#
# wartet auf Verbidungsaufnahme durch eine Gegenstelle
#
def accept(self):
  c_tcpconn.accept(self)

#
# sendet sendstr mittels des Stop-Wait Protokolls
# dabei wird die Nachricht segmentiert (blocksize)
# und die Uebertragung eines Pakets bei
# Uebertragunsfehlern
# bis zu 10 mal wiederholt
#
def send_str(self,sendstr):
  hilf=sendstr
  errcount=0
  muell=self.poll(self.blocksize+1,1)

  while len(hilf)>0 and errcount < 10:
    self.send(hilf[:self.blocksize])

    if self.debug:
      print "send:",hilf[:self.blocksize],
         len(hilf[:self.blocksize])

    ack=self.poll(8,self.timeout)

    if self.debug:
      print "recv:",ack

    if ack=="ACK"+str(len(hilf[:self.blocksize])):
      hilf=hilf[self.blocksize:]
```

```
        else:
          errcount=errcount+1

      # Konnten wir alles uebertragen
      if errcount<10:
        ackend=0
        while ackend==0 and errcount <10:
          self.send("END")
          ack=self.poll(8,self.timeout)

          if self.debug:
            print "recv:",ack

          if ack=="ACK3":
            ackend=1
          else:
            errcount=errcount+1
      # Rueckgabewert ermitteln
      if errcount < 10:
        return 0
      else:
        return -1

    #
    # wie send_str, nur wird die Datei fname gesendet
    #
    def send_file(self,fname):
      fd=c_bsio.c_bsio(fname,"r")
      errcount=0
      buff="␣"
      fd.open()
      while buff!=""and errcount < 10:
        buff=fd.read(self.blocksize)
        self.send(self,buff)
        ack=self.poll(8,self.timeout)
        if self.debug:
          print "recv:",ack
        if ack!="ACK"+str(len(buff)):
          fd.seek(-len(buff),0)
             errcount=errcount+1
      # Konnten wir alles uebertragen
      if errcount<10:
        ackend=0
        while ackend==0 and errcount <10:
          self.send(self,"END")
```

```
        ack=self.poll(8,self.timeout)
        if self.debug:
          print "recv:",ack
        if ack=="ACK3":
          ackend=1
        else:
          errcount=errcount+1
    fd.close()
    # Rueckgabewert ermitteln
    if errcount < 10:
      return 0
    else:
      return -1


  #
  # empf"angt␣eine␣Zeichenkette␣von␣␣der␣Gegenstelle
  #
  def recv_str(self):
    erg=""
    buff=""
    if self.typ=="server":
      select.select([self.peerconn.fileno()],[],[])
    else:
      select.select([self.sock.fileno()],[],[])

    while buff !="END":
      buff=self.poll(self.blocksize,self.timeout)

      if self.debug:
        print "recv:",buff

      if buff !="":
        if buff !="END":
          erg=erg+buff
        self.send("ACK"+str(len(buff)))

        if self.debug:
          print "send:ACK"+str(len(buff))

      else:
        self.send("ERR")

        if self.debug:
          print "send:ERR"
```

```
  return erg
#
# schreibt alle empfangenen Daten in die Datei mit
# dem Namen fname
#
def recv_file(self,fname):
  fd=c_bsio.c_bsio(fname,"a+")
  fd.open()
  buff=""
  while buff !="END":
    buff=self.poll(self.blocksize,self.timeout)
    if self.debug:
      print "recv:",buff
    if buff !="":
      if buff !="END":
        fd.write(buff)
      self.send("ACK"+str(len(buff)))
      if self.debug:
        print "send:ACK",str(len(buff))
    else:
      self.send("ERR")
      if self.debug:
        print "send:ERR"
    fd.close()
    return 0
```

Klasse c_send

```
# -------------------------------------------------------------
# Projekt          : Digitale Bibliotheken Projekt
#                     Uni-Frankfurt/M, Professur Telematik und
#                     verteilte Systeme, Prof. O. Drobnik
#                     Diplomarbeit, Matzen,Hans, 1998
# Dateiname        : c_send.py
# Datum            : 09.12.1997
# letzte Änderung  :
# Autor            : Hans Matzen, 1997, Frankfurt/M, Deutschland
# Sprache          : Python v1.4
# Beschreibung     : Diese Klasse versendet Teile eines bestimmten
#                    Dokuments ueber ein separate Netzverbindung
# Anmerkungen      : Diese Klasse wird von den Klassen c_sgmlquery
#                    und c_stopwait1 abgeleitet.
#
# -------------------------------------------------------------
#
class c_send(c_sgmlquery,c_stopwait1):
      #
      # Konstruktor
      # ident gibt die Dokument-ID an
      # peerhost, peeport geben die Netzadresse des Peers an
      # an diese Adresse werden alle Daten gesendet
      # blocksize gibt die Paketgroese an (s. c_stopwait1)
      # timeout gibt den timeout fuer das Warten auf Paket-
      # bestaetigungen an.
      #
      def __init__(self,ident,peerhost,peerport,blocksize,
        timeout):
        # Konstruktoren der Basisklassen aufrufen
        c_sgmlquery.__init__(self,ident)
        c_stopwait1.__init__("",0,blocksize,timeout)
        self.peerport=peerport
        self.peerhost=peerhost
        self.connect(self.peerhost,self.peerport)
        self.ident=ident
      #
      # Destruktor
      #
      def __del__(self):
        # Destruktor der stopwait Klasse aufrufen
        c_stopwait1.__del__()
      #
```

```
# sendet als Bitstrom gespeicherte Daten
# und zwar von der Position start bis Position end
# wobei start und end absolute Positionen innerhalb
# des Bitstroms bezeichnen
#
def send_bs(self,start,end):
  import c_bsio
  from sc_globals import *
  fname=DL_BSOUTPATH+str(ident)+'.bs'
  fd=c_bsio.c_bsio(fname,'r')
  fd.seek(start)
  while fd.tell()<=end:
    buff=fd.readline()
    self.send_str(buff)

  fd.close()
  del fd
#
# sendet den Inhalt des tagcounten Tags mit Namen
# tagname an den Peer
#
def send_tagcontents(self,tagname,tagcount):
  self.send_str(self.sgmlquery_tagcontens(tagname,
     tagcount))
  return 0
#
# sendet den das tagcounte Tag mit Namen
# tagname an den Peer
#
def send_tag(self,tagname,tagcount):
  self.send_str(self.sgmlquery_tag(tagname,tagcount))
  return 0
#
# sendet die Tag-Struktur (Strukturbaum) des Dokuments
# an den Peer
#
def send_structure(self):
  self.send_str( self.sgmlquery_structure())
  return 0
#
# sendet das gesamte Dokument an den Peer
#
def send_doc(self):
  self.send_str(self.sgmlquery_doc())
  return 0
```