Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

# Erweiterung eines Dokumentenservers um multimediale und zusammengesetzte Dokumente

Diplomarbeit

Leipzig, Juli 2003

vorgelegt von

Thomas Weise
geb. am: 03.06.1977 in Leipzig

Studiengang Informatik

## *Kurzfassung*

An Universitäten und anderen öffentlichen Einrichtungen gibt es zunehmend wissenschaftliche Arbeiten, die Multimediadaten enthalten. Ein rein textbasierter Dokumentenserver reicht diesen Anforderungen nicht mehr. Ziel ist es, ein Konzept zu entwickeln, einen solchen bereits existierenden Dokumentenserver für multimediale und zusammengesetzte Dokumente zu erweitern.
Die Nutzung des bisherigen Datenbestandes soll sichergestellt werden.

# Inhaltsverzeichnis

# *Abbildungs- und Tabellenverzeichnis*

# 1 Einleitung

## *Motivation*

Diese Arbeit basiert auf einem Datenbank- und Web-basierten Dokumentenserver, welcher an der Universität Leipzig entwickelt wurde. Er dient seit 1998 dazu, einen zentralen Zugangspunkt für Forschungsergebnisse, also Dissertationen, Habilitationsschriften, Diplom- und Magisterarbeiten, aber auch Reports, Preprints, Zeitschriften- und Tagungsbeiträge sowie Monographien anzubieten. Dabei werden dezentrale Verwaltung und verschiedene Arbeitsabläufe unterstützt, um organisatorische und rechtliche Erfordernisse für bestimmte Dokumentarten zu erfüllen. Alle Dokumente sind in mehreren Formaten abrufbar und können auch online seitenweise betrachtet werden. Der Dokumentenbestand kann mittels Volltextsuche und anhand von bibliographischen Daten durchsucht werden. Über eine mehrstufige Navigationsschnittstelle können die Dokumente auf mehreren Ebenen eingeteilt werden. Der Dokumentenserver ist interoperabel mit globalen digitalen Bibliotheken wie NCSTRL und kann an die Bedürfnisse verschiedener Organisationen angepasst werden.

In Zeiten, in denen überall Informationen multimedial aufgearbeitet werden, ist es notwendig, diese Werkzeuge auch in einen modernen Dokumentenserver einzubauen. Zunehmend enthalten wissenschaftliche Arbeiten multimediale Daten, seien es Bilder, Audio- oder Videodaten. Ein rein textbasierter Dokumentenserver erfüllt diese Anforderungen nicht mehr. Es muss ein Konzept entwickelt werden, das auf die Bedürfnissen an eine solche Erweiterung eingeht. Ziel ist es ebenfalls, mehr auf die Struktur von Dokumenten einzugehen, d.h. einzelne Kapitel zur Verfügung zu stellen oder z.B. mehrere Berichte einer Tagung als ein Gesamtdokument anzusehen. Dabei wird betrachtet, wie die Erweiterungen in das bestehende Konzept eingearbeitet werden können. Die weitere Nutzung des Datenbestandes eines existierenden Systems soll selbstverständlich sichergestellt werden.

## *Gliederung der Arbeit*

Im folgenden Kapitel werden zunächst die Datenformate für die multimedialen Erweiterungen ausgearbeitet. Dabei wird erläutert, welche Anforderungen an die Multimediaformate gestellt werden.
Kapitel 3 beschäftigt sich mit Überlegungen, wie zusammengesetzte Dokumente in einem Dokumentenserver archiviert werden können.
Im Kapitel 4 wird der Aufbau des bestehenden Dokumentenservers kurz erläutert und die allgemeine Funktionsweise beschrieben.
Kapitel 5 beschäftigt sich mit den Änderungen, die mit dem Einbinden der zusätzlichen Formate in das bestehende Konzept einher gehen. Diese Änderungen werden dann im Detail im Kapitel 6 betrachtet.
Mit dem Hauptteil der Arbeit, der Implementation der Erweiterungen, befasst sich Kapitel 7.
Schließlich wird im Kapitel 8 über den Test und Beispielanwendungen des erweiterten Dokumentenservers berichtet.
Eine Zusammenfassung in Kapitel 9 hebt noch einmal die Kernpunkte der Arbeit hervor und gibt einen Ausblick auf mögliche Verbesserungen und Erweiterungen.

## *Bisherige Dokumentationen*

Als Grundlage für die Erweiterungen war es notwendig, die bereits zur Verfügung stehende Dokumentation zu nutzen. Einen grundlegenden Überblick geben dabei die Berichte [1] und [2]. Natürlich standen die Quelltexte zum bestehenden und laufenden System zur Verfügung, welche die Grundlage für die Erweiterungen bildeten.

## *Danksagung*

Ich möchte mich bei meinen Betreuern, Herrn Prof. Dr. E. Rahm und Herrn Dr. D. Sosna, sowie Herrn S. Melnik für die interessante Aufgabenstellung und die ausgezeichnete Betreuung bedanken. Mein Dank gilt auch den anderen Mitarbeitern der Abteilung Datenbanken für die freundliche Unterstützung bei technischen Problemen.

Nicht zuletzt möchte ich meinen Eltern und allen meinen Freunden für ihre Unterstützung und ihr Verständnis dafür danken, dass ich in den letzten Monaten nur wenig Zeit für sie hatte. Vor allem aber danke ich meiner Freundin Kristin für ihre Unterstützung und ihre Geduld.

# 2 Formate für die multimedialen Erweiterungen

## *2.1 Vorüberlegungen*

Bisher wurden auf dem Dokumentenserver die eingehenden Dokumente im *Postscript*-Format (PS) bzw. im *Portable Document Format* (PDF) eingereicht und zur Verfügung gestellt. Eine automatische Konvertierung sicherte dabei, dass jedes Dokument in beiden Formaten bereitliegt. Für die Erzeugung von ASCII-Dateien aus Postscript wird das kostenlose Tool *Pscript* von Günther Radestock [3], Universität Karlsruhe, genutzt. Zur Erzeugung der PDF-Dateien dient der *Acrobat Distiller* [4].
An einer Universität entstehen aber nicht nur Textdokumente, sondern auch Multimediadaten in Form von Bilddokumenten (z.B. aus Medizin, Kunstwissenschaften, Astronomie), Audiodaten (z.B. Musikwissenschaften) und Videodaten (z.B. Kommunikations- und Medienwissenschaften, allgemeine Präsentationen). Als Videodaten sind hierbei auch Animationen und Simulationen gemeint. Durch die rasante Entwicklung im Internet bezüglich Übertragungsgeschwindigkeit und Verbreitung ist es möglich und sogar notwendig, solche multimedialen Inhalte einem breiteren Publikum zur Verfügung zu stellen. Mithilfe eines solchen Dokumentenservers gelingt die zentrale Sammlung dieser Dokumente zumindest auf Universitätsebene.
Da aber auch andere Institutionen derartige Online-Bibliotheken anbieten, stellt sich die Frage, welche Formate man im Einzelnen auswählt, um die multimedialen Inhalte zu präsentieren. Dabei muss man betrachten, welche Formate bei großen Online-Bibliotheken und Suchmaschinen weit verbreitet sind, um eine ausreichende Kompatibilität zu gewährleisten.
Ein Teil der Arbeit widmet sich den Eigenschaften der wichtigsten und am weitesten verbreiteten Formate sowie deren Nutzen für den Dokumentenserver. Diese werden hinsichtlich Speicherplatzbedarf und Qualität der späteren Präsentation betrachtet. Da die Erweiterungen ohne zusätzliche Kosten erfolgen sollen, wird die Verwendung der Formate auch diesbezüglich untersucht.
Bei einigen Formaten wird das Thema Digital Rights Management (DRM) angesprochen. DRM zielte ursprünglich auf die Sicherheit und Verschlüsselung von digitalen Daten zur Freigabe des Inhaltes gegen Bezahlung und zum Schutz vor unerlaubtem Kopieren. Heute ist es ein System zur Beschreibung, Identifikation, Überwachung, Protokollierung und zum Schutz aller Formen des Gebrauches von Rechten für materielle und immaterielle Werte. Es ist wichtig zu beachten, dass DRM die „digitale Verwaltung von Rechten“ ist und nicht die „Verwaltung von digitalen Rechten“ [25]. DRM-geschützte Musikdateien können beispielsweise so freigegeben werden, dass sie nur auf bestimmten PCs oder nur für einen bestimmten Zeitraum abgespielt werden können.

## 2.2 Formate für Bild-, Audio- und Videodaten

Für die Speicherung sollte man zwei Zielstellungen unterscheiden: zum einen den schnellen Zugriff, bei dem es weniger auf die Qualität der dargestellten Informationen ankommt und zum anderen die Präsentation ohne Qualitätsverlust, unabhängig vom Speicherplatzbedarf. Optional in es vom Vorteil, wenn die Formate selbst - zur Wahrung von Urheberrechten - Informationen über den Autor enthalten können.
Für einen Online-Dienst, wie einen Dokumentenserver, ist es von Vorteil, dass die abrufbaren Bilddaten ohne zusätzliche Software betrachtet werden können. Das heißt, dass man auch die von den Internet-Browsern unterstützten Formate einbeziehen sollte. Standardmäßig sind zum Beispiel folgende Bildformate im Web-Browser lesbar:

| | |
|---|---|
| Netscape: | gif, jpg, png |
| Mozilla: | gif, jpg, png, bmp |
| Internet Explorer: | gif, jpg, png, bmp |

Der Browser ermittelt anhand des HTTP-Headers des empfangenen Dokuments den Typ der nachfolgenden Daten und wählt anhand einer internen Zuordnungstabelle das geeignete Programm zur Präsentation. Der Dokumententyp wird im MIME-Format (Multipurpose Internet Mail Extensions) im HTTP-Header kodiert. Beispielsweise erkennt der Browser ein PNG-Bild an folgendem Eintrag:

content-type: image/png

Diese Erweiterungen sind weltweit standardisiert.

Ein weiterer Vorteil bei der Benutzung von jeweils zwei verschiedenen Formaten ist die Erhöhung der Wahrscheinlichkeit, dass der Benutzer ein geeignetes Programm zur Darstellung hat.
Weiterhin sollte man die von anderen Online-Diensten verwendeten Multimediadaten und deren Formate berücksichtigen. Diese Datenbestände werden sowohl bei Suchmaschinen als auch bei Online-Bibliotheken noch ganz unterschiedlich unterstützt. Die Suchmaschine „Google“ [5] etwa beschränkt sich im Speziellen nur auf Bilder. So kann separat nach Bildern im JPEG- oder GIF-Format gesucht werden. Nicht nur nach Bildern, sondern auch nach Audio- und Videodateien kann bei der Suchmaschine „Altavista“ [6] gesucht werden. Dabei werden jedoch nur Bilder mit dem JPEG-Format berücksichtigt. Audiodaten müssen im WAV- oder MP3-Format vorliegen. Bei den Videodaten werden eine Vielzahl von Formaten unterstützt: Windows Media Dateien (AVI, ASX, ASF), Real Video (RM, RAM) und QuickTime-Movies (MOV).
Der Datenbestand von Online-Bibliotheken ist ebenfalls nicht einheitlich. Nicht alle dieser Institutionen archivieren jeweils Bild-, Audio- und Videodaten.
Das MILESS-Projekt der Universität Essen [7] bietet Bilddaten unter anderem in den Formaten TIFF, BMP und JPEG. Audiodaten gibt es im MP2- und MP3-Format, Videodaten als MPEG-1, MPEG-2 und Real Video. Die Digitale Bibliothek Thüringen [8], die auf dem UrMEL-Projekt der Universität Jena basiert, unterstützt Bilder im GIF- bzw. JPEG-Format und Videodaten im MPEG-1, MPEG-2 und Real Video-Format. Das

„Digitale Video- und Audioarchiv“ (DIVA) der Universitätsbibliothek Karlsruhe [9], Grundlage unter anderem für das Bibliotheksservice-Zentrum Baden-Württemberg bzw. den Südwestdeutschen Bibliotheksverbund, präsentiert Audiodaten im Real Audio-Format, Videodaten im Real Video-, Windows Media- oder AVI-Format. Der Mediaserver der Universität Heidelberg [10] bietet nur Videodaten (Real Video, Windows Media), der Archivserver DEPOSIT.DDB.DE der Deutschen Bibliothek [11] in Frankfurt/Main sogar nur Textdokumente.

## *2.3 Bilder*

Im Folgenden soll ein Überblick über die verschieden Formate für Bilddaten gegeben werden. Im Anschluss folgt eine Auswertung. Formate, welche nicht betrachtet werden, sind im Allgemeinen noch weniger verbreitet, oder genügen den Anforderungen nur unzureichend. Hierzu zählen zum Beispiel Vektorgrafiken.

### 2.3.1 BMP

Das BMP - Format („Bitmap“) ist hauptsächlich unter Windows verbreitet. Dabei werden die Farbwerte für Rot, Grün und Blau aus einem bestimmten Wertebereich (Farbtiefe) für jedes Pixel angegeben. Dadurch werden die Bildinformationen ohne Qualitätsverlust abgespeichert, was jedoch einen hohen Speicherplatzbedarf zur Folge hat. Der MIME-Typ zur Übertragung im Internet ist „image/bmp“.

### 2.3.2 GIF

Das "Graphics Interchange Format“ (GIF) gewann an Bedeutung, als das Internet sich rasant entwickelte. Durch die Komprimierung der Bilddaten und die Möglichkeit von „progressive download“ ist es ideal für das Internet, da Teile des Bild schon während des Ladens angezeigt werden können. Der Nachteil besteht darin, dass die Benutzung der Lempel-Ziv-Welch – Kompression für GIF-Dateien von der Firma *Compuserve* lizenziert ist. Das Format ist ideal für Bilder mit bis zu 256 Farben/Graustufen und wenigen Details. Außerdem ist es möglich, in einer GIF-Datei mehrere Bilder und Kommentare abzuspeichern. Der MIME-Typ ist „image/gif“.

### 2.3.3 JPEG

Das JPEG-Format benutzt eine Kompression, die von der "Joint Photographics Expert Group" entwickelt wurde. Sie erlaubt die Komprimierung von Bilddaten mit beliebiger Anzahl von Farben oder Graustufen in beliebigen Komprimierungsgraden. Besonders bei Bildern mit mehr als 8 Bit Farbtiefe macht sich der geringere Platzbedarf bemerkbar. Der MIME-Typ ist „image/jpeg“.

### 2.3.4 PNG

Das PNG-Format ("Portable Network Graphics") wurde als lizenzfreier Ersatz für das Graphics Interchange Format entwickelt. Es bietet eine verlustfreie Komprimierung von Bilddaten im Rasterformat mit 1 bis 16 Bit Farbtiefe/Graustufen, die Möglichkeit von „progressive download“, Überprüfung der Dateiintegrität und einfache Erkennung von allgemeinen Übertragungsfehlern sowie teilweise effektivere Komprimierung als beim GIF-Format. Der MIME-Typ ist „image/png“. Weiterhin ist die Einbindung von (un)komprimierten Kommentaren wie Copyright und Bildinformationen möglich.

### 2.3.5 TIFF

Das "Tagged-Image File Format" wurde von der Aldus Corporation eingeführt und nach deren Übernahme durch Adobe weiterentwickelt. Es ist ein frei verwendbares Raster-Dateiformat für Bilddaten, die hauptsächlich von Scannern, Capture-Tools und aus Bildbearbeitungssoftware stammen. Es werden verschiedene Farbtiefen (1-24 Bit) und Komprimierungsalgorithmen unterstützt, z.B. JPEG und LZW. Eine TIFF-Datei kann mehrere Bilder enthalten. Der MIME-Typ ist „image/tiff“.

### 2.3.6 Fazit

Wegen der Qualitätsvorteile bietet sich zunächst das PNG-Format zur Nutzung an. Es liefert einen hohen Komprimierungsgrad ohne Verlust von Bildinformationen. Das BMP-Format sowie das TIFF-Format sind aufgrund der großen Datenmenge bei der Archivierung im wissenschaftlichen Umfeld weniger verbreitet. Für den schnellen Zugriff ist das JPEG-Format gut geeignet. Es ist sehr platzsparend bei frei wählbarer Qualität sowie aufgrund dieser Eigenschaften weit verbreitet. Das GIF-Format eignet sich wegen der Lizenzproblematik weniger zur Nutzung.
Die gewählten Formate sind auf allen Plattformen verfügbar (DOS, Windows, AIX, BeOS, FreeBSD, HP-UX, Linux, MacOS, OS/2, QNX, SGI). Meist sind die Dateibetrachter schon im den jeweiligen Web-Browsern integriert. In Tabelle 2.1 ist eine Auswahl von weiteren Dateibetrachtern und -konvertern für die jeweiligen Betriebssysteme aufgelistet.
Aufgrund der Anforderungen an den Dokumentenserver sollte man beachten, dass das Konvertieren zu Informationsverlust führen kann, jedoch nicht unbedingt auch Speicherplatzeinsparungen zur Folge hat. Deswegen sollten die Bilddateien von den Autoren mit möglichst hoher Qualität geliefert werden. Die beschriebenen Formate können so auch bequem zur Erstellung von Photo-CDs verwendet werden, was jedoch nicht Bestandteil dieser Arbeit sein soll.

| Anwendung | Bezugsquelle, unterstützte Betriebssysteme |
|---|---|
| ImageMagick-2.0 | |
| | http://www.imagemagick.org |
| | Windows, Linux, UNIX, MacOS, OS/2, VMS – Freeware |
| Tiffy | |
| | http://www.tiffy.de |
| | DOS, Java – Shareware |
| Irfan View | |
| | http://www.irfanview.com |
| | Windows - Freeware |
| ACDSee | |
| | http://www.acdsystems.com |
| | Windows - Shareware |
| Gimp | |
| | http://www.gimp.org |
| | Linux - Freeware |
| XnView | |
| | http://www.xnview.com |
| | Linux - Freeware |
| GraphicConverter | |
| | http://www.lemkesoft.de/de_index.html |
| | MacOS - Shareware |
| Image Viewer | |
| | http://www.sun.com |
| | Solaris, gehört zum Betriebssystem |
| Image Viewer | |
| | http://www.be.com |
| | BeOS, gehört zum Betriebssystem |
| XPaint | |
| | https://sourceforge.net/projects/sf-xpaint/ |
| | Linux, AIX, BSD, HP-UX, SunOS/Solaris, OS/2 – Freeware |
| xv | |
| | http://www.trilon.com/xv/index.html |
| | alle UNIX/Linux-Derivate mit X11R4 – Shareware |
| xloadimage | |
| | http://rpmseek.com/rpm-pl/xloadimage.html?hl=de&cx=645:X:0 |
| | alle UNIX/Linux-Derivate mit X11 - Freeware |
| xnview | |
| | http://perso.wanadoo.fr/pierre.g/xnview/enhome.html |
| | Windows, Linux, FreeBSD, Solaris, Irix, HP-UX, AIX, BeOS, OS/2 - Freeware |

**Tabelle 2.1: Bildbetrachter und -konverter**

## *2.4 Audio*

Im folgenden werden die verschieden Formate für Audiodaten betrachtet. Eine Auswertung erfolgt im Anschluss. Auch hier gilt, dass Formate, welche nicht betrachtet werden, im Allgemeinen noch weniger verbreitet sind oder den Anforderungen nur unzureichend genügen. Hierzu zählen zum Beispiel Audioformate, die nicht für hoch qualitative Aufnahmen geeignet sind.

### 2.4.1 AAC

AAC (MPEG-2 „Advanced Audio Coding“) ist der neue Standard für professionelles Audio als Weiterentwicklung von MPEG Layer-3. Es liefert eine effizientere Kompression als andere Formate (zum Beispiel MP3) bei CD-Qualität. Dadurch haben die Daten bei gleicher Qualität eine geringere Größe. Das Format wurde von der MPEG-Gruppe (Dolby, Fraunhofer Gesellschaft, AT&T, Sony, Nokia) entwickelt. Es basiert auf neuen Technologien der Signalverarbeitung. Merkmale sind dabei: Samplefrequenz zwischen 8 und 96 kHz, Unterstützung von bis zu 48 Kanälen, variable Bitrate-Kodierung (VBR), Abtastrate bis 96 kHz, geringe Prozessorleistung für die Dekodierung und die Kompatibilität zum Digital Rights Management (DRM).
Ein Ziel bei der Entwicklung von AAC war die Suche nach einer effizienten Methode zum Kodieren von Surround Sound Signalen. Als relativ neue Entwicklung ist dieses Format noch nicht weit verbreitet. Der MIME-Typ ist unter anderem „audio/mp[e]g“.

### 2.4.2 AIFF (Apple)

Das AIFF-Format („Audio Interchange File Format“) ähnelt dem WAV-Format (siehe unten). Es speichert digitale Audiosignale in Wellenform und wird seit 1988 besonders auf Apple Computern verwendet. Dabei werden mehrere Audiokanäle mit unterschiedlichen Abtasttiefen und –raten unterstützt. Optional können Informationen des Autors abgespeichert werden.
Die Verbreitung des Formats beschränkt sich hauptsächlich auf MacOS und Windows. Der MIME-Typ ist „audio/[x-]aiff“.

### 2.4.3 AU

Das AU-Format wurde meist nur unter UNIX (Sun und NeXT) benutzt, hat aber keine große Bedeutung mehr. Es unterstützt eine beliebige Anzahl von Abtasttiefen (8 bis 64 Bit), Sampleraten und Audiokanälen. Der MIME-Typ ist „audio/basic“.

### 2.4.4 MIDI

Beim MIDI-Format werden die Audiodaten selbst nicht gespeichert, sondern nur die Information zu deren Erzeugung. Dadurch ist die Verwendung hardwareabhängig und sehr beschränkt in der Anwendung. Der MIME-Typ ist „audio/[x-]mid[i]“.

### 2.4.5 MP3

Das MP3-Format (MPEG Audio Layer-3) dient ebenfalls zum Speichern von digitalen Audiodaten in Wave-Form, im Unterschied zu WAV aber verlustbehaftet. Es wurde von der Fraunhofer Gesellschaft entwickelt. Im Gegensatz zum WAV-Format werden die Frequenzen ignoriert, die der Mensch nicht hören kann. Die daraus resultierenden Daten werden außerdem komprimiert. Dadurch kommt man auch ohne hörbaren Qualitätsverlust mit viel weniger Platz aus. Der Platzbedarf steigt proportional zur Qualität. Sie richtet sich nach der Bitrate der Audiodaten, die zwischen 8 und 448 Kilobit pro Sekunde liegen kann. Schon bei einer Bitrate von 128 kbps sind kaum noch Unterschiede zum WAV-Format zu hören, wobei der Platzbedarf bereits auf ein Zehntel sinkt. Aufgrund dieser Eigenschaften ist das MP3-Format sehr beliebt und weit verbreitet. Optional können Informationen über eine Datei mit Hilfe des sogenannten ID3-Tags gespeichert werden (Titel, Interpret, Jahr, Genre u.a.).
Eine Weiterentwicklung ist das abwärtskompatible MP3PRO-Format, welches durch die erweiterte Analyse der Audiodaten (SBR - Spectral Band Replication) bei gleicher Bandbreite eine wesentlich bessere Qualität liefert. Der MIME-Typ ist „audio/[x-]mp3“.

### 2.4.6 OGG Vorbis

Das OGG-Format ist ein neues Audio-Komprimierungsformat. Es ist vergleichbar mit anderen Formaten zum Speichern und Abspielen von verlustbehafteten digitalen Audiodaten wie MP3 und VQF (siehe unten), mit dem Unterschied, dass es lizenzfrei ist. Vorbis ist der Name des speziellen Audio-Kompressionsschemas zum Erstellen von OGG Vorbis-Dateien. Die Bitraten liegen beim OGG-Format zwischen 32 und 500 kbps bei 44,1 kHz Abtastrate. Informationen über den Inhalt können auch bei diesem Format gespeichert werden. „Progressive download“ („Streaming“) ist, genauso wie das Einbinden von anderen multimedialen Daten (Bilder, Video, MIDI), ein wichtiger Punkt bei der Entwicklung gewesen. Diese Eigenschaften befinden sich aber noch in einem Entwicklungsstadium.
Trotz der lizenzfreien Verwendung ist dieses Format weniger weit verbreitet als MP3. Der MIME-Typ ist „application/x-ogg“.

### 2.4.7 Real Audio

Das „Real Audio“-Format wurde von der Firma RealNetworks in Zusammenarbeit mit Sony entwickelt. Das Ziel war dabei, Audiodaten mit möglichst hoher Qualität über das Internet zu verbreiten (Streaming). Hierzu können die ursprünglichen digitalen Audiodaten (auch Mehrkanal-Surround-Signale) mit Hilfe verschiedener RealAudio Codecs komprimiert werden. Die Daten werden dabei mit Bitraten von 16 Kbps bis 400 Kbps kodiert. Das Abspielen von Real Audio (Dateityp meist *.ra*, *.rm, .rmm* oder *.ram*) erfolgt mit Hilfe von *RealJukebox* oder mit dem *RealPlayer*, den es für Windows, Linux, UNIX und QNX gibt.
Durch das Streaming-Verfahren können Audiodaten ideal im Internet präsentiert werden. Der MIME-Typ ist „audio/vnd.rn-realaudio“ oder „audio/x-pn-realaudio“. Um Real Audio-Dateien ohne Einschränkungen zu erstellen, ist aber eine kommerzielle Software von RealNetworks nötig. Aufgrund dieses Aspekts ist das Format für den Dokumentenserver nicht geeignet.

### 2.4.8 VQF

Das „Vector Quantization Format“ ist ein verlustbehaftetes Audio-Kompressions-Format, das von NTT Labs entwickelt wurde. Es unterstützt Bitraten von 8 bis 96 Kbit pro Sekunde und eignet sich besonders für hohe Qualität bei niedrigen Bitraten. Es komprimiert etwa 30 Prozent besser als MP3. Eine 96 Kbit/s VQF-Datei ist qualitativ vergleichbar mit einer 128 Kbit/s MP3-Datei. Dabei wird eine ähnliche Technologie wie die SBR bei MP3PRO verwendet. Aufgrund fehlender Weiterentwicklung ist das VQF-Format kaum verbreitet und durch das OGG Vorbis-Format bereits technisch überholt. Der MIME-Typ ist „audio/x-twinvq“.

### 2.4.9 WAV

Das WAV-Format ist ein Dateiformat zum unkomprimierten Speichern von digitalen Audiodaten in Wellenform. Es unterstützt eine beliebige Anzahl von Abtasttiefen (8 bis 32 Bit), Sampleraten (8 bis 48 kHz) und Audiokanälen. Im Rahmen dieser Eigenschaften werden die Audiodaten ohne Qualitätsverlust abgespeichert und dadurch aber auch sehr umfangreich. Das WAV-Format hat eine große Bedeutung auf PC Plattformen, ist aber auch unter UNIX weit verbreitet. Der MIME-Typ zur Übertragung im Internet ist „audio/[x-]wav“.

### 2.4.10 WMA

Das WMA-Format „Windows Media Audio“ ist ein von Microsoft entwickeltes Format für Abspielen, Streaming und Download von Audiodaten. Dabei können die Daten unkomprimiert oder mit Hilfe des Windows Media Audio-Codec komprimiert werden. Optional können Informationen über den Inhalt gespeichert werden. Außerdem überwacht das bei WMA verwendete Digital Rights Management (DRM) die Vervielfältigung einer Datei. Der MIME-Typ ist „audio/x-ms-wma“.

### 2.4.11 Fazit

Bezüglich der Qualität der Audiodaten sollte man das WAV-Format verwenden, alle anderen Formate haben keine große Bedeutung oder sind nicht auf allen Plattformen verfügbar. Für den schnellen Zugriff ist das MP3-Format ideal. Es bietet verschiedene Qualitätsstufen bei weitaus geringerem Platzbedarf und ist aufgrund dieser Eigenschaften auch sehr weit verbreitet. Dadurch werden mittelfristig das AAC- und das OGG-Format trotz geringfügig besserer Qualität vermutlich auch keine bedeutende Rolle spielen.
Die ausgewählten Formate sind auf allen Plattformen verfügbar (DOS, Windows, AIX, BeOS, FreeBSD, HP-UX, Linux , MACOS, OS/2, QNX, SGI, Solaris) und Programme zum Abspielen werden meist vom Betriebssystem bereitgestellt.
Tabelle 2.2 listet weitere bekannte Anwendungen zum Abspielen für die einzelnen Betriebssysteme auf.

| Anwendung | Bezugsquelle, unterstützte Betriebssysteme |
|---|---|
| Windows Media Player | http://www.microsoft.com/windows/windowsmedia/ |
| | Windows, Solaris, MacOS – Freeware |
| Quicktime | http://www.apple.com/quicktime |
| | Windows, MACOS – Freeware |
| RealPlayer | http://www.real.com |
| | Windows, Linux, Solaris – Freeware |
| WinAmp | ftp://ftp-eude.netscape.com/pub/blind/partner/winamp/ |
| | Windows – Freeware |
| Sonique | http://sonique.lycos.com/ |
| | Windows – Registrierung notwendig |
| mp3PRO | http://www.thomson-multimedia.com/ |
| | Windows, MacOS – Freeware |
| xmms | http://www.xmms.org/download.html |
| | Linux – Freeware |
| Freeamp | http://www.freeamp.org/ |
| | Linux, Windows - Freeware |
| mpg123 | http://www.mpg123.de/mpg123/mpg123-0.59r.tar.gz |
| | Linux, FreeBSD, Sun, HP-UX, SGI Irix – Freeware |
| | http://www.users.wineasy.se/nil/mpg123/ |
| | OS/2 – Freeware |
| Mint Audio | http://www.unsanity.com/products.php |
| | MacOS – Shareware |
| MPEG Layer 3 – Player | ftp://ftp.iis.fhg.de/pub/layer3/mpl3_14b2.hqx |
| | MacOS – Freeware |
| SoundJam MP | http://www.soundjam.com/dl/ |
| | MacOS – Freeware |
| MacAmp | http://www.subband.com/macamp |
| | MacOS – Shareware |
| MPEG Audio Player | http://www3.pair.com/odreer/mpeg.html |
| | MacOS – Freeware |
| PM123 | http://www.teamos2.sci.fi/pm123/pm123101.zip |
| | OS/2 – Freeware |
| SoundPlay | http://www.xs4all.nl/~marcone/be.html |
| | BeOS – Freeware |
| Media Player | BeOS , gehört zum Betriebssystem |
| Mpxplay | http://www.geocities.com/mpxplay/ |
| | DOS – Freeware |
| Damp | http://www.damp-mp3.co.uk/ |
| | DOS - Freeware |

**Tabelle 2.2: Anwendungen zum Abspielen von Audiodateien**

Um MP3-Dateien zu erstellen, ist ein Programm nötig, welches mit Hilfe eines sogenannten MP3-Codecs die WAV-Daten komprimiert. Tabelle 2.3 listet einige dieser Anwendungen auf. Die dabei verwendeten MP3-Codecs sind auch getrennt erhältlich. Die Verwendung einiger dieser Codec ist aber in Deutschland wegen lizenzrechtlicher Probleme nicht gestattet.
Tabelle 2.4 zeigt eine Auswahl von MPEG Audio Layer-3 Codecs.

| **Anwendung** | **Bezugsquelle, unterstützte Betriebssysteme** |
|---|---|
| mp3enc | ftp://ftp.iis.fhg.de/pub/layer3/mp3encdemo_3_1_win32.zip |
| | Windows, Linux – Freeware |
| | ftp://ftp.iis.fhg.de/pub/layer3/mp3encdemo_3_1_solaris.tgz |
| | Solaris – Freeware |
| | ftp://ftp.iis.fhg.de/pub/layer3/mp3encdemo_3_1_irix.tgz |
| | Irix – Freeware |
| MusicMatch Jukebox | http://www.musicmatch.com |
| | Linux, MACOS – Freeware |
| GoGo | http://www.daszler.de/linux/shots/gogo.html |
| | Linux – Freeware |
| | http://bezip.de/app/21/ |
| | BeOS – Freeware |
| WinDAC | http://www.windac.de |
| | Windows – Freeware |
| krabber | http://krabber.automatix.de |
| | Linux – Freeware |
| SoundApp | http://www.spies.com/~franke/SoundApp/ |
| | MacOS – Freeware |
| XingMP3 Encoder | http://www.xingtech.com/ |
| | Windows – Registrierung notwendig |

**Tabelle 2.3: Audiokonverter**

| Audio-Codecs | Bezugsquelle, unterstützte Betriebssysteme |
|---|---|
| LameEnc | |
| | http://www.sulaco.org/mp3 |
| | Linux, Unix, Windows, DOS, MacOS, BeOS, riskOS, AmigaOS, OS/2 – Freeware |
| BladeEnc | |
| | http://bladeenc.mp3.no |
| | BeOS, NeXT, Net-/Open-/FreeBSD, HP-UX, Linux, QNX, Sun/Solaris, Windows, AIX, DEC, Irix, MacOS, DOS, OS/2 – Freeware (aber illegal in Deutschland) |
| Windows Media Codec | |
| | http://www.microsoft.com |
| | Windows, UNIX, MacOS – Freeware |
| NotLame | |
| | http://www.idiap.ch/~sanders/not_lame/ |
| | Linux, Solaris, AIX, NetBSD, FreeBSD - Freeware |
| Fraunhofer IIS MPEG Layer-3 Codec | |
| | http://www.iis.fhg.de/amm |
| | Windows, MacOS, Linux, Sun Solaris, Irix, Alpha – Freeware |
| DivX Audio Codec | |
| | http://www.divx.com |
| | Windows, MacOS, Linux – Freeware |
| Kristal Studio | |
| | http://www.soft-ware.net/multimedia/video/kompression/programme.asp?p02148 |
| | Windows – Freeware |

**Tabelle 2.4: Codecs für Audiodateien**

Auch beim Konvertieren von Audiodaten kann es durch die Änderung der Abtasttiefe oder Abtastfrequenz zu Informationsverlust ohne Speicherplatzeinsparungen kommen. Deswegen sollten die Audiodateien von den Autoren mit möglichst hoher Qualität eingebracht werden. Die Mindestanforderungen an die Formate sollten 8 Bit Abtasttiefe und 11025 Hz Abtastfrequenz sein. Die ausgewählten Formate können nebenbei auch leicht für die Erstellung von Audio-CDs verwendet werden.

## *2.5 Video*

Im folgenden werden die verschieden Formate für Videodaten betrachtet. Im Anschluss folgt eine Auswertung. Auch hier gilt, dass hier nicht vorgestellte Formate im Allgemeinen wenig verbreitet sind, oder den Anforderungen nur unzureichend genügen. Hierzu zählen zum Beispiel Formate, die keine Unterstützung für kombinierte Audio- und Videodaten vorzeigen und die nicht für hoch qualitative Aufnahmen geeignet sind.

### 2.5.1 AVI

Das „Audio/Video Interleaved“ - Format ist ein Dateiformat zum Austausch, Aufnehmen, Bearbeiten und Abspielen von Audio- und Videodaten auf dem PC. Es wurde von Microsoft entwickelt und gilt durch seine weite Verbreitung als Quasi-Standard. AVI-Dateien können mehrere (un-)komprimierte Audio- und (MPEG-) Videodatenströme enthalten. Die Daten können durch beliebige externe Audio- und Video-Codecs komprimiert werden.
Der MIME-Typ kann „video/avi“ als auch „video/[x-]msvideo“ sein.

### 2.5.2 MOV

Das MOV-Format wurde ursprünglich für den *Apple Macintosh* entwickelt. Es wird benutzt, um QuickTime Filme und andere zeitbasierte Daten zu speichern und auszutauschen. Verfügbar ist das Format für Windows, MacOS und UNIX („*xanim*“), aber hauptsächlich nur unter Windows und MacOS verbreitet.
Der MIME-Typ ist „video/quicktime“.

### 2.5.3 MPEG

Das MPEG-Format wurde von der *Moving Picture Expert Group* entwickelt. Es dient zum Speichern und Streamen von Audio- und Videodaten. Eine Anwendung ist unter anderem das digitale Fernsehen. Die Audiodaten werden dabei in bis zu 5 Kanälen mit jeweils 48 kHz, 44,1 kHz oder 32 kHz Abtastfrequenz und bis zu 24 Bit Abtasttiefe gespeichert. Die Videodaten werden in verschiedenen Unterformaten als MPEG-1, MPEG-2 oder MPEG-4 codiert. Der MIME-Typ ist „video/mp[e]g“.

### 2.5.4 Real-Player-Formate

Die Firma *RealNetworks* hat für ihren RealPlayer eigene Formate entwickelt zur Darstellung von Audio- und Videodaten über das Internet. Diese sind zum Beispiel RealAudio (siehe Kapitel 2.4.7.), RealPlayer-Video (*.rms, .rv*) und RealPlayer Flash (*.rf*). Dabei sind verschiedene Qualitätstypen möglich: Modem (ab 34 Kbps), VHS (ab 160 Kbps) und DVD-Qualität (ab 400 Kbps). Den RealPlayer gibt es für Windows, Linux, UNIX, QNX und MacOS. Der MIME-Typ ist sehr unterschiedlich. Einige Typen sind „application/vnd.rn-realmedia“, „video/vnd.rn-realvideo“ (RealPlayer-Video) und „image/vnd.rn-realflash“ (RealPlayer Flash). “.
Um Real Video-Dateien ohne Einschränkungen zu erstellen, ist auch eine kommerzielle Software von RealNetworks nötig, wodurch das Format für den Dokumentenserver nicht geeignet ist.

### 2.5.5 WMV

Das „Windows Media Video“ - Format ist ein von Microsoft speziell für den Windows Media Player entwickeltes Format zum Abspielen und Download (Streaming) von (un-) komprimierten Audio- und Videodaten. Die Daten werden dabei mit Hilfe der Windows Media Codecs komprimiert, die es nur für Windows gibt. Für Audiodaten sind beliebige Bitraten bei einer Samplerate von 8 kHz bis 48 kHz möglich. Für Videodaten sind Bitraten ab 28,8 Kbit (inklusive Audiodaten) bei unterschiedlichen Auflösungen, Frameraten und Videonormen (PAL, NTSC) möglich. Das Betrachten ist nur auf den Plattformen möglich, für die es den Windows Media Player gibt (DOS, Windows, Sun, HP, MacOS). Optional können Informationen über den Inhalt gespeichert werden. Auch hier kommt das Digital Rights Management (DRM) zur Anwendung. Der MIME-Typ ist „video/x-ms-wmv“.

### 2.5.6 FAZIT

Während Bild- und Audiodaten in jeweils zwei Formaten vorgehalten werden, sollte es aufgrund der immensen Datenmengen ausreichen, Videodaten in nur einem Format anzubieten. Der Aufwand für die Speicherung eines weiteren Formats würde den Nutzen nur minimal vergrößern.
Durch die weite Verbreitung und Kompatibilität zu anderen Anwendungen fällt die Wahl auf das AVI-Format. Es ist auf fast allen Plattformen verfügbar und Programme zum Abspielen werden meist vom Betriebssystem bereitgestellt. Zusätzlich soll es aber auch möglich sein, Videodaten im MPEG-, Windows Media- und Real Video-Format einzureichen.
Tabelle 2.5 listet weitere bekannte Anwendungen zum Abspielen von Videodaten für die einzelnen Betriebssysteme auf.

| Anwendung | Bezugsquelle, unterstützte Betriebssysteme |
|---|---|
| Media Player | |
| | http://www.microsoft.com/windows/windowsmedia/ |
| | Windows, Sun, MacOS – Freeware |
| Real Player | |
| | http://www.real.com |
| | Windows, Linux, SGI, Sun, QNX – Freeware (Basisvariante) |
| xanim | |
| | http://smurfland.cit.buffalo.edu/xanim/home.html |
| | Linux, Sun, DEC, SGI, HP-UX, AIX, QNX – Freeware |
| Media Player | |
| | BeOS, gehört zum Betriebssystem |

**Tabelle 2.5 : Anwendungen zum Abspielen von Videodateien**

Um den Platzbedarf der Videodaten weiter zu verringern, sollte man zusätzlich einen Video-Codec verwenden. Das AVI-Format kann verschiedene Video- und Audio-Datenströme enthalten, die mit Codecs komprimiert werden können. Für die Archivierung soll den Autoren empfohlen werden, einen Standard zu verwenden, z.B. MPEG-1 (Super-/Video-CD), MPEG-2 (DVD-Format) oder MPEG-4.
Damit man AVI-Dateien im platzsparenden MPEG-4 Format erstellen kann, ist ein MPEG-4 Encoder notwendig.
Tabelle 2.6 zeigt eine Auswahl von Anwendungen, die diese Konvertierung mit Hilfe externer Codecs ermöglichen.

| Anwendung | Bezugsquelle, unterstützte Betriebssysteme |
|---|---|
| Windows Media Encoder | http://www.microsoft.com/windows/windowsmedia/en/software/ |
| | Windows – Freeware |
| Xing MPEG | |
| | http://www.xingtech.com/ |
| | Windows – Shareware |
| VirtualDub | |
| | http://www.virtualdub.org |
| | Windows – Freeware |
| FlasK MPEG | |
| | http://go.to/flaskmpeg |
| | Windows – Freeware |

**Tabelle 2.6: Konverter für Videodaten**

Die dabei verwendbaren Codecs sind meistens auch getrennt erhältlich. Tabelle 2.7 listet einige Codecs für die Video-Komprimierung auf. Die Audio-Komprimierung kann mit Hilfe der in Kapitel 2.4. vorgestellten Audio-Codecs erfolgen.
Aufgrund der unterschiedlichen Codecs ist es wichtig zu wissen, mit welcher Software und welchem Codec ein Video erstellt wurde. Diese Information sollte der Autor bei der Einbringung seiner Dokumente auf jeden Fall mit angeben. Wenn man sich bei der Erstellung der Videodaten an entsprechende Standards hält, kann man später die Videodaten in Form der AVI-Dateien auch als Video-CD bereitstellen. Eine Grundlage bilden dabei der Video-CD-Standard (VCD) und das Super-Video-CD (SVCD) Format.

**Video-CD**

Audio: 44,1 kHz, 224 kBit/s; inklusive Videodaten 1123 Kilobit/Sekunde
Video: MPEG-1 mit 352x240 oder 704x480 Pixel bei 29,97 Hz (NTSC) oder
352x240 Pixel bei 23,967 Hz (PAL) oder
352x288 oder 704x576 Pixel bei 25 Hz (PAL)

**Super-Video-CD**

Audio: 44,1 kHz, 224 kBit/s; inklusive Videodaten 2600 Kilobit/Sekunde
Video: MPEG-2 mit 480x576 oder 704x576 Pixel (PAL) oder
480x480 oder 704x480 Pixel (NTSC)

| **Video-Codecs** | **Bezugsquelle, unterstützte Betriebssysteme** |
|---|---|
| CinePak Codec by Radius | |
| | http://www.cinepak.com/ |
| | Windows, MacOS – Shareware |
| Indeo Codec | |
| | http://www.ligos.com/ |
| | Windows, MacOS – Shareware |
| Windows Media Codec | |
| | http://www.microsoft.com/downloads |
| | Windows, UNIX, MacOS – Freeware |
| DivX Video Codec | |
| | http://www.divx.com |
| | Windows, MacOS, Linux – Freeware (Basisvariante) |
| OpenDivX | |
| | http://www.projectmayo.com/projects |
| | Windows, MacOS, Linux – Freeware |
| Kristal Studio Codecs Pack | |
| | http://www.stokebloke.com/video/ |
| | Windows – Freeware |

**Tabelle 2.7: Auswahl von Codecs für Videodaten**

# 3 Konzept für zusammengesetzte Dokumente

Bisher ist jede Publikation auf dem Dokumentenserver eine zusammenhängende Postscript- bzw. PDF-Datei. Das heißt, beim Archivieren muss der Autor seine Arbeit mit allen zugehörigen Anhängen zu einer einzigen Datei zusammenfassen. Dieses Dokument ist dann auch unter einer eindeutigen Dokument-ID archiviert. Bei einfachen Abbildungen, die dem Verstehen des Textes dienen, ist das in eingeschränkter Weise auch ausführbar. Aber sobald man Multimediadaten zur Verfügung stellen will, die man nicht „liest“, zum Beispiel Musik oder Video, ist das nicht mehr möglich.

Ziel ist es also, ein Konzept zu entwickeln, mehrere Einzeldokumente zu einem Gesamtdokument zusammenzufassen. Dieses Gesamtdokument ist dann weiterhin unter der Dokument-ID auf dem Dokumentenserver abrufbar. Beispiele für zusammengesetzte Dokumente sind in diesem Zusammenhang unter anderem Vorlesungsskripte. Diese bestehen meist aus verschiedenen Kapiteln, beziehungsweise lassen sich gut in Kapitel aufteilen. Das Gesamtdokument ist in diesem Fall das vollständige Skript, während die Einzeldokumente die jeweiligen Kapitel darstellen und sich somit auch im Einzelnen betrachten oder referenzieren lassen.
Ein anderes Beispiel ist die Bild- und Signalverarbeitung. Hier arbeitet man oft mit mehrfarbigen Bilddaten von großer Detailgenauigkeit. Diese Abbildungen lassen sich jetzt einzeln in der benötigten Qualität archivieren. Im Text braucht man dann nur auf die jeweiligen Dateien verweisen.

Wenn man Audiodaten ohne hörbare Beispiele archivieren will, kann man das bisher im Bereich der Musikwissenschaften nur in Form von Notendokumenten. In technischen, aber auch in anderen Bereichen, in denen ein Ton nicht von einem Musikinstrument erzeugt wird, lassen sich Töne durch die Angabe der Frequenz bestimmen. Nunmehr ist es möglich, diese Audiodaten direkt als Hörprobe zu archivieren. Die einzelne Audiodatei ist dann wiederum ein Teildokument der Gesamtarbeit. Mit der Verwendung von Videodaten als Teil eines Gesamtdokuments ist es möglich, Videosequenzen, Animationen oder Simulationen einzubringen, die zum besseren Verständnis dienen. Diese Teildokumente können dann – aufgrund ihrer Größe – einzeln betrachtet beziehungsweise heruntergeladen werden.

Eine weitere Einsatzmöglichkeit besteht darin, ein zusätzliches Dokument beliebigen Formats zu archivieren. Das kann zum Beispiel ein Word-Dokument oder eine PowerPoint-Präsentation sein. Ebenso ist ein Format denkbar, das nicht in die vier bisherigen Bereiche Text, Bild, Ton und Video passt und somit für zukünftige Anwendungen verfügbar ist. Da eine Universität viele Fachbereiche hat, gibt es unterschiedliche Daten aus vielen verschieden Anwendungen. Bauingenieure würden zum Beispiel Daten aus Konstruktions- und Statikprogrammen einbringen können.

Dieses zusätzliche Dokument soll aber nur eine Option darstellen. Ein einzureichendes Textdokument sollte immer aus mindestens einer Postscript- oder PDF-Datei bestehen. Diese Formate sichern eine hohe Druckqualität, die sich nicht von der digitalen Form unterscheidet. Das heißt, so wie das Dokument am Bildschirm aussieht, erscheint es auch auf dem Papier. Ein weiteres Ziel bei der Entwicklung des Dokumentenservers war es, die Authentizität der Dokumente zu sichern. Damit ist gemeint, dass man sie nicht trivial verändern oder manipulieren kann. Diese Eigenschaften können bei den zusätzlichen Formaten nicht sichergestellt werden.

Weiterhin soll es möglich sein, die Teildokumente mit einer Angabe zum Autor, Titel und einem individuellen Kommentar zu versehen. Dies kann die Kapitelüberschrift sein oder der Name der Multimediadateien, auf die im Text referenziert wird. Bei anderen Gesamtdokumenten mit gemeinsamen Bezug, zum Beispiel Tagungsbeiträge, können die einzelnen Beiträge mit den Namen der Autoren und dem Thema der jeweiligen Arbeit kommentiert werden. Zusätzlich können zu den Multimediadaten deren Eigenschaften angegeben werden. Eine kurze Beschreibung und technische Daten, wie Auflösung bei Bildern, Dateigröße, Spieldauer und verwendeter Codec bei Audio- und Videodaten können sehr hilfreich sein.

Im Folgenden wird noch einmal zusammengefasst, welche Dateiformate als Teildokumente eingebracht werden können und welche Dateiformate daraus erzeugt werden.
Bei den Volltextformaten können weiterhin Postscript- oder PDF-Dateien eingebracht werden. Das jeweilige andere Format wird – soweit möglich – automatisch erzeugt, ebenso wie das ASCII-Format und eine seitenweise Vorschau als Bilder im GIF-Format.
Die Bilddaten können im PNG- oder JPG-Format eingereicht werden, wobei wiederum das jeweils andere Format automatisch erzeugt wird. Die Audiodaten können wahlweise im WAV-Format oder im MP3-Format eingebracht werden, es erfolgt ebenso eine automatische Konvertierung in das fehlende Format.
Videodaten können im AVI-, MPEG-, Windows Media- oder RealVideo-Format eingereicht werden. Dabei erfolgt aber keine Konvertierung aufgrund der Größe und unterschiedlichen Formateigenschaften.
Als Zusatzoption kann ein beliebiges Dateiformat eingebracht werden, hierbei erfolgt aber aufgrund der Vielzahl von Formaten auch keine Konvertierung.
Alle Dateiformate können außerdem komprimiert im GZIP- oder ZIP-Format übertragen werden.

Zusätzlich können alle Teildokumente eines Dokumentes als komprimierte ZIP-Datei heruntergeladen werden. Bei mehreren verfügbaren Formaten werden jeweils Postscript-, JPG- und MP3-Format bevorzugt.

# 4 Aufbau des bestehenden Systems

Dieser Abschnitt beschreibt die Details des zugrunde liegenden Dokumentenservers. Seine Architektur basiert auf dem Schichtenmodell für Dokument-Archivierungssysteme von Crespo et al [12]. Die Grundidee bei diesem Modell ist, die Komponenten eines solchen Systems in Schichten zu organisieren. Diese bieten genau definierte Schnittstellen zu den darüber liegenden Schichten an. Jede Schicht kann ausgetauscht werden, ohne die anderen Schichten zu beeinflussen. Abbildung 4.1 zeigt die einzelnen Komponenten des Systems.

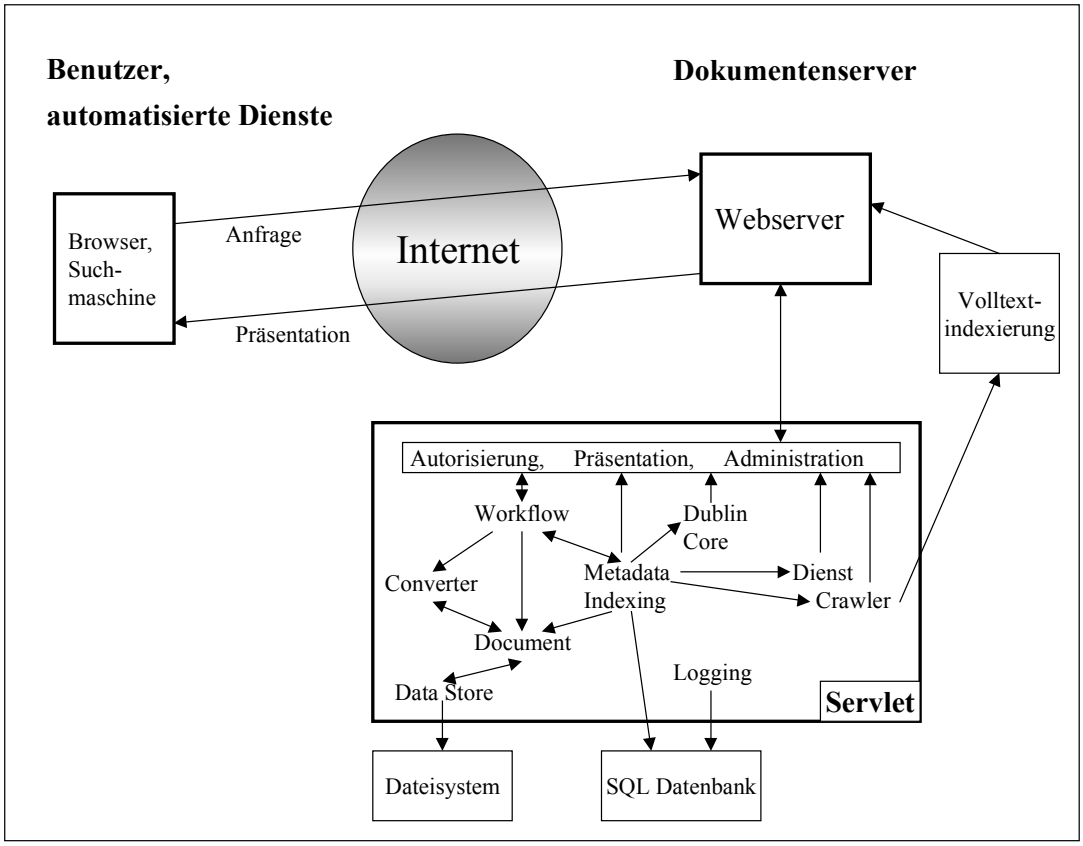


**Abbildung 4.1: Die Komponenten des Dokumentenservers**

Der zentrale Bestandteil des Serverstandorts ist ein *Java Servlet*, das die meisten Verwaltungs- und Präsentationsaufgaben übernimmt. Dieses Servlet wird durch einen Webserver ausgeführt. Die Textdokumente werden mit Hilfe der sogenannten *Data Store*-Komponente (siehe Kapitel 5.1) gespeichert, die eine Schnittstelle zum Dateisystem bildet. Es kann aber genauso durch eine andere Implementation ersetzt werden, die zum Beispiel BLOBS in einer Datenbank speichert. Gegenwärtig unterstützte Volltext-Formate zur Darstellung und zum Download sind sowohl komprimierte als auch unkomprimierte PostScript-, PDF- und ASCII-Dateien. Weiterhin gibt es eine seitenweise Vorschau der Dokumente mit Hilfe von Bildern im GIF-Format. Diese Vorschaubilder werden mit dem Tool *Pscript* von Günther Radestock erzeugt. Zusätzlich werden die bibliographischen und administrativen Metadaten der Dokumente im Dateisystem gespeichert. Die Metadaten werden mit Hilfe einer SQL-Datenbank indiziert, um die Suche nach Dokumenteigenschaften effektiv zu ermöglichen. Aber auch Arbeitsabläufe und Zugriffsberechtigungen werden so organisiert. Die Volltextsuche erfolgt über ein externes Tool zum Indizieren; gegenwärtig wird das kostenlose *htDig* [14] verwendet.
Die *Document*-Komponente dient dazu, auf die Metadaten der Dokumente zuzugreifen, und diese mit Hilfe der LRU (Least recently used) Methode in einem Cache zu speichern. Die *Metadata Indexing*-Komponente ermittelt die relevanten Metadaten für Suchabfragen und repliziert sie in einer SQL-Datenbank.
Die *Workflow*-Komponente ist für den Ablauf beim Einbringen und Administrieren der Dokumente verantwortlich. So wird zum Beispiel die *Converter*-Komponente aufgerufen, um die Dokumente automatisch in andere Formate zu konvertieren.
Der Dokumentenserver liefert mehrere Schnittstellen für Suchmaschinen und externe Dienste wie NCSTRL [15]. Diese Schnittstellen werden von den obengenannten Servlet-Bestandteilen unterstützt. Die *Crawler*-Schnittstelle wird außerdem dazu verwendet, um einen Volltextindex der Sammlung mit Hilfe von *htDig* zu erstellen.
Die *Presentation*-Komponente dient dazu, mit Hilfe der Metadaten HTML-Seiten zu generieren.

## *4.1 XML-basierte Konfiguration*

Zur Erleichterung der Konfiguration und Instandhaltung des Dokumentenservers sind die meisten der Konfigurationsoptionen explizit in einer XML-Datei gespeichert. Genau wie HTML ist die „Extensible Markup Language“ (XML) eine vom World Wide Web Consortium (W3C) reglementierter Variante des reichlich komplexen Dokumentations- und Textstandards SGML (Standard Generalized Markup Language). Im Gegensatz zu HTML sind Autoren bei XML aber nicht an statisches Markup gebunden. Die sogenannten Tags und Attribute lassen sich im Prinzip beliebig definieren. Dazu bedarf es einer geeigneten Dokumententypdefinition (DTD), Extensible Stylesheet Language (XSL) oder eines XML-Schemas. Aufgrund ihrer Flexibilität wird XML auch als Metasprache bezeichnet, die mittlerweile den De-facto-Standard für den offenen Datenaustausch, zum Beispiel in der Industrie, darstellt.

Durch die Verwendung von XML ist der Dokumentenserver kompatibel zu anderen Anwendungen und offen für zukünftige Erweiterungen.
Für den Dokumentenserver enthält diese Datei die folgende Information:

- Definitionen von bibliographischen Feldern
- mehrsprachige Textelemente
- Berechtigungsinformation
- Workflow- und Umwandlungsparameter
- verschiedene Systemparameter, wie Datenbankkonfiguration, maximale Anzahl von Suchergebnissen, NCSTRL Parameter, Speicherort von Protokolldateien usw.

Die Namen aller Textelemente, die auf den vom Server generierten HTML-Seiten erscheinen, sind in der Konfigurationsdatei enthalten. Dieser Ansatz erlaubt es, den Server international leicht einzusetzen. Gegenwärtig werden Schnittstellen für Englisch und Deutsch unterstützt.

## *4.2 Browsing und Navigation*

Die Navigationskomponente ist darauf ausgerichtet, den Benutzern einen schnellen Überblick von Inhalt und Umfang der Dokumentensammlung zu geben. Dies wird durch eine Client-basierte JavaScript Anwendung realisiert. So können alle Dokumente auf einer Ebene (Organisation, Sprache, Dokumentart oder Erscheinungsjahr) sortiert ausgegeben werden. Auf einer zweiten Ebene wird dann nach den jeweils anderen Ebenen unterteilt. Es kann also zum Beispiel erst nach einer bestimmten Dokumentart gesucht werden und in dieser Ergebnismenge nach allen Dokumenten eines Jahrgangs.
Dabei wird jedes Mal die Anzahl der Dokumente in der entsprechenden Ebene angezeigt. Diese Informationen werden als kompakte JavaScript-Datenbank beim Aufrufen der Seite übertragen.

## *4.3 Metadaten und Indizierung*

Jedes Dokument auf dem Dokumentenserver umfasst mehrere Volltext-Versionen und zwei Metadaten-Dateien. Die erste Datei ist die sogenannte BIB-Datei [16], welche die bibliographischen Daten des Dokuments enthält. Beispiel für eine BIB-Datei:

```
BIB-VERSION:: CS-TR-v2.1
ID:: 1998-43
AUTHOR:: Wetzer, Barbara;Pfadler, Alexander; Györvary, Erika; Pum,
Dietmar; Lösche, Mathias; Sleytr, Uwe B.
TITLE:: S-layer reconstitution at phospholipid monolayers
CITATION:: Langmuir, 14 (1998),6899-6906
PAGES: 21
LANGUAGE:: en
ORGANIZATION:: LEI.12
TYPE:: LEI-Preprint
DISCIPLINE:: DNB-5-29;DNB-5-30
...
```

Das Beispiel zeigt eine Veröffentlichung mit dem Titel "S-layer reconstitution at phospholipid monolayers“, gespeichert auf dem Dokumentenserver in Leipzig mit der Dokument-ID „1998-43“. Die komprimierten PostScript- und PDF-Dateien sind im *Data Store* als Dateien *43.ps.gz* und *43.pdf.gz* gespeichert. Die bibliographischen Daten sind in der Datei *43.bib* enthalten.

Die zweite Datei mit der Bezeichnung *43.state* enthält die Informationen für Administration und Workflow-Zustände, welche in XML gespeichert sind.
Hier der Auszug aus einer Beispieldatei:

```
<state version="1.0" contact-email="loesche@physik.uni-leipzig.de"
doc_id="1998-43" submission-date="1998-10-23">
      <converted PDF="yes" PS="original" ASCII="yes"
                  graphics="yes" reverse="no" color="yes"/>
      <workflow state="legitimate"/>
      <copyright submission-date="1998-11-09"/>
      <history>
...
</state>
```

Die Metadaten zeigen an, dass das Dokument am 23. Oktober 1998 im PostScript-Format eingereicht wurde und sich in einem "legitimen" Zustand befindet, d.h. der Administrator hat die Copyrighterklärung für das Dokument am 9. November 1998 erhalten. Die PostScript-Datei ist in PDF und ASCII umgewandelt worden und die graphische Vorschau wurde erstellt, was durch das XML-Tag <converted> anzeigt wird.
Die Aufgabe der Komponente für *Metadata* und *Indexing* ist es, die bibliographischen Metadaten der Dokumente zu extrahieren und sie mit Hilfe der JDBC-Schnittstelle in einer relationalen Datenbank abzuspeichern. Dabei werden die Informationen aus den BIB-Dateien gesammelt und für die spätere Suche nach bibliographischen Daten verwendet. Die Indizierung läuft auch als separater Prozess, welcher periodisch überprüft, ob Dokumente verändert worden sind. Wenn dies der Fall ist, werden die bibliographischen Daten des Dokuments neu indiziert.
Die administrativen Metadaten werden in erster Linie dazu verwendet, um festzulegen, welche Dokumente öffentlich sind, d.h. während Suche und Navigation gefunden werden können. Dokumente ohne eingegangene Copyrighterklärung sind nicht für die Öffentlichkeit freigegeben.

## *4.4 Workflow und Formatumwandlung*

Während seiner Lebensdauer wechselt ein Dokument mehrere Workflow-Zustände. Diese einzelnen Zustände, welche in der XML-Datei gespeichert werden, sind in Abbildung 4.2 zusammengefasst.

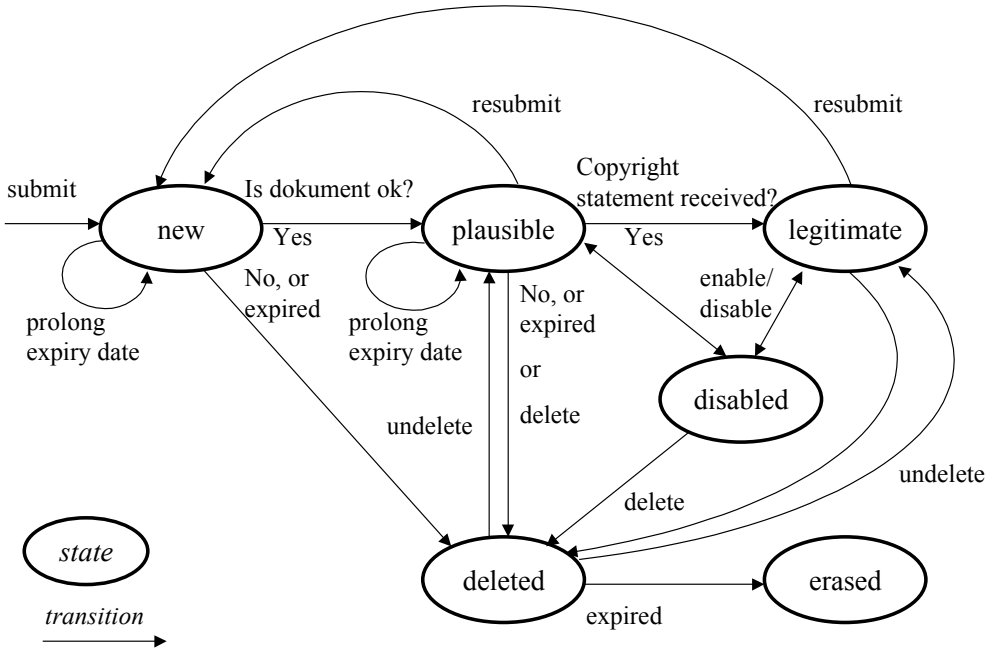


**Abbildung 4.2: Der Workflow-Graph eines Dokumentes**

Ein neu eingereichtes Dokument wird als "neu" gekennzeichnet. Sofort beginnt ein automatischer Umwandlungsprozess im Hintergrund. Der Konverter versucht, das fehlende Format zu generieren. Liegt das Dokument beispielsweise im PostScript-Format vor, wird die Konvertierung in das PDF-Format gestartet. Gelingt diese Umwandlung in einem bestimmten Zeitintervall nicht, wird der Prozess abgebrochen. Wurde das eingereichte Dokument vom Administrator überprüft, wechselt es in den Zustand "plausibel".
Nachdem er die Copyrighterklärung vom Autor erhalten hat, wird das Dokument "legitim". In jedem Zustand, außer "legitim", verfällt das Dokument nach einer bestimmten Zeit, wenn die erforderlichen Erklärungen nicht eingereicht werden. Erst nach einem weiteren Zeitraum wird das Dokument physisch gelöscht, während dessen es erneut aktiviert werden kann.
Ein weiterer Workflow-Zustand ist die zeitweilige Sperrung einzelner Dokumente für die Öffentlichkeit. Die Sperre kann vom Administrator zum gegebenen Zeitpunkt wieder aufgehoben werden. Zusätzlich kann der Autor privilegiert werden, Änderungen an seinem Dokument vorzunehmen. Falls einmal ein Dokument beschädigt sein sollte, kann es vom Autor erneut eingereicht werden, ohne die bibliographischen Daten erneut eingeben zu müssen.

## *4.5 Schnittstellen für den Export und Volltextsuche*

Die *Crawler*-Schnittstelle exportiert die bibliographischen Metadaten und die Dokumente im ASCII-Format auf Anfrage an die jeweiligen Dienste. Ein *Crawler* ist zum Beispiel eine Suchmaschine. Immer wenn auf ein Dokument zugegriffen wird und das "User Agent"-Feld in der HTTP-Anfrage auf einen Suchroboter schließt, werden die Metadaten und der Volltext-Inhalt in einem kompakten Format geliefert, das sich von der dem menschlichen Benutzer gezeigten Seite unterscheidet.
Außer der Crawler-Schnittstelle bietet der Dokumentenserver noch zwei andere Arten an, auf die Metadaten zuzugreifen. Die HTML-Seiten enthalten – außer den für die Ansicht notwendigen Beschreibungen – eingebettete Dublin Core-Metadaten im RDF/XML-Format [17]. Dublin Core-Metadaten werden benutzt, um bestehende Methoden zur Suche und Indizierung von Web-basierten Metadaten zu erweitern. Diese Metadaten beschreiben Objekte im Internet zur einfachen Katalogisierung in Information Retrieval-Systemen.
Eine interaktivere Schnittstelle für den Zugriff auf Dokument-Metadaten wird über das *Dienst*-Protokoll angeboten. Dieses Protokoll, welches Metadaten inkrementell abfragen kann [18], wird beim NCSTRL-System verwendet. Die *Dienst*-Komponente übermittelt dabei die schrittweisen Änderungen in der Dokumentensammlung.
Die Volltextsuche und -indizierung der Dokumentsammlung wird durch eine lose gekoppelte, vom Dokumentenserver unabhängige, interne Suchmaschine realisiert. Diese Suchmaschine auf dem Dokumentenserver wird periodisch als ein separater Prozess aufgerufen. Dabei werden die Volltext-Inhalte aller öffentlich freigegebenen Dokumente abgerufen und daraus ein Volltext-Index generiert.

# 5 Systemerweiterungen im Allgemeinen

Während das vorhergehende Kapitel einen Überblick über das bestehende System gibt, zeigt der folgende Abschnitt, welche vorhandenen Komponenten des bestehenden Systems erweitert werden müssen und wie dies im Einzelnen erfolgt.

## *5.1 Data Store*

Der *Data Store* ist ein System zum Archivieren von Daten mit unter anderem folgenden Eigenschaften: persistente Speicherung von Daten sowie Autorisationskontrolle für Lese- und Schreibzugriffe. Dieses System kann zum Beispiel ein Dateisystem oder eine Datenbank sein. Die geforderten Eigenschaften stellt meistens das darunter liegende Verwaltungssystem sicher – zum Beispiel Betriebssystem oder Datenbanksystem. Von diesem Verwaltungssystem übernimmt der Dokumentenserver die Kontrolle der Zugriffsrechte und Persistenz der Daten.
Dem bestehenden Dokumentenserver liegt eine Archivierung im Dateisystem zugrunde. Dabei wird eine Verzeichnisstruktur verwendet, die sich nach dem Zeitpunkt der Einbringung eines Dokumentes durch den Autor richtet. Die oberste Ebene bildet dabei das Jahr. Im Laufe eines Jahres werden die Dokumente fortlaufend in der Reihenfolge des Einbringens nummeriert. Daraus ergibt sich die Dokument-ID. Im Dateisystem wird das durch eine Verzeichnisstruktur realisiert. Das eigentliche Dokument kann dann durch Dokument-ID und Angabe des Formates (Postscript, PDF, Text) abgerufen werden. Auf die gleiche Weise werden auch die bibliographischen und administrativen Metadaten gespeichert.
Durch die Unterteilung eines Gesamtdokuments in mehrere Teildokumente, sind in dieser Struktur einige Änderungen notwendig. Die Dokument-ID steht jetzt für eine Publikation, die aus mehreren Kapiteln und Multimediadaten - nachfolgend nur als Kapitel bezeichnet - bestehen kann. Dabei genügt es nicht mehr, nur das gewünschte Format anzugeben. Man muss auch das jeweilige Kapitel einbeziehen. Hierbei werden die Kapitel, Bilder, Audio- und Videodateien jeweils einzeln durchnumeriert. Für ein Kapitel sind jetzt also Dokument-ID, fortlaufende Nummer innerhalb des Medienformats (Text, Bild, Ton, Video) und das Dateiformat zur Referenzierung notwendig.

Die Verfügbarkeit und Sicherung der Daten wird durch das zugrunde liegende Speichermedium gewährleistet. Die Dokumente und Metadaten selber werden je nach verwendeten Speichermedium gesichert. Bei der Verwendung eines Dateisystems ist außerdem ein regelmäßiges Backup erforderlich. Aufgrund des erhöhten Platzbedarfes

durch die Multimediadateien ist es sinnvoll, diese einmalig auf persistenten Datenträgern zu archivieren und vom regelmäßigen Backup auszuschließen. Im Recovery-Fall können sie dann nachträglich wieder hergestellt werden.

## *5.2 Metadaten*

An den bibliographischen Metadaten finden keine Änderungen statt, da sie sich auf das Dokument als Ganzes beziehen. Zusätzlich wird aber eine neue dritte Datei für Metadaten erstellt, in der die Struktur des Gesamtdokuments abgespeichert wird. Dabei werden die Multimediaformate einzeln durchnummeriert, und mit Autor, Titel und Kommentar versehen (soweit angegeben). Für Kapitel im „sonstigen Format“ und Videos werden das Dateiformat zusätzlich mit abgespeichert. Diese dritte Datei, welche als *structure*-Datei bezeichnet wird, liegt im XML-Format vor, um sich an das bestehende Konzept der Administrationsdatei anzupassen und für spätere Erweiterungen kompatibel zu sein. Der Aufbau der *structure*-Datei – welche analog zum Beispiel aus Kapitel 4.3 *43.struct* heißt – sieht wie folgt aus:

```
<structure doc_id="1998-43">
     <content>
          // einzelne Kapitel haben Autor, Titel und
               Kommentar (id), evtl. anderes Format
          // Reihenfolge der Kapitel festgelegt ("rank")
          <chapter rank="1" author=".." title=".." id="..">
          <chapter rank="2" author=".." ... format="ppt">
          <chapter rank="3" author=".." ... format="doc">
          ...
          // Bilder
          <image rank="1" author=".." title=".." id="..">
          <image rank="2" author=".." title=".." id="..">
          ...
          // Audiodateien
          <audio rank="1" author=".." title=".." id="..">
          ...
          // Videodateien; Angaben z.B. über Dauer, Größe,
          // Software/Codec, womit die Datei erstellt wurde
          <video rank="1" author=".." title=".." id="..">
          <video rank="2" author=".." ... format="wmv" >
          ...
     </content>
</structure>
```

Eine Änderung ergibt sich an den Daten in der *state*-Datei für die administrativen Metadaten. Hier werden nicht Herkunft und Workflow aller Teil-, sondern nur des Gesamtdokumentes berücksichtigt.
Die Metadaten der Dokumente – und somit auch die neuen Struktur-Daten – werden, wie bisher, zusätzlich in einer Datenbank gespeichert. Dadurch können die Vorteile

einer Datenbank für die Volltextsuche (siehe nächstes Kapitel) und Daten-Export genutzt werden. Die Aktualität der Daten stellt ein regelmäßiger Abgleich mit dem *Data Store* sicher. Ebenso werden eventuelle Inkonsistenzen in der Datenbank vermieden.

## *5.3 Suche nach Multimediadaten*

Durch die Speicherung von multimedialen Inhalten im Dokumentenserver ist es natürlich auch notwendig, gezielt nach einzelnen Daten suchen zu können. Die Suche im Volltextindex wird hier nicht betrachtet, da sie getrennt implementiert ist. Es wird eine Suche nach dem jeweiligen Medienformat durchgeführt. So kann nach Bildern, Ton- und Videodaten sowie Dokumenten mit mehreren Kapiteln gesucht werden. Aber auch die kombinierte Suche nach beispielsweise Bildern *und* Audio-Daten, wobei Videodaten nicht berücksichtigt werden, wird auf diese Weise ermöglicht.
Die Suche kann mit der Angabe eines oder mehrerer Stichwörter verbunden werden. Wird kein Stichwort angegeben, werden alle Dokumente geliefert, die ein Teildokument des gewünschten Formates besitzen. Ansonsten werden die Kommentare und Angaben zu Autor und Titel der einzelnen Teildokumente nach denjenigen Stichwörtern durchsucht, welche der Autor bei der Einbringung des Dokumentes angegeben hat (siehe Kapitel 3 und 5.2). Es ist also ebenfalls Aufgabe des Autors, seine Teildokumente so genau wie möglich zu beschreiben.
Das Auffinden von Multimediadaten erfolgt, wie schon die Suche nach bibliographischen Daten, mit Hilfe der Datenbank. Dabei erfolgt eine Erweiterung des Datenbankschemas. Es wird eine neue Tabelle angelegt, in der alle Teildokumente einzeln aufgeführt sind. Jeder Eintrag beinhaltet Informationen zu Medienformat, Rangfolge sowie Autor, Titel und Kommentar der Kapitel ebenso wie Informationen zur Konvertierung.

## *5.4 Schnittstellen zu anderen Diensten*

Im bestehenden System werden die bibliographischen Metadaten und die Dokumente im ASCII-Format zu den Crawlern exportiert. Dieses Konzept wird auch weiterhin beibehalten. Der Export wird so erweitert, dass die Volltext-Inhalte aller Kapitel eines Dokuments geliefert werden.
Die eigentliche HTML-Seite zu einem Dokument, die die bibliographischen Daten enthält und von der man die einzelnen Quellen abrufen kann, stellt jetzt zusätzliche Informationen für Suchmaschinen bereit, wenn das Dokument Bilder beinhaltet. Dabei wird für jedes Bild im HTML-Code ein `<img>`-Tag erzeugt. Dieses Tag enthält einen eindeutigen Dateinamen abhängig von der Dokument-ID und den bei der Einreichung übermittelten Autor, Titel und Kommentar. Manche Suchmaschinen werten bei der Indizierung von Seiten diese `<img>`-Tags aus und archivieren sowohl die Bildinformationen als auch die Bilder getrennt.
Weiterhin enthalten die HTML-Seiten eingebettete *Dublin Core* Metadaten. Dieser Code ist für Crawler, die Metadaten im RDF/XML-Format erwarten. Hier werden ebenfalls zusätzliche Informationen über die jeweiligen Teildokumente exportiert. Da es keine Änderungen an den bibliographischen Metadaten im BIB-Format gibt - hier

werden nur Angaben über das Gesamtdokument gemacht - , ergeben sich auch keine Modifikationen in der Komponente, in der die Daten im *Dienst Protokoll* exportiert werden.

## *5.5 Konvertierung*

Das im bestehenden Dokumentenserver vorliegende Konzept der automatischen Konvertierung der Dokumente zur Bereitstellung in verschiedenen Dateiformaten soll auch auf die neu hinzukommenden Mediendateien übertragen werden. Die bisherige Konvertierung von Textdateien aus Postscript-Dateien erfolgt durch das Tool *Pscript* von Günther Radestock und soll auch weiterhin eingesetzt werden. Ebenso wird die Postscript-Erzeugung aus PDF-Dateien durch *Adobe Acroread* und die PDF-Erzeugung aus Postscript-Dateien durch den *Acrobat Distiller* beibehalten.
Ein Ziel ist es, für die Konvertierung der Multimediadaten frei verfügbare Software zu verwenden. Diese Programme werden mit den nötigen Parametern über sogenannte *Wrapper* aufgerufen. Mit Hilfe dieser Schnittstellen zum Dokumentenserver können aber auch beliebig andere Programme verwendet werden.
Für Bilddaten eignet sich das Programm *convert* aus dem ImageMagick Toolkit [19] besonders gut. Es kann mittels verschiedener Optionen sehr viele Bildtypen untereinander umwandeln.

Beispiel: convert Testbild.png Testbild.jpg

In dem Beispiel wird die Datei *Testbild.png* vom PNG-Format in das platzsparende JPEG-Format konvertiert.
Bei der Audiokonvertierung bietet sich der *LAME-Codec* an, der sich auch als eigenständiger Konverter einsetzen lässt. Hierbei sind ebenfalls verschiedene Optionen möglich, die sich auf die Qualität der Resultate auswirken.

Beispiel: lame_351_sol26 Audio1.wav [Audio1.mp3]

Das Beispiel zeigt die Konvertierung der Audiodatei Audio1.wav in das MP3-Format. Wie die Konvertierung mit Hilfe des Codecs der Fraunhofer Gesellschaft [20] geschieht, zeigt folgendes Beispiel.

Beispiel: mp3enc -if Audio2.wav -of Audio2.mp3 .

Da diese Tools aber nicht die umgekehrte Richtung der Konvertierung unterstützen, wird für die Erstellung von WAV-Dateien aus MP3-Daten das Tool *mpg123* [21] verwendet. Folgendes Beispiel zeigt die Konvertierung der MP3-Datei Audio3.mp3 in das WAV-Format mit Hilfe von *mpg123*.

Beispiel: mpg123 -w Audio3.wav Audio3.mp3

Aufgrund der Größe von Videodaten, erfolgt – wie in Kapitel 2.5.6 bereits erwähnt – keine Konvertierung. Eine Erweiterung wäre aber nach demselben Schema möglich. Außerdem wird es auch für das zusätzliche Format von sonstigen Dokumenten keine Konvertierung geben.

Der Status der Konvertierung bei den Teildokumenten wird jetzt in der *structure*-Datei gespeichert, um die Datei mit den Metadaten für Workflow und Administration nicht unnötig zu vergrößern. Außerdem würden bei der bisherigen Vorgehensweise redundante Informationen in beiden Dateien stehen, da die einzelnen Konvertierungsmerkmale auch für die Teildokumente gespeichert werden müssen. Dadurch würde indirekt in der *state*-Datei noch einmal die Dokumentstruktur abgelegt.
Der Status zeigt an, ob die Konvertierung in das jeweilige Format bereits versucht wurde oder nicht. Ob der Versuch erfolgreich war, ist nicht ausschlaggebend, da ein weiterer missglückter Versuch höchstwahrscheinlich das gleiche negative Ergebnis liefern würde.

```
<structure doc_id="2002-99">
     <content>
          <chapter rank="1" author="..." ...
               PDF="original" PS="yes" graphics="yes"
               color="yes" reverse="no">
          <image rank="1" author="..." ...
               PNG="original" JPG="yes">
          <image rank="2" author="..." ...
               PNG="original" JPG="yes">
          <audio rank="1" author="..." ...
               WAV="original" MP3="yes">
          <video rank="1" author="..." ...
               AVI="original">
     </content>
</structure>
```

## *5.6 Benutzerschnittstelle*

Aufgrund der umfangreichen Änderungen am Konzept des Dokumentenservers ist eine Umgestaltung der Benutzeroberfläche notwendig. Als Schnittstelle zu den Anwendern kommen weiterhin WWW-Browser zum Einsatz. Zum Anzeigen eines Dokuments ist auch künftig die Dokument-ID notwendig. Bei der Präsentation werden dann die einzelnen Teildokumente separat dargestellt. Sie sind wie gewohnt in mehreren Formaten abrufbar und, wenn möglich, getrennt kommentiert.
Die Suche wird so erweitert, dass man gezielt nach multimedialen Inhalten suchen kann. Sie ist auch unter Berücksichtigung von Stichwörtern möglich (siehe Kapitel 5.3.).

# 6 Systemerweiterungen im Detail

Da die vollständige Beschreibung aller Details der Implementierung sehr umfangreich wäre, werden nachfolgend die Klassen der zugehörigen Komponenten des Servlets beschrieben sowie die geänderten Methoden erläutert. Das vorhergehende Kapitel gab einen Überblick über die Erweiterungen im Allgemeinen. Im folgenden Kapitel werden die Änderungen an der Hauptkomponente des Servlets beschrieben. Danach werden die Modifikationen in den anderen Komponenten erläutert, die für die Erweiterungen bedeutsam sind. Dabei wurde das bisherige Konzept weitgehend beibehalten und lediglich an den relevanten Stellen erweitert.

## *6.1 Das Servlet*

Da der Dokumentenserver in einer Java-Umgebung läuft, wird die Benutzerschnittstelle zum Dokumentenserver über ein Java-Servlet realisiert. Der Server kann somit über jeden beliebigen Java-fähigen Web-Browser verwaltet und genutzt werden. Das Hauptprogramm des Servlets ist die *Servlet*–Komponente. Die Aufgaben der *Servlet*-Komponente sind die Initialisierung und das Beenden des Servlets sowie die Anfrage-Abarbeitung. Dabei werden die Anfragen von einem Web-Browser durch den Web-Server an das Servlet übertragen. In der *Servlet*-Komponente werden diese Anforderungen an die einzelnen Teilkomponenten weiterleitet (*processRequest*). Eine Auswahl der Methoden in der *Servlet*-Komponente ist in Abbildung 6.1 dargestellt.

Aufgrund der Erweiterungen ergeben sich dabei Änderungen in den folgenden Methoden:

- *processRequest()*
- *submitForm()*
- *submit()*
- *showDoc()*
- *showFulltext()*
- Administation (siehe Kapitel 6.6)
- Export von Metadaten (siehe Kapitel 6.7)

Neue Methode:

- *doSearchMM()*, Suche nach Multimedia-Daten

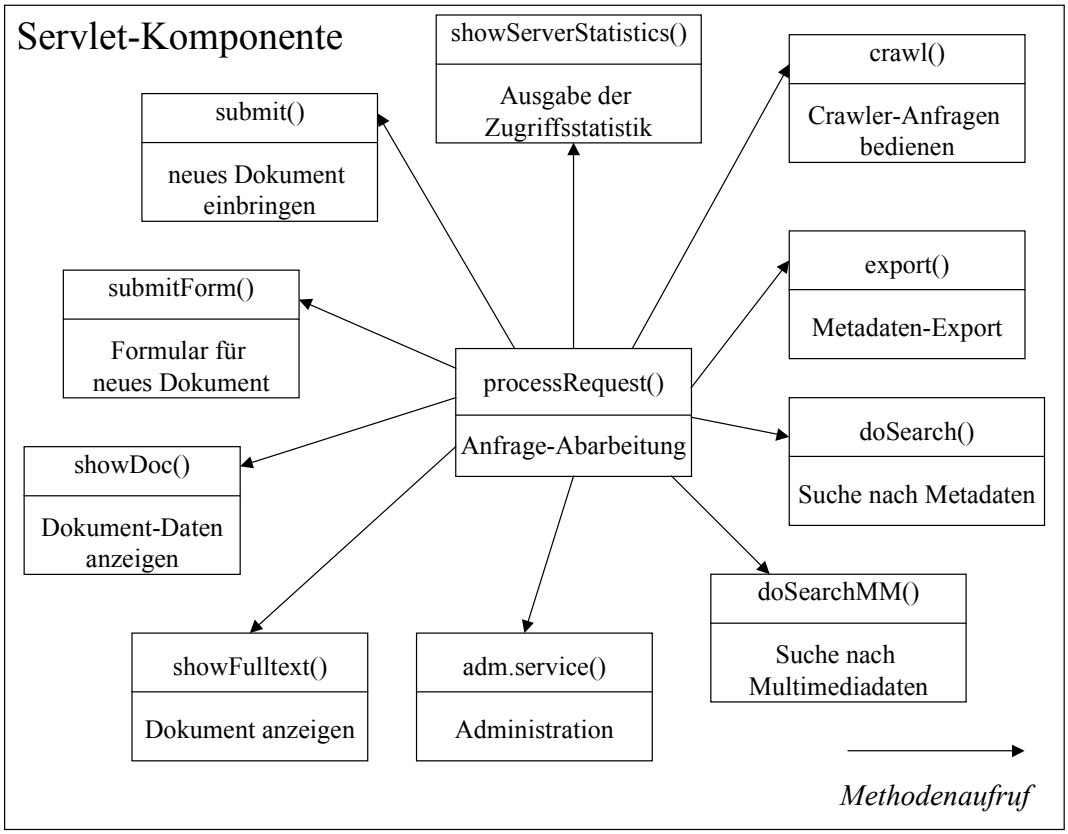


**Abbildung 6.1: Die Servlet-Komponente**

## Die Methoden im Detail

```
void processRequest(HttpServletRequest req,
     HttpServletResponse res)
```

Alle Anfragen an das Servlet verarbeitet diese Methode. Die Parameter der Anfrage werden als Datentyp *HttpServletRequest* übergeben, die Daten für die Antwort als Dateityp *HttpServletResponse*. Dabei werden anhand der Parameter die zugehörigen Methoden aufgerufen. Für das Einbringen eines komplexen Dokuments werden dadurch zusätzliche Parameter ausgewertet, die die Anzahl der Kapitel beschreiben. Weiterhin wird hier die neue Methode für die Suche nach Multimedia-Dokumenten aufgerufen.

```
void submitForm(HttpServletRequest req,
     HttpServletResponse res, int lang, int chapter,
     int images, int audio, int video)
```

Diese Methode erstellt ein Formular für die Einreichung eines neuen Dokumentes. Die Parameter für Anfrage und Antwort werden auch hier zur weiteren Verarbeitung übergeben. Zusätzlich wird ein interner Wert für die Sprache übermittelt. Neu ist dabei

die gewünschte Anzahl der zu übertragenden Kapitel, Bilder, Audio- und Videodokumente.

```
void submit(HttpServletRequest req,
     HttpServletResponse res, MultipartData mfdata,
     int lang)
```

Die zur Einreichung eines Dokumentes notwendigen Daten überträgt diese Methode zum Dokumentenserver. Die Daten befinden sich in dem Parameter vom Typ *MultipartData*. Wie der Name bereits sagt, enthält dieser Wert zahlreiche Daten, die von verschiedenem Typ sein können. Diese beinhalten zum Beispiel die übertragenen bibliographischen Daten und die eigentlichen Dokumente. Der Medientyp eines Eintrages wird durch den *Content-Typ* (siehe Kapitel 2.2.) festgelegt. Auch hier werden die Parameter für Anfrage, Antwort und Sprache wieder übermittelt. Neu ist hier die Unterstützung für Dokumente mit mehreren Kapiteln, sowie deren Speicherung und die Erstellung der Strukturdaten.

```
void showDoc(BibDoc doc, int lang,
     HttpServletRequest req, HttpServletResponse res)
```

Die HTML-Seite mit den bibliographischen Daten und den Links zu den eigentlichen Dokumenten erstellt diese Methode. Dabei werden die bibliographischen Daten für ein Dokument als Parameter vom Typ *BibDoc*, ein interner Wert für die Sprache, sowie die Parameter für die Anfrage und Antwort übertragen. Bei der Generierung der Links müssen jetzt alle Kapitel eines Gesamtdokumentes betrachtet werden. Zusätzlich werden Größe, Autor, Titel und Kommentar zu jedem Kapitel angezeigt.

```
void showFulltext(BibDoc doc, int lang,
     HttpServletRequest req, HttpServletResponse res)
```

Diese Methode liefert als Antwort den Inhalt eines Teildokuments oder alle Teildokumente in einer komprimierten Zip-Datei als Datenstrom. Um welches Gesamtdokument es sich handelt, legt der Parameter *doc* fest. Kapitel, Komprimierungsart und Datenformat sind unter anderem im Parameter *req* verzeichnet. Die Unterstützung für neue Datenformate wurde zusätzlich implementiert.

```
void doSearchMM(HttpServletRequest req,
     HttpServletResponse res, int lang)
```

Diese neue Methode führt mit Hilfe der Datenbank eine Suche nach Dokumenten mit mehreren Kapiteln sowie Bildern, Ton- und Videodaten durch und gibt die Ergebnisse in Tabellenform aus. Der Parameter *req* beinhaltet dabei unter anderen den oder die zu suchenden Medientypen und eventuelle Stichwörter.

## 6.2 Die Indexing–Komponente

Diese Komponente ist für den Abgleich der administrativen und Metadaten mit der Datenbank zuständig. Eine komplette Indizierung des Datenbestandes (*Rebuild*) wird jeweils nach einem Neustart des Dokumentenservers ausgeführt. Ändert sich ein Workflow-Zustand oder wird ein neues Dokument eingebracht, wird nur das betroffene Dokument neu indiziert (*Reindex*). Eine weitere Aufgabe der Komponente ist die Überprüfung, ob für jedes Kapitel die möglichen Dateiformate vorliegen, und startet gegebenenfalls die Konvertierung. Die Methoden im Einzelnen sind in Abbildung 6.2 dargestellt.

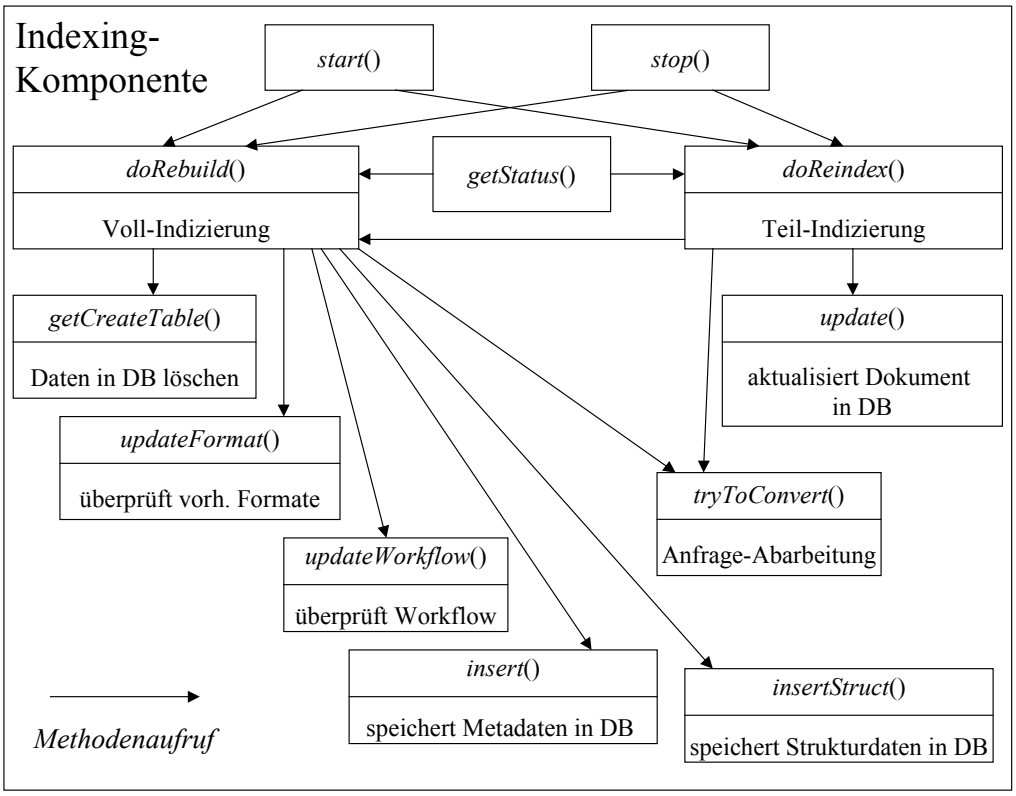


**Abbildung 6.2: Die Indexing – Komponente**

Die von Änderungen betroffenen beziehungsweise neuen Methoden sind:

- *doRebuild()*
- *updateFormat()*
- *tryToConvert()*
- *getCreateTable()*

Neue Methode:

- *insertStruct()*

## Die Methoden im Detail

```
public synchronized void doRebuild()
```

Diese Methode führt eine komplette Neuindizierung des Datenbestandes durch. Dabei werden die bibliographischen und strukturellen Daten aller Dokumente aus dem *Data Store* gelesen. Zusätzlich werden Workflow-Zustände aktualisiert und ausstehende Konvertierungen gestartet. Aufgrund dieser Informationen wird die Datenbank neu generiert. Die Methode wird nur bei einem Neustart des Dokumentenservers aufgerufen. Die Einbeziehung der Strukturdaten der Dokumente bei der Neuindizierung wurde zusätzlich implementiert.

```
boolean updateFormat(BibDoc doc)
```

In dieser Methode wird überprüft, ob das Speicherungsformat des Dokumentes auf dem aktuellsten Niveau ist. Sie liefert *TRUE* zurück, wenn eine Änderung beim Workflow oder bei den Konvertierungen stattgefunden hat. Um welches Gesamtdokument es sich handelt, legt der Parameter *doc* fest. Die Funktionalität wurde auf die neuen Teildokumente erweitert.

```
boolean tryToConvert(BibDoc doc)
```

Aufgabe dieser Methode ist es, alle Teildokumente zu finden, bei denen noch keine Konvertierung versucht wurde. Trifft die Bedingung zu, wird *TRUE* zurückgeliefert und eventuelle Konvertierungen werden gestartet. Dabei wird das Gesamtdokument über den Parameter *doc* referenziert. Auch hier wurde die Funktionalität auf die neuen Datenformate erweitert.

```
public String getCreateTable(String table)
```

Diese Methode erstellt SQL-Anweisungen, um in der Datenbank neue Tabellen für die bibliographischen und strukturellen Daten sowie Daten für Protokollierungs- und Suchfunktionen anzulegen. Der Parameter *table* enthält dabei eine Konstante, um welche Tabelle es sich dabei handelt. Der Rückgabewert enthält dabei die vollständige SQL-Anweisung. Die Tabelle für die Strukturdaten ist neu hinzugekommen.

```
void insertStruct(BibDoc doc, Statement stmt,
     boolean tryUpdate)
```

Diese neue Methode fügt einen Datensatz über die Strukturdaten eines Dokumentes in die Datenbank ein. Das betreffende Dokument wird über den Parameter *doc* referenziert, *stmt* ist ein Platzhalter für die spätere SQL-Anweisung. Der Parameter *tryUpdate* legt fest, ob ein neuer Datensatz angelegt oder ein bestehender aktualisiert wird.

## 6.3 Komponente für Document, Metadata und Workflow

Diese Komponente bildet eine Abstraktionsklasse für eine Dokumenteninstanz. Sie besteht aus 3 Teilkomponenten:

- bibliographische und strukturelle Attribute
- Kontroll-Informationen (Workflow-Zustände)
- Volltext- und Multimediadaten

Die Implementierung bietet eine logische Sicht auf eine Dokumenteninstanz, unabhängig von der jeweiligen Speicherungsart. Die speziellen Aufgaben sind Laden, Manipulation, Verifizierung und Speicherung der Metadaten sowie das Laden und Speichern der Quellen.
Einen Auszug der Methoden dieser Komponente zeigt Abbildung 6.3. Aufgrund der Nähe zu den eigentlichen Quelldokumenten ergeben sich umfangreiche Änderungen in folgenden Methoden:

- *getConverted()*
- *isOriginal<FORMAT>(), (für alle Formate)*
- *isConverted<FORMAT> (), (für alle Formate)*
- *setFulltext()*
- *hasFulltext()*
- *getFulltext()*
- *getGZIPedFulltext()*
- *getZippedFulltext()*

Zusätzlich sind auch einige Methoden neu hinzugekommen:

- *getPublicMMURL()*
- *initStruct()*
- *fetchStruct()*
- *setStructElement()*
- *editStructElement()*
- *getStructure()*
- *getContent()*
- *structToXML()*

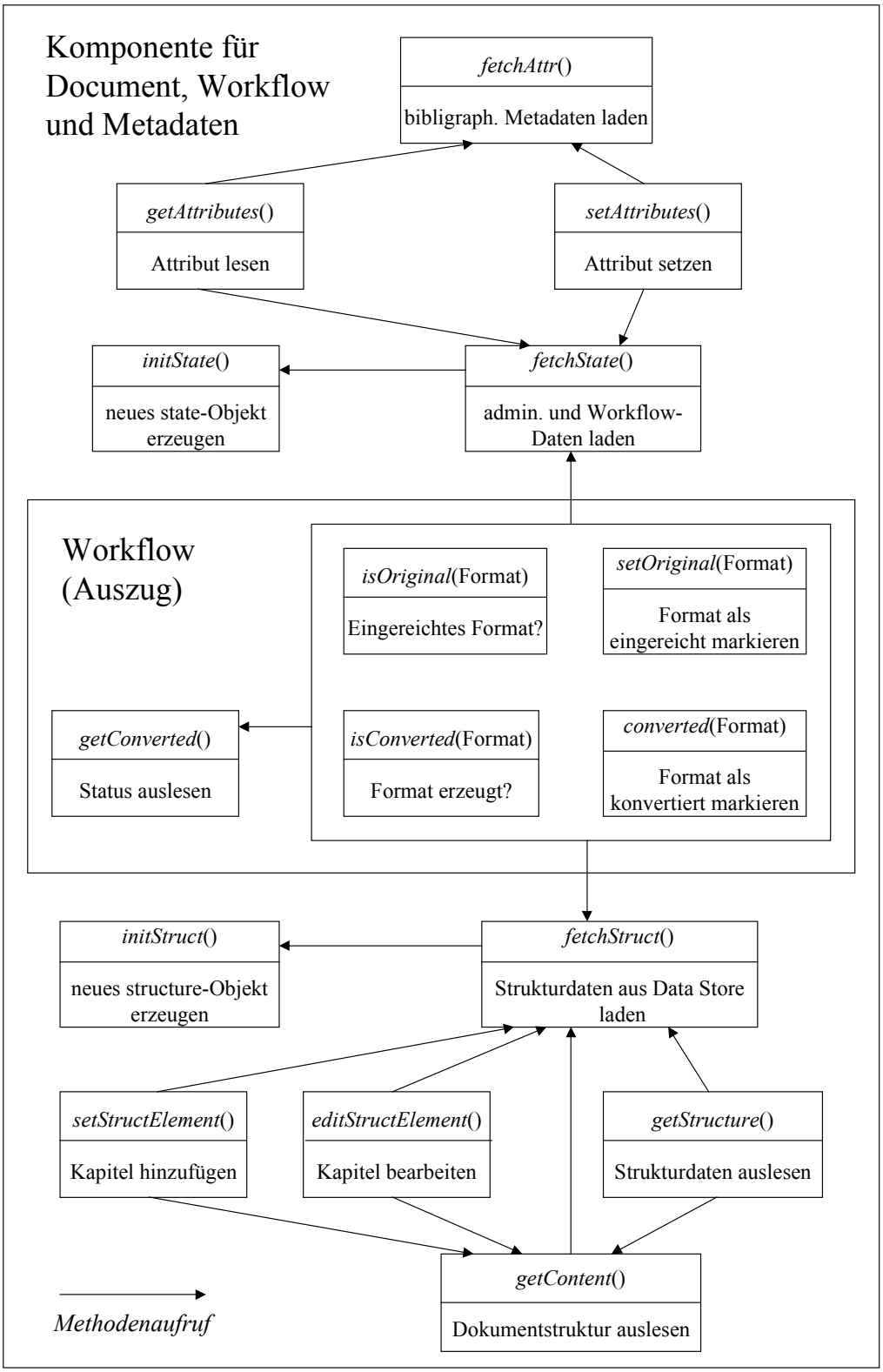


**Abbildung 6.3: Die Komponente für Document, Workflow und Metadaten**

## Die Methoden im Detail

```
Element getConverted(Name type, int index)
```

Diese Methode lieferte ursprünglich das <converted>-Element aus der Workflow-Datei eines Dokumentes im XML-Format vom Datentyp *Element*. Durch die zusätzliche Strukturierung der Dokumente müssen aber auch Medientyp (*type* – Kapitel, Bild, Ton oder Video) und Kapitelnummer (*index*) extra angegeben werden. Diese Konvertierungsinformationen stehen außerdem zukünftig in der *structure*-Datei.

```
public boolean isOriginal<FORMAT>(int index)
```

Wenn das Teildokument mit der Kapitelnummer *index* im Format <*FORMAT*> eingereicht wurde, liefert diese Methode *TRUE*. Für *index=0* werden die Dokumente im alten Format (keine Strukturdaten) betrachtet. Diese Funktionalität wurde auch auf die neuen Formate erweitert.

```
public boolean isConverted<FORMAT>(int index)
```

Diese Methode liefert *TRUE*, wenn das Format <*FORMAT*> für das Teildokument mit der Kapitelnummer *index* erst durch Konvertierung erzeugt wurde. Für *index=0* werden die Dokumente im alten Format (keine Strukturdaten) betrachtet. Diese Funktionalität wurde ebenso auf die neuen Formate erweitert.

```
public void setFulltext(InputStream in, String type,
     int count)
```

Die bei der Einreichung temporär gespeicherten Dokumente werden in dieser Methode in den *Data Store* eingebracht. Der Parameter *in* enthält dabei einen Verweis auf eine dieser temporären Dateien mit Medientyp *type*, die als Teildokument mit der Kapitelnummer *count* abgespeichert wird. Die Abspeicherung erfolgt im Kompressionsformat *GNU-ZIP*. Die Funktionalität wurde auf die neuen Formate erweitert.

```
public long hasFulltext(String chapter, String type)
```

Diese Methode lieferte ursprünglich *TRUE*, wenn ein Dokument im Format *type* vorliegt. Sie wurde erweitert, um zusätzlich die Kapitelnummer des Teildokumentes als Parameter *chapter* anzugeben, und außerdem die Dateigröße zurückzuliefern.

```
public void getFulltext(OutputStream out, String chapter,
     String type)
```

Um den eigentlichen Inhalt eines Dokumentes anzuzeigen, kopiert diese Methode den

unkomprimierten Inhalt eines Teildokumentes vom Medientyp *type* mit der Kapitelnummer *chapter* in einen Datenstrom vom Datentyp *OutputStream*. Die Funktionalität wurde auf die neuen Formate erweitert.

```
public void getGZIPedFulltext(OutputStream out,
     String chapter, String type)
```

Diese Methode kopiert den GNU-ZIP–komprimierten Inhalt eines Teildokumentes vom Medientyp *type* mit der Kapitelnummer *chapter* in einen Datenstrom vom Datentyp *OutputStream*. Die Funktionalität wurde ebenso auf die neuen Formate erweitert.

```
public void getZippedFulltext(OutputStream out,
     String chapter, String type)
```

Den ZIP-komprimierten Inhalt eines Teildokumentes vom Medientyp *type* mit der Kapitelnummer *chapter* kopiert diese Methode in einen Datenstrom vom Datentyp *OutputStream*. Die Funktionalität wurde zusätzlich auf die neuen Formate erweitert.

```
public static String getPublicMMURL(Environment env,
     Handle handle, int lang, String format, int rank)
```

Diese neue Methode liefert die URL eines Teildokumentes, wie sie in der Dublin Core-Komponente verwendet wird. Der Parameter *env* enthält dabei die Umgebungsvariablen des Dokumentenservers (zum Beispiel die URL) und *handle* enthält die Dokument-ID. *Lang*, *format* und *rank* definieren Anzeigesprache, Dateiformat und Kapitelnummer des Teildokumentes.

```
void initStruct()
```

Diese neue Methode erzeugt ein leeres Strukturobjekt im XML-Format für ein Dokument.

```
synchronized void fetchStruct()
```

Diese neue Methode liest die Strukturinformationen aus einer *structure*-Datei im XML-Format.

```
public void setStructElement(String type, int count,
       String author, String title, String comm,
       String format, String other)
```

Diese neue Methode generiert ein neues Strukturelement (Kapitel) im Strukturobjekt. Den Medientyp (Kapitel, Bild, Ton oder Video) legt *type* fest und *count* definiert die Kapitelnummer. *Author*, *title* und *comm* enthalten Autor, Titel und einen Kommentar zu

diesem Kapitel. Das Format, in dem das Teildokument eingebracht wurde, wird durch *format* festgelegt. Falls das Teildokument zu den sonstigen oder Video-Formaten gehört, enthält *other* den Dateityp.

```
public void editStructElement(Name type, int index,
     String author, String title, String comm,
     Name format, String formatOp, boolean resetConverted)
```

Diese neue Methode modifiziert ein Strukturelement im Strukturobjekt. Das zu ändernde Element legen *type* (Medientyp) und *index* (Kapitelnummer) fest. Ein nichtleerer Parameter *author*, *title* oder *comm* ändert den Autor, Titel oder Kommentar zu diesem Kapitel. Ein nichtleerer Parameter *format* legt das Dateiformat fest, auf das die Operation *formatOp* angewendet wird. Operation definiert dabei das Setzen der Attribute „Format original eingereicht“, „Konvertierung wurde durchgeführt“ und „Konvertierung wurde noch nicht durchgeführt“. Ist *resetConverted* auf TRUE gesetzt, werden alle Formate des gleichen Medientyps außer <*format*> als „Konvertierung wurde noch nicht durchgeführt“ markiert.

```
public Vector getStructure()
```

Diese neue Methode liefert einen Vektor zurück, der alle Strukturelemente (inklusive ihrer Attribute) eines Dokumentes enthält. Für Dokumente im alten Format wird *null* zurückgeliefert.

```
public Element getContent()
```

Diese neue Methode liest den Inhalt eines Strukturobjektes aus. Der Rückgabewert enthält die Strukturdaten im XML-Format, wie sie von *getStructure()* ausgewertet werden.

```
String structToXML()
```

Diese neue Methode formatiert die Strukturdaten eines Dokumentes für die Ausgabe im XML-Format.

## *6.4 Data Store–Komponente*

Die *Data Store* – Komponente bildet die Verbindung des Dokumentenservers zum Speichermedium. Hier laufen alle Operationen zum Lesen und Schreiben der Daten zusammen. Dadurch braucht man nur diese Komponente bei der Verwendung eines anderen Speichermediums auswechseln.
Die wichtigen Methoden in dieser Klasse zeigt die Abbildung 6.4.

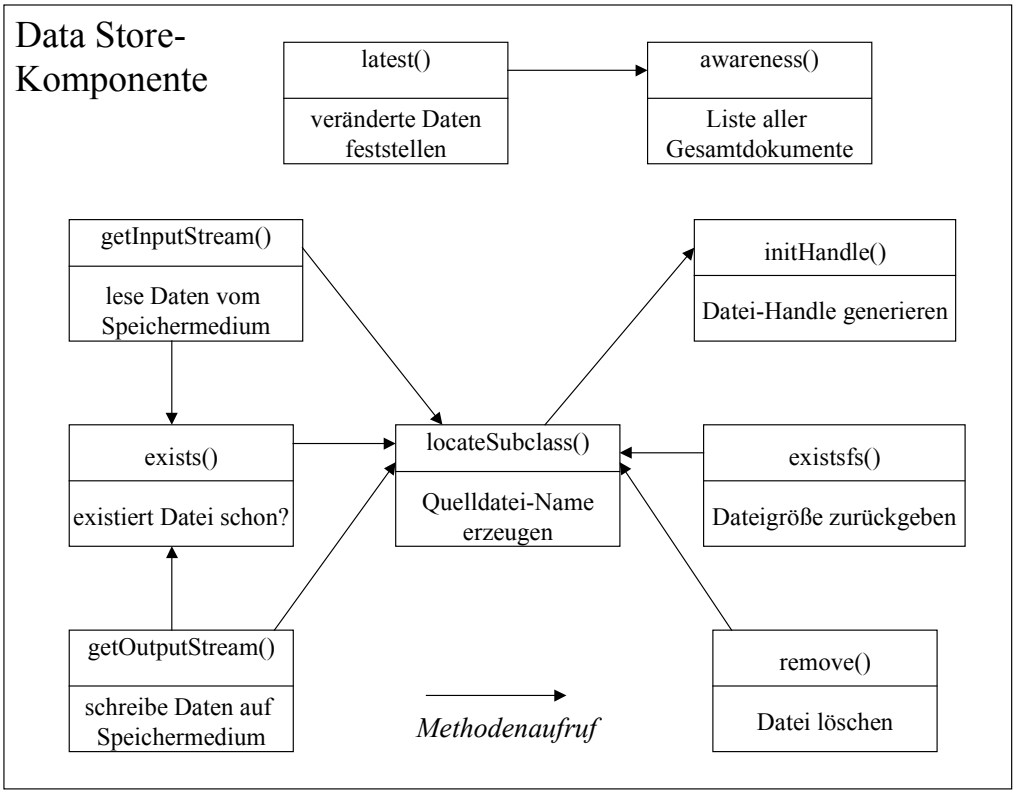


**Abbildung 6.4: Die Data Store – Komponente**

Aufgrund der modifizierten Speicherstruktur der Quellen ergeben sich auch hier umfangreiche Änderungen in folgenden Methoden:

- *getInputStream()*
- *getOutputStream()*
- *remove()*
- *exists()*
- *locateSubclass()*

Neue Methode:

- *existsfs()*

## Die Methoden im Detail

```
public InputStream getInputStream(Handle h, String chapter,
     String subclass)
```

Diese Methode erstellt einen lesenden Zugriff auf eine Datei aus dem *Data Store* und

liefert dabei einen Datenstrom vom Datentyp *InputStream* zurück. Der Parameter *h* enthält dabei die Dokument-ID, *chapter* die Kapitelnummer und *subclass* den Dateityp. Die Funktionalität wurde der erweiterten Archivierungsstruktur angepasst, wodurch die Kapitelnummer neu hinzugekommen ist.

```
public OutputStream getOutputStream(Handle h,
     String chapter, String subclass)
```

Diese Methode erstellt einen schreibenden Zugriff auf eine Datei im *Data Store* und liefert dabei einen Datenstrom vom Datentyp *OutputStream* zurück . Der Parameter *h* enthält dabei die Dokument-ID, *chapter* die Kapitelnummer und *subclass* den Dateityp. Die Funktionalität wurde der erweiterten Archivierungsstruktur angepasst, wodurch die Kapitelnummer neu hinzugekommen ist.

```
public boolean remove(Handle h, String chapter,
     String subclass)
```

Diese Methode löscht im *Data Store* eine Datei vom Format *subclass*, die zum Dokument mit der durch *h* festgelegten Dokument-ID gehört. Nach dem erfolgreichen Löschen wird *TRUE* zurückgeliefert. Der Parameter *chapter* legt die Kapitelnummer fest, falls es sich um ein Teildokument handelt. Dieser Parameter ist strukturbedingt neu hinzugekommen.

```
public boolean exists(Handle h, String chapter,
     String subclass)
```

Diese Methode überprüft, ob im *Data Store* eine Datei mit der Kapitelnummer *chapter* und dem Dateityp *subclass* existiert. Ist dies der Fall, wird *TRUE* zurückgegeben. Dabei wird das Dokument betrachtet, dessen Dokument-ID über den Parameter *h* definiert ist. Die Funktionalität wurde mit Hilfe des Parameters *chapter* der erweiterten Archivierungsstruktur angepasst.

```
public long existsfs(Handle h, [String chapter,]
     String subclass)
```

Diese neue Methode ist ähnlich der Methode exists(), nur dass sie die Größe der Datei im Data Store zurückliefert (*fs = filesize*). Existiert diese Datei nicht, ist der Rückgabewert „0“. Handelt es sich um ein Dokument im alten Format, wird der Parameter chapter weggelassen.

```
synchronized File locateSubclass(Handle handle,
     [String chapter,] String subclass)
```

Diese Methode liefert ein *File*-Objekt auf eine Datei zurück, die über die Parameter *handle*, *chapter* und *subclass* definiert wird. Für Dokumente im neuen Format ist der Parameter *chapter* notwendig.

## *6.5 Die Converter–Komponente*

Die *Converter*-Komponente ist für eine der Hauptaufgaben des Dokumentenservers zuständig: die Konvertierung der eingebrachten Dokumente in andere Formate. Hier werden die einzelnen Konvertierungsvorgänge gestartet, überwacht und gegebenenfalls beendet. Dauert eine Konvertierung länger als vorgegeben, wird die Konvertierung abgebrochen.
Abbildung 6.5 zeigt die wichtigen Methoden dieser Komponente, in der sich durch die Integration neuer Formate auch neue Konvertierungsvorgänge ergaben. Dadurch ist eine Vielzahl zusätzlicher Methoden hinzugekommen:

- *convertJPG()*
- *convertPNG()*
- *convertWAV()*
- *convertMP3()*
- *doConvertJPG()*
- *doConvertPNG()*
- *doConvertWAV()*
- *doConvertMP3()*

Modifizierte Methoden:

- *get()*
- *Converter()*
- *run()*
- *toPostscript()*
- *distill()*
- *convertASCII()*
- *convertGraphics()*
- *doPostscript()*
- *doDistill()*
- *doConvertASCII()*
- *doConvertGraphics()*

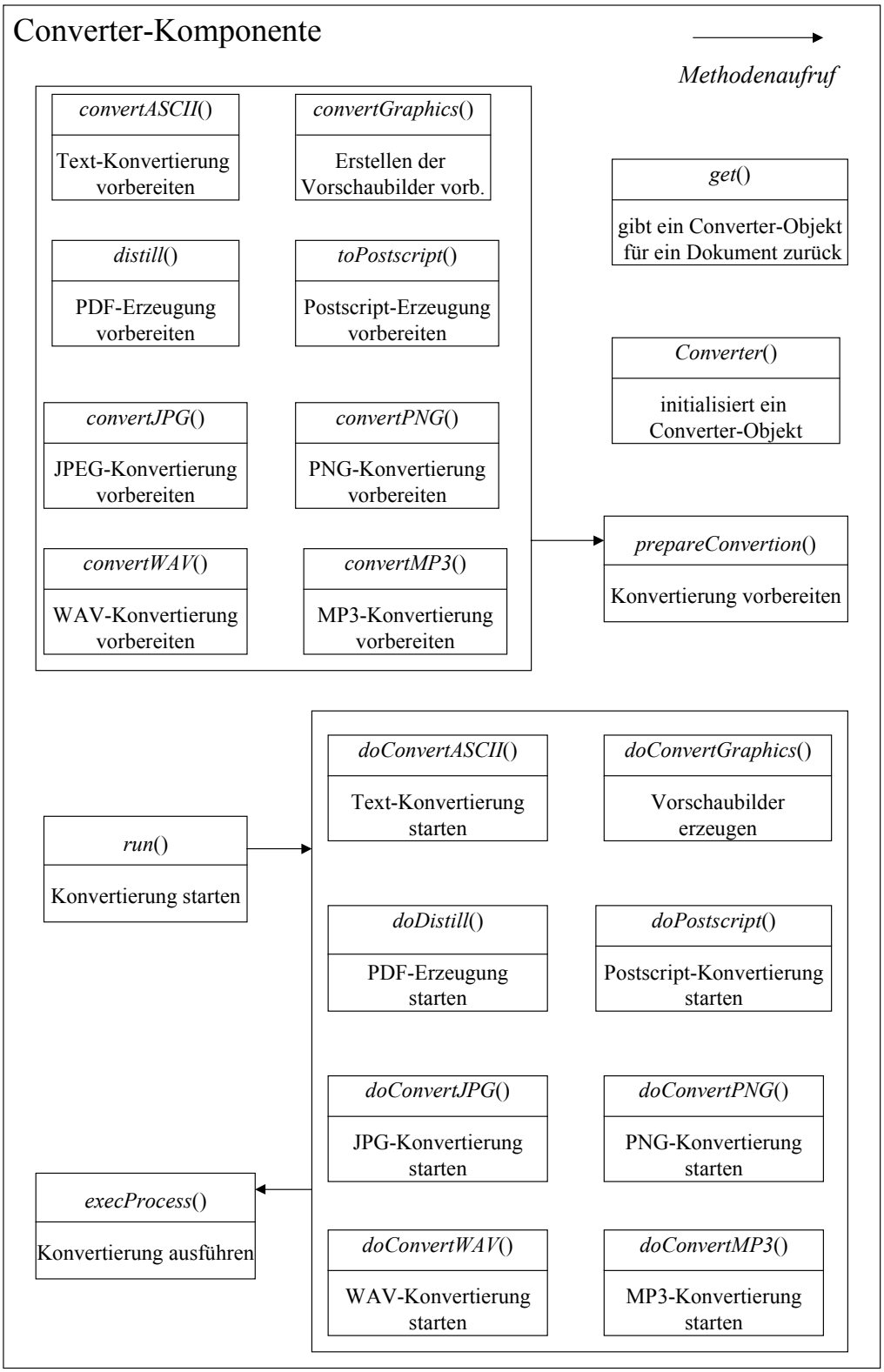


**Abbildung 6.5: Die Converter – Komponente**

## Die Methoden im Detail

```
Converter(Environment env, Handle handle, String name,
     String user, String rank)
```

Diese Methode erzeugt ein neues *Converter*-Objekt für ein Dokument. Der Parameter *env* enthält dabei wichtige Angaben zum *Data Store*, *handle* liefert die Dokument-ID, *name* enthält eine fortlaufende Nummer zur Identifizierung des Konvertierungsvorgangs und *user* bezeichnet den Benutzer, der die Konvertierung initialisiert hat. Über den neuen Parameter *rank* wird die Kapitelnummer des Teildokumentes festgelegt, das konvertiert werden soll.

```
public static Converter get(Environment env, Handle handle,
     String user , String rank)
```

Diese Methode liefert ein *Converter*-Objekt aus der Liste der bereits laufenden Konvertierungen. Das Objekt wird dabei über den Parameter *handle* angesprochen. Existiert zum Parameter *handle* kein solches Objekt, wird es neu erzeugt.

```
public void run()
```

Diese Methode startet eine Konvertierung. Die Funktionalität wurde für die neuen Formate erweitert.

```
public synchronized boolean toPostScript(String rank),
public synchronized boolean distill(String rank),
public synchronized boolean convertASCII(boolean reverse,
     int lang, String rank),
public synchronized boolean convertGraphics(
     boolean reverse, boolean color, String rank)
```

Diese Methoden bereiten eine Konvertierung in die jeweiligen Formate vor. Dabei wird aufgrund der erweiterten Struktur die Kapitelnummer für das zu konvertierende Kapitel über den Parameter *rank* angegeben.

```
public synchronized boolean convertJPG(String rank),
public synchronized boolean convertPNG(String rank),
public synchronized boolean convertWAV(String rank),
public synchronized boolean convertMP3(String rank)
```

Diese neuen Methoden bereiten eine Konvertierung in die jeweiligen Formate vor. Dabei wird jeweils die Kapitelnummer für das zu konvertierende Kapitel über den Parameter *rank* angegeben.

```
public void doDistill(String rank),
public void doPostScript(String rank),
void doConvertASCII(String rank),
void doConvertASCII(String rank)
```

Diese Methoden starten eine Konvertierung in die jeweiligen Formate mit Hilfe eines sogenannten externen *Wrappers*, dem alle notwendigen Parameter übergeben werden. Zusätzlich wird aufgrund der erweiterten Struktur die Kapitelnummer für das zu konvertierende Kapitel über den Parameter *rank* angegeben.

```
public void doConvertPNG(String rank) ,
public void doConvertJPG(String rank) ,
public void doConvertWAV(String rank) ,
public void doConvertMP3(String rank)
```

Diese neuen Methoden starten eine Konvertierung in die jeweiligen Formate mit Hilfe eines sogenannten externen *Wrappers*, dem alle notwendigen Parameter übergeben werden. Auch hier wird die Kapitelnummer über den Parameter *rank* angegeben.

## *6.6 Die Komponente für die Administration*

Diese Komponente beinhaltet alle Funktionen zur Administration des Dokumentenservers. Die Aufgaben sind die Bearbeitung von bibliographischen Daten, Aktualisierung von Dokumenten, das (Ent-)Sperren von Dokumenten für die Öffentlichkeit sowie Löschen und Wiederherstellen von Dokumenten, bei denen die Verfallsfrist für das Einreichen der Copyright-Erklärungen verstrichen ist. Weiterhin gibt es hier die Möglichkeit, die Konvertierung von Formaten manuell zu starten.
Abbildung 6.6 zeigt die wichtigsten Methoden dieser Komponente, die (ähnlich der *processRequest()*-Methode in der Servlet-Komponente) durch eine zentrale *service*-Methode aufgerufen werden.
Änderungen ergeben sich in dieser Komponente durch die Einbindung der neuen Multimediaformate, sowohl bei der nachträglichen Einreichung, als auch bei der Konvertierung:

- *userEdit()*
- *userSubmit()*
- *convertForm()*
- *convertDoc()*

Die nachträgliche Einreichung von Dokumenten beinhaltet jetzt nicht nur den Austausch eines fehlerhaften Dokumentes, sondern auch die Einbringung eines weiteren Teildokumentes. Durch die Verwendung von expliziten Strukturdaten ist deren Erweiterung ohne Probleme möglich.

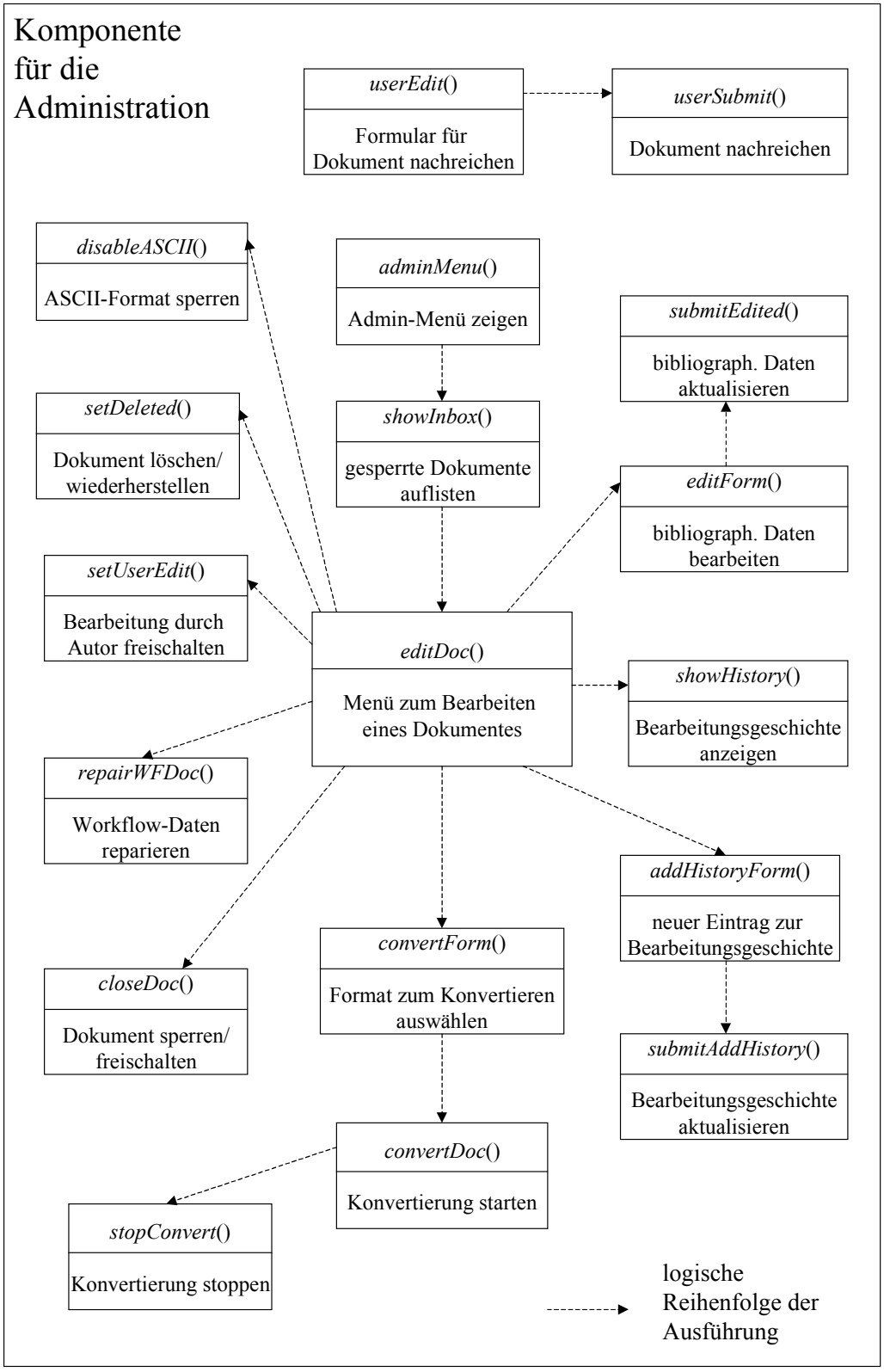


**Abbildung 6.6: Die Komponente für die Administration**

## Die Methoden im Detail

```
void userEdit(BibDoc doc, int lang, HttpServletRequest req,
     HttpServletResponse res)
```

Diese Methode erstellt ein Formular für die nachträgliche Einreichung eines oder mehrerer Teildokumente. Zusätzlich kann ein neues Kapitel eingebracht werden. Die Parameter *req* und *res* enthalten dabei die Webserver-Angaben für Anfrage und Antwort. Einen internen Wert für die verwendete Sprache des Formulars liefert *lang*, und *doc* enthält die Dokument-ID des Dokument. Die Änderungsmöglichkeiten wurden erweitert, so dass die Teildokumente einzeln nachgereicht oder hinzugefügt werden können.

```
void userSubmit(BibDoc doc, HttpServletRequest req,
     HttpServletResponse res, MultipartData mfdata,
     int lang)
```

Mit Hilfe dieser Methode werden die Daten aus dem Formular für das Nachreichen von Kapiteln zum Dokumentenserver übertragen. Das betreffende Dokument wird durch *doc* referenziert. Die Parameter der Anfrage und Antwort sind in *req* und *res* abgelegt. Die eigentlichen zu übertragenden Dokumentdaten befinden sich dabei in dem Parameter vom Typ *MultipartData*. Einen internen Wert für die Anzeigesprache der Übertragungsbestätigung beinhaltet *lang*. Die Funktionalität dieser Methode wurde auf die zusätzlichen Strukturinformationen und -daten erweitert.

```
void convertForm(BibDoc doc, int lang,
     HttpServletRequest req, HttpServletResponse res)
```

Um einzelne Teildokumente erneut manuell konvertieren zu können, erstellt diese Methode ein Formular. Das betreffende Dokument wird durch den Parameter *doc* gekennzeichnet. Die Anzeigesprache des Formulars wird durch den internen Parameter *lang* ausgewählt. Die Anfrage und Antwort-Daten des Webservers sind wieder in *req* und *res* abgespeichert. Die Funktionalität wurde auf die neuen Medienformate erweitert.

```
void convertDoc(BibDoc doc, int lang,
     HttpServletRequest req, HttpServletResponse res,
     int cmd)
```

Diese Methode startet die manuelle Konvertierung eines Teildokumentes. Der Parameter *doc* beinhaltet dabei einen Verweis auf das zugehörige Gesamtdokument. Die Ausgabesprache der Statusmeldungen legt der Parameter *lang* fest. Die Parameter für Anfrage und Antwort an den Webserver sind in *req* und *res* abgelegt. Ein interner Wert für das Zielformat der Konvertierung ist im Parameter *cmd* abgespeichert. Die Funktionalität wurde auf die neuen Medienformate erweitert.

## 6.7 Dublin Core–Komponente

Wenn die Hauptseite eines Dokuments aufgerufen wird, werden im HTML-Code die zugehörigen Metadaten im RDF-Format [17] übertragen. Dies hat keinen Einfluss auf den Inhalt und das Design der zu betrachtenden Seite, diese Informationen können nur bestimmte Dienste auswerten. Dabei werden im *Header* der HTML-Seite weiterführende Angaben zum Inhalt der Seite erstellt. Folgender Ausschnitt zeigt den HTML-Code für das Beispiel aus Kapitel 4.3.:

```
<html>
<head>
<title>Wetzer, Barbara; Pfandler, Alexander; Györvary, Erika; Pum,
Dietmar; Lösche, Mathias; Sleytr, Uwe B.: S-layer reconstitution at
phospholipid monolayers</title>
<!--
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.0/">
<rdf:Description about="http://dol.uni-leipzig.de/pub/1998-43/en"
   dc:Publisher="Leipzig University Publication Server"
   dc:Format="text/html"
   dc:Language="en"
   dc:Date="1998">
  <dc:Title>S-layer reconstitution at phospholipid monolayers
  </dc:Title>
<dc:Description>The recrystallization of the S-layer protein from
...
</dc:Description>
  <dc:Creator>
    <rdf:Seq
      rdf:_1="Wetzer, Barbara"
      rdf:_2="Pfandler, Alexander"
      rdf:_3="Györvary, Erika"
      rdf:_4="Pum, Dietmar"
      rdf:_5="Lösche, Mathias"
      rdf:_6="Sleytr, Uwe B."/>
  </dc:Creator>
  <dc:Type rdf:resource="http://dol.uni-leipzig.de/types/LEI-
   Preprint"/>
  <dc:Subject rdf:resource="http://dol.uni-leipzig.de/subjects/DNB-5-
   29"/>
  <dc:Subject rdf:resource="http://dol.uni-leipzig.de/subjects/DNB-5-
   30"/>
</rdf:Description>
</rdf:RDF>
-->
</head>
<body bgcolor='white'>
...
```

Die RDF-Syntax enthält dabei Angaben zur Schema-Version und den bibliographischen Metadaten des Dokuments, wie Titel, Autor, Organisation und Jahr der Einreichung. Die Informationen zu den einzelnen Teildokumenten werden nun zusätzlich übertragen. Dies erfolgt mit wenig Aufwand, da nur die Strukturdaten des Dokuments ausgewertet

werden. Folgendes Beispiel zeigt den RDF-Abschnitt für ein komplexes Dokument.

```
<!--<rdf:RDF
...
<rdf:Description about="http://lipsia.informatik.uni-
leipzig.de:8180/pub/2003-9/en"
   dc:Publisher="Leipzig University Publication Server"
   dc:Format="text/html"
   dc:Language="de"
   dc:Date="2003-03">
  <dc:Title>Test</dc:Title>
  <dc:Description>Diese Dokumente ... </dc:Description>
  <dc:Creator>
    <rdf:Seq
      rdf:_1="Melnik, Sergej"
      rdf:_2="Weise, Thomas"/>
  </dc:Creator>
  <dc:Type rdf:resource="http://dol.uni-leipzig.de/types/LEI-Other"/>
  <dc:Subject rdf:resource="http://dol.uni-leipzig.de/subjects/DNB-5-
28"/>
</rdf:Description>
<rdf:Description rdf:about="http://lipsia.informatik.uni-
leipzig.de:8180/pub/showDoc.Fulltext?lang=en&doc=2003-
9&format=ps&compression=&rank=1"
   dc:Format="text"
   dc:Description="Melnik, Sergej;DOL: An Interoperable Document
Server"
</rdf:Description>
<rdf:Description rdf:about="http://lipsia.informatik.uni-
leipzig.de:8180/pub/showDoc.Fulltext?lang=en&doc=2003-
9&format=png&compression=&rank=1"
   dc:Format="image"
   dc:Description="Bomberman world"
</rdf:Description>
<rdf:Description rdf:about="http://lipsia.informatik.uni-
leipzig.de:8180/pub/showDoc.Fulltext?lang=en&doc=2003-
9&format=mp3&compression=&rank=1"
   dc:Format="audio"
   dc:Description="Audio-Datei"
</rdf:Description>
<rdf:Description rdf:about="http://lipsia.informatik.uni-
leipzig.de:8180/pub/showDoc.Fulltext?lang=en&doc=2003-
9&format=ps&compression=&rank=2"
   dc:Format="text"
   dc:Description="Weise, Thomas; Personal Portal"
</rdf:Description>
<rdf:Description rdf:about="http://lipsia.informatik.uni-
leipzig.de:8180/pub/showDoc.Fulltext?lang=en&doc=2003-
9&format=png&compression=&rank=2"
   dc:Format="image"
   dc:Description="c't Lady"
</rdf:Description>
<rdf:Description rdf:about="http://lipsia.informatik.uni-
leipzig.de:8180/pub/showDoc.Fulltext?lang=en&doc=2003-
9&format=ps&compression=&rank=3"
   dc:Format="text"
   dc:Description="Weise, Thomas; Diplomarbeit"
</rdf:Description>
</rdf:RDF>-->
```

Der erste „*rdf:Description*“-Abschnitt liefert wie gewohnt Angaben über das Gesamtdokument. Alle weiteren Abschnitte enthalten Angaben zu den Teildokumenten, so zum Beispiel Format und URL.
Dabei ergibt sich die einzige Modifikation in der *Dublin Core*–Komponente in der Methode *toRDF()*. Diese Methode gibt die Informationen über ein Dokument im RDF-Format aus.

Modifizierte Methode:

- *toRDF()*

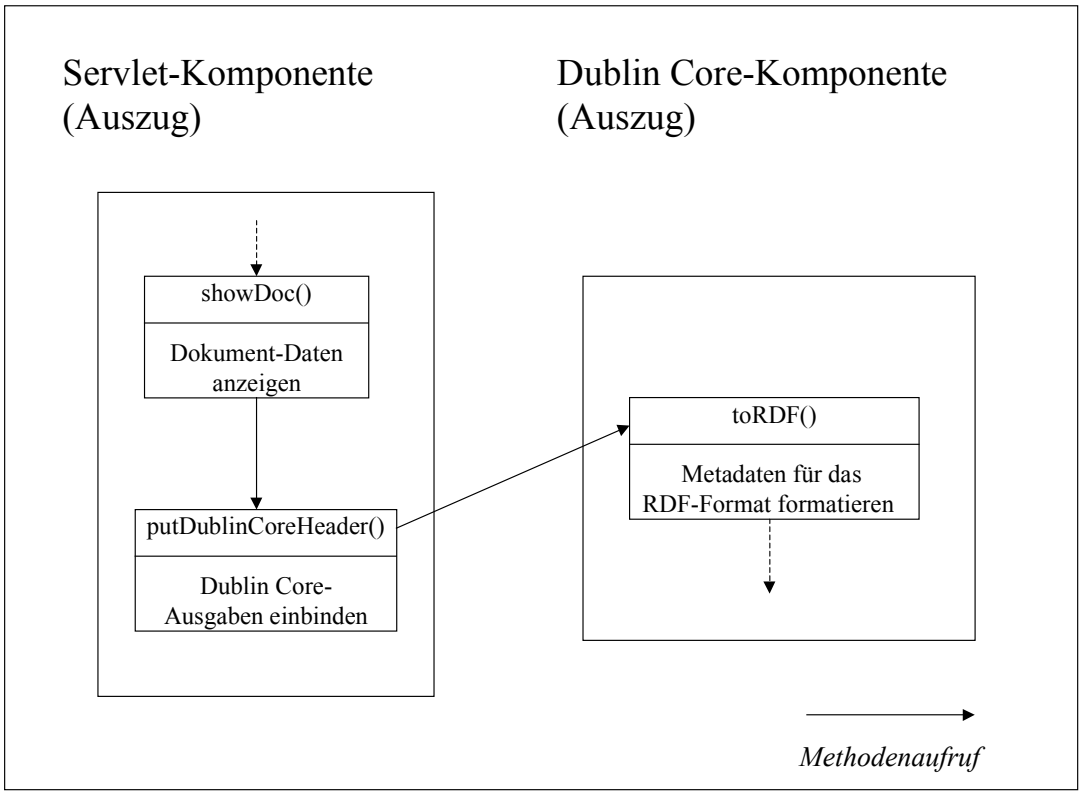


**Abbildung 6.7: Verwendung der Dublin Core-Komponente**

## Die Methode im Detail

```
public static String toRDF(Environment env, BibDoc doc)
```

Diese Methode liest die bibliographischen Daten eines Dokumentes aus der Datenbank und sie als Datentyp String im RDF-Format zurück. Das Dokument wird dabei über den Parameter *doc* referenziert, und der Parameter *env* enthält die Umgebungsvariablen des Dokumentenservers. Die Ausgabe der Daten wurde auf die Teildokumente erweitert.

# 7 Implementierung

Während die vorhergehenden Kapitel die inhaltlichen Änderungen am Dokumentenserver erläutern, beschreibt dieses Kapitel die Aspekte zur Implementierung. Dabei werden die Software-Voraussetzungen betrachtet, die Installation des Dokumentenservers, Kompilierung und Probleme bei der Installation sowie bei der Implementierung. Die Installation wurde weitgehend automatisiert, lediglich die Konfiguration des Dokumentenservers und der Datenbank muss weiterhin per Hand erfolgen.

## *7.1 Software*

Das bestehende System wurde unter dem UNIX-Betriebssystem *Solaris* Version 2.6 (SunOS 5.6) [22] entwickelt. Im Prinzip ist aber auch jede andere UNIX-Variante denkbar, da keine Solaris-spezifischen Programme und Funktionen verwendet werden. Die Portierung sollte ohne Probleme möglich sein.
Die Grundvoraussetzung für den Dokumentenserver stellt der Webserver dar. Hierbei wird der freie Server von *Apache* [13] in der Version 1.3.12 benutzt, der für fast alle Betriebssysteme verfügbar ist. Auch hier besteht die Möglichkeit, einen anderen Webserver einzusetzen.
Da die Web-Anbindung des Dokumentenservers als Servlet konzipiert ist, wird für diese Apache-Version ein zusätzliches Modul benötigt. Dabei handelt es sich um *Apache JServ* [23], Version 1.1, welches zusätzlich installiert werden muss.
Um die Funktionalität der Indizierung nutzen zu können, ist ein installiertes Datenbanksystem notwendig, welches auch für die Speicherung der Dokumente verwendet werden kann. Hierbei wird das freie System *MySQL* in der Version 3.21.33 verwendet. Es kann aber auch jedes andere relationale SQL-Datenbanksystem verwendet werden, welches eine JDBC-Schnittstelle besitzt. Über diese Schnittstelle kommuniziert der Dokumentenserver mit der Datenbank. Als JDBC-Treiber wird *mysql-connector-java-2.0.14* verwendet.
Für die Volltext-Indizierung zur Suche über den Dokumentenbestand wird das Programm *htDig* [12] in der Version 3.1.5 verwendet, das ebenfalls durch eine beliebige andere Suchmaschine ersetzt werden kann.
Die zur Konvertierung notwendigen Programme wurden bereits in Kapitel 5.5 erwähnt, daher hier noch einmal eine Zusammenfassung in Tabellenform:

| Quellformat | Zielformat | Programm |
|---|---|---|
| Postscript | PDF | Adobe Acrobat Distiller 3.02b |
| PDF | Postscript | Adobe Acroread 5.0 |
| Postscript | ASCII-Text | Pscript 2.81 |
| PNG | JPG | convert (Image Magick Toolkit) 5.5.1 |
| JPG | PNG | convert (Image Magick Toolkit) 5.5.1 |
| WAV | MP3 | LAME-Codec 3.51 (siehe Kapitel 2.4.11) |
| MP3 | WAV | mpg123 0.59r |

**Tabelle 7.1: Programme zur Konvertierung**

Diese Programme können ebenso ohne großen Aufwand ausgetauscht werden. Details dazu im folgenden Kapitel.
Die Implementierung des Dokumentenservers erfolgte in der Programmiersprache JAVA. Zur Kompilierung des Quellcodes wurde der JAVA-Compiler in der Version 1.4.0 verwendet.

## *7.2 Installation und Konfiguration*

In diesem Kapitel wird die Vorgehensweise bei einer Installation des Dokumentenservers mit den im vorigen Kapitel betrachteten Programmen erläutert. Dafür wird zunächst das Installationspaket benötigt, welches man sich unter der Internet-Adresse

**`http://dol.uni-leipzig.de/`**

herunterladen kann. Dieses entpackt man am besten in ein eigenes Verzeichnis, z.B. *$HOME/pubs* (*pub*lication *s*erver).
Danach hat man folgende Verzeichnisstruktur:

```
pubs/
pubs/app
pubs/doc
pubs/documents
pubs/source
```

Das Verzeichnis `pubs/app` beinhaltet alle zusätzlichen JAVA-Pakete wie XML-Unterstützung, HTTP-Erweiterungen und den JDBC-Treiber.
Die Dokumentation zu dem JAVA-Paket des Dokumentenservers befindet sich im Verzeichnis `pubs/doc` im JAVADOC-Format.
Unter `pubs/documents` werden die eigentlichen Dokumente des Dokumenten-servers archiviert. Nach der Installation ist in diesem Verzeichnis bereits ein Beispieldokument hinterlegt.

Das Verzeichnis `pubs/source` beinhaltet die Quellen und Binärdateien des Dokumentenservers, alle notwendigen Scriptdateien und die statischen Webinhalte. Die Scriptdateien und *Wrapper* (siehe Kapitel 5.5.) liegen in `pubs/source/scripts`. Hier können die zur Konvertierung verwendeten Scripte und Programme an die eigenen Wünsche angepasst werden.
Weiterhin liegen hier die eigentlichen Konfigurationsdateien für den Betrieb des Dokumentenserver, mit deren Hilfe aus sogenannten Schablonen („templates“) die eben genannten Komponenten generiert werden.
Im Verzeichnis `pubs/` liegen die Textdateien mit Informationen zur Installation (*INSTALL.TXT*), Konfiguration (*CONFIG.TXT*) und zur Fehlerbehebung („Frequently Asked Questions“ – *FAQ.TXT*). Weiterhin befindet sich hier die für die Installation notwendige Konfigurationsdatei (*INSTALL-CONFIG*) und das Installations-Script (*install.sh*). In der Datei *INSTALL-CONFIG* wird unter anderem das Ziel-Verzeichnis angegeben, in das der Dokumentenserver installiert wird. Unterscheidet es sich von dem Verzeichnis mit den entpackten Quellen, wird es neu angelegt und die notwendigen Quellen kopiert. Ebenso können die Verzeichnisse für die Dokumente und zusätzlichen Installationspakete angepasst werden.

Nach dem Entpacken kann die Konfiguration für die Installation (*INSTALL-CONFIG*) und den Dokumentenserver (*pubs/source/templates/config.xml*) erfolgen. Welche Angaben für die Installation benötigt werden, zeigt die Beispieldatei in Abbildung 7.2.
In der XML-Datei zur Konfiguration des Dokumentenserver stehen dagegen alle Informationen zum laufenden Betrieb. Dazu gehören administrative und organisatorische Angaben sowie mehrsprachige Textelemente für die dynamisch generierten HTML-Seiten. Diese Angaben können für jede beliebige Sprache erweitert werden.

```
#####################################################################
# necessary environment variables for install.sh                    #
# variables you have to edit are marked as  <---IMPORTANT--->       #
# others work with the default value for a normal installation      #
#                                                                   #
# Last changed: 2003-07-09, Thomas Weise                            #
#####################################################################

# the `home` of the Publication Server
PUBS_HOME=${HOME}/pubs
# directory for downloaded packages (web server, JServ, htDig, Pscript)
TEMP_ALL=${PUBS_HOME}/temp
# directory for extracted applications and jar's
APP_HOME=${PUBS_HOME}/app
# the home of the web server
HTTPD_DIR=${PUBS_HOME}/httpd
# JServ directory
JSERV_DIR=${HTTPD_DIR}/jserv
# directory for the search engine
HTDIG_DIR=${PUBS_HOME}/htdig
# Path to Pscript Tool
PSCRIPT_DIR=${PUBS_HOME}/pscript
# Path to documents
DOC_DIR=${PUBS_HOME}/documents
DOC_WWWDIR=/psdocs
# root-URL for the "public" access
PUB=/pub
# root-URL for static pages
AUX=/aux
# log files directory
LOG_DIR=${HTTPD_DIR}/logs
# Home directory for JAVA                                  <---IMPORTANT--->
JAVA_HOME=/dargb/j2sdk1.4.0_02
# location of the Java binaries
JAVA_DIR=$JAVA_HOME/bin
# the java servlet package
SERVLET_JAR=${APP_HOME}/servlet-2.0.jar
# jdbc driver
JDBC_DRIVER=${APP_HOME}/mysql-connector-java-2.0.14-bin.jar
# Path to Acrobat Distiller                                <---IMPORTANT--->
DISTILLER=/usr/loc_inf/Acrobat3/bin/distill
# Apache and JServ Port,                                   <---IMPORTANT--->
# need root rights for APACHE_PORT < 1024
APACHE_PORT=8080
SERVLET_PORT=8007
# Database server                                          <---IMPORTANT--->
SQL_SERVER_HOST=lipsia.informatik.uni-leipzig.de:1507
# Name of database, user name and password for database    <---IMPORTANT--->
SQL_DBNAME=dol2
SQL_DBUSER=dol
SQL_DBPWD=test
# Virtual server name                                      <---IMPORTANT--->
WEB_SERVER=lipsia.informatik.uni-leipzig.de:8080
# Real server name used for POSTing of forms               <---IMPORTANT--->
REAL_WEB_SERVER=lipsia.informatik.uni-leipzig.de:8080
# Name of the web server                                   <---IMPORTANT--->
WEB_SERVER_NAME_en="Leipzig University Publication Server"
WEB_SERVER_NAME_de="Dokumentenserver der Universit\&auml;t Leipzig"
# additional information for existing environment variables
#PATH=$HOME/bin:$PATH
#LD_LIBRARY_PATH=
#LD_RUN_PATH=
# PATH TO CONFIG-FILES, which will survive re-installation, leave blank for
# NOT using old configuration, $HOME/.pubs will be created as default
USE_OLD_CONF=
```

**Abbildung 7.2: Beispiel für die Konfigurationsdatei INSTALL-CONFIG**

Den Installationsprozess startet man mit dem Script *install.sh*, welches man wiederholt ausführen kann, sollten Probleme auftauchen. Bereits getätigte Schritte können dabei übersprungen werden.
Zuerst werden alle notwendigen Verzeichnisse angelegt und falls eine ältere Konfiguration vorhanden ist, diese auf Vollständigkeit überprüft. Dann wird der *Apache* Webserver mit den zugehörigen Komponenten installiert, konfiguriert und getestet. Die Suchmaschine und das *Pscript*-Tool werden als nächstes installiert. Nach diesem Schritt muss das vorhandene Datenbanksystem für den Zugriff des Dokumentenservers konfiguriert werden. Diese Einstellungen müssen aufgrund der unterschiedlichen Systeme manuell erfolgen.

Die Konfiguration des Dokumentenservers wird dann automatisch durch das Script *pubs/source/scripts/config.sh* durchgeführt. Dieses Script muss auch immer dann ausgeführt werden, wenn man Änderungen an der XML-Konfigurationsdatei des Dokumentenservers vornimmt. Die geänderten Einstellungen bleiben sonst ohne Wirkung, da die individuelle Konfigurationsdatei (*pubs/source/config-autogenerated.xml*) erst in diesem Schritt erzeugt wird. Diese darf dann nicht mehr verändert werden!

Danach werden noch zwei automatisierte Ausführungsanweisungen der *crontab* des Benutzers, unter dem der Dokumentenserver läuft, hinzugefügt. Die *crontab* ist eine Liste, die dem System sagt, wann es welche Programme automatisch ausführen soll. Die erste Anweisung führt einen täglichen Neustart des Dokumentenservers durch, um die Konsistenz zwischen dem Server und der Datenbank zu gewährleisten. Diese Anweisung in Form des Scripts *pubs/source/scripts/restart.sh* kann auch dazu genutzt werden, einen manuellen Neustart des Dokumentenservers durchzuführen.
Die zweite Anweisung erzwingt eine Neuindizierung der Volltext-Dokumente für die Suche.

Sollte es Probleme bei der Installation geben, zeigt die Datei *FAQ.TXT* eventuelle Fehlerquellen sowie deren Lösungsmöglichkeiten. Diese hier beschriebene Neuinstallation belegt etwa 61 Megabyte.

## *7.3 Kompilierung*

Die Klassen der JAVA-Quelltexte zum Dokumentenserver liegen im Verzeichnis *pubs\source\classes\edu\unile\dbs\pubs\main*. Die Komponenten sind als einzelne JAVA-Dateien getrennt. So können Erweiterungen implementiert werden, ohne die anderen Komponenten zu beeinflussen.

Um das gesamte Projekt zu kompilieren, muss man die JAVA-Datei *Library.java* übersetzen. Dabei ist zu beachten, dass alle notwendigen Verzeichnisse eingebunden werden sowie die PATH und CLASSPATH Umgebungsvariablen richtig gesetzt sind. Einen Anhaltspunkt gibt das bereits erwähnte Skript *restart.sh*. Am einfachsten ersetzt man für die Kompilierung in einer umbenannten Kopie dieses Skripts den Befehl *java* durch den Befehl *javac*. Das „restart.sh“-Skript muss auch nach einer erfolgreichen Neuübersetzung ausgeführt werden, damit die Änderungen wirksam werden.

## *7.4 Allgemeine Hinweise*

Natürlich traten während der Implementierung Fragen auf, an die man vorher nicht gedacht hatte. Zum Beispiel erwies es sich als Vorteil, den Pfad zur benutzten JAVA-Version und zum JDBC-Treiber als Konfigurationsvariable explizit anzugeben. So kann man eventuelle Probleme mit unterschiedlichen JAVA-Versionen oder JDBC-Treibern vermeiden.
Bei der Programmierung der Erweiterungen musste besonders auf die Kompatibilität zu älteren Installationen und deren Dokumentsammlungen geachtet werden. Dieser Umstand erforderte an manchen Stellen zusätzliche Überlegungen und zeitlichen Aufwand.
Durch die Größe der aufzunehmenden Multimediadateien trat ein weiteres Problem auf. So war bisher die Größe eines Dokumentes auf 50 Megabyte beschränkt. Diese Größe war fest einprogrammiert. Nunmehr kann sie in der XML-Konfiguration angepasst werden, 300 bis 700 Megabyte sollten ausreichen. Eine Änderung dieses Wertes sowie anderer Eigenschaften im laufenden Betrieb müssen mit Hilfe des Konfigurations-Skripts *pubs/source/scripts/config.sh* übernommen werden, gefolgt von einem Neustart des Dokumentenservers.
Da die Dokumente bei der Einreichung zunächst temporär gespeichert werden, ist es auch notwendig, das systemweite „Temp“-Verzeichnis großzügig einzurichten. Bei der Übertragung werden zunächst alle Dokumente in diesem Verzeichnis zwischengespeichert. Zusätzlich entstehen temporäre Daten durch den sofortigen Beginn der Konvertierungsvorgänge für diese Dokumente. Die Größe des temporären Verzeichnisses sollte aus Erfahrung mindestens das Vierfache der oben erwähnten maximalen Dokumentgröße betragen.

# 8 Test und Beispielanwendungen

## *8.1 Erweiterung eines bestehenden Systems*

Für den erweiterten Dokumenten-Server der Universität *Leipzig (Multimedia-Dokumenten-Server*) wurde ein neuer Benutzer-Account angelegt. Diese Vorgehensweise hatte zwei Hauptgründe: Verlagerung der Installation auf einen leistungsstärkeren Rechner sowie paralleler Betrieb der alten und neuen Version während des Migrationsprozesses.
Nach dem Herunterladen und Auspacken des Installationspaketes musste zunächst die Konfigurationsdatei *INSTALL-CONFIG* angepasst werden (siehe Kapitel 7.2). Danach wurde die Installation mit Hilfe des Skripts *install.sh* gestartet. Sie dauerte etwa zwei Stunden. Bei der Installation auf einem neuen System mit *Solaris* kann es vorkommen, dass beim Kompilieren von *htDig* C++-Bibliotheken nicht gefunden werden. Aus diesem Grund musste während der Installation die Pfadvariable *LD_RUN_PATH* entsprechend gesetzt sein. Das kann nachträglich in der Konfigurationsdatei *INSTALL-CONFIG* angepasst werden.
Nachdem die Installation erfolgreich beendet war, wurden die Dokumente vom alten System in das neue Dokumentenverzeichnis kopiert. Das dauerte ungefähr 120 Minuten für etwa 500 Publikationen (Stand Juli 2003). Alle Dokumente hatten einen Umfang von 2,5 Gigabyte, was durchschnittlich 5 Megabyte pro Publikation entspricht.
Danach musste der Dokumentenserver neu gestartet werden, um die Dokumente erstmalig zu indizieren. Zusätzlich sollte die Volltextindizierung einmalig von Hand gestartet werden (mit Hilfe des Skripts *.../pubs/source/scripts/dig-pubs* ), um sofort alle Such- und Navigationsfunktionen nutzen zu können.

## *8.2 Beispielanwendungen*

Aufgrund der Entwicklung in JAVA ist der Dokumentenserver auf zahlreichen Plattformen einsetzbar. Das System wurde so auch unter Red Hat Linux [24] erfolgreich getestet.
Sobald der Dokumentenserver seinen Betrieb aufgenommen hat, können neue Dokumente eingebracht werden. Ein mögliches Szenario wäre besonders im Umfeld einer Universität das Einbringen eines Vorlesungsskripts. Dabei können die einzelnen Kapitel getrennt und parallel zur Vorlesung eingereicht werden. Als Test wurde das Vorlesungsskript „Datenbanksysteme 1“ von Professor E. Rahm eingereicht. Es besteht aus sechs Postscript-Dateien mit einem Umfang von 9,3 Megabyte. Das Übertragen der Dateien dauerte dabei 25 Sekunden.

Ein weiteres Beispiel sind die schriftlichen Ausarbeitungen zu einem Seminar. Allgemein ist bei der Übertragung zu beachten, dass das Gesamtdokument möglichst vollständig ist. Wenn die Reihenfolge von Bedeutung ist, sollte ein fehlendes Teildokument durch einen Platzhalter präsentiert werden und später durch die eigentliche Komponente ersetzt werden. So wurden die Ausarbeitungen zum Seminar „Bio-Datenbanken“ im Wintersemester 2003/2003 unter der Leitung von Professor E. Rahm eingereicht. Es handelt sich hierbei um sieben Ausarbeitungen und drei Platzhalter mit einer Gesamtgröße von etwa sechs Megabyte. Bei den Platzhaltern wurde im Kommentar vermerkt, dass die Ausarbeitung (noch) nicht vorhanden ist. Die Übertragung dauerte etwa 15 Sekunden.

Die zu einer Tagung veröffentlichten Beiträge können nunmehr gesammelt als ein Gesamtdokument eingebracht werden. So ist der Zusammenhang durch die inhaltliche Zugehörigkeit zu einem Themengebiet gegeben. Zusätzlich können die Beiträge einzeln betrachtet und referenziert werden. Die Angabe von Autor und Überschrift des Beitrages sowie eines Kommentars ist ebenso möglich.
Als Beispiel dienten die Proceedings zur BTW 2003 in Leipzig mit einem Umfang von 44 PDF-Dateien und etwa zehn Megabyte Gesamtgröße. Die Übertragung der Dateien dauerte eine halbe Minute. Jedes Teildokument konnte mit Vortragendem und Beitragstitel spezifiziert werden

Ein weiterer Vorteil ergibt sich durch die Möglichkeit einer getrennten Archivierung von Text- und Bilddaten. Postscript und PDF-Dateien mit eingebetteten Grafiken können mitunter sehr groß werden. Zur Betrachtung des Dokumentes musste es vollständig heruntergeladen werden, inklusive der Bilddaten. Durch die Trennung der Daten werden die Volltext-Dokumente verkleinert und die direkte Betrachtung und Referenzierung der Bilddaten wird somit ermöglicht.

Für große Multimediadaten ohne Bezug zu einer Arbeit, ist die alleinige Archivierung vorgesehen. Lediglich Angaben zu Autor und Titel sowie ein möglicher Kommentar mit Informationen zum Dokument können hier notiert werden.
Als Beispiel wurden zwei Audiodateien und vier Videodateien mit einer Gesamtgröße von etwa 20 Megabyte eingebracht. Die Übertragung dauerte eine Minute und 30 Sekunden.

# 9 Zusammenfassung und Ausblick

In der vorliegenden Diplomarbeit wurde ein Konzept entwickelt, einen bestehenden textbasierten Dokumentenserver um multimediale und zusammengesetzte Dokumente zu erweitern. Der zweite Teil der Arbeit war die Realisierung dieser Erweiterungen. Die Kompatibilität zu bereits bestehenden Dokumentsammlungen sollte dabei gewährleistet werden.

Zu Beginn der Arbeit wurden die Formate ausgearbeitet, die für multimediale Erweiterungen in Frage kamen. Dabei waren drei Gruppen zu unterscheiden: Bilder, Tondokumente und Videodaten. Ziel war es, für jede Gruppe zwei Formate auszuwählen, um die Daten bezüglich Qualität und Datengröße ideal präsentieren zu können. Dieses Konzept erhöht außerdem die Wahrscheinlichkeit, das der Benutzer geeignete Anwendungen zum Anzeigen der Daten hat. Für Bilddaten wurden das PNG-Format (kein Informationsverlust) und das JPEG-Format (minimale Datengröße bei geringem Qualitätsverlust) ausgewählt. Tondokumente sollen im WAV-Format (kein Informationsverlust) und im MP3-Format (geringe Datengröße, kaum Qualitätsverlust, sehr weit verbreitetes Format) archiviert werden. Für Videodaten wird aufgrund der großen Datenmengen auf eine Konvertierung verzichtet. Dafür sind aber mehrere Formate für die Einreichung möglich: AVI, MPEG, WMV und Real Video.

Eine zusätzliche Erweiterung des Dokumentenservers ist die Archivierung von zusammengesetzten Dokumenten. Diese Dokumente können aus mehreren Teil-dokumenten bestehen, so zum Beispiel mehreren Kapiteln und/oder multimedialen Daten. Das Gesamtdokument ist somit unter einer eindeutigen Dokumenten-ID archiviert, während die Teildokumente getrennt abrufbar sind. Die Struktur dieses Dokumentes wird über Metadaten abgespeichert. Alle Teildokumente können auch zusammen heruntergeladen werden.

Kapitel 4 zeigt die Architektur des bestehenden Systems und seine Arbeitsweise. Dabei werden die einzelnen Komponenten und deren Schnittstellen beschrieben. Diese Komponenten sind: XML-basierte Konfiguration, Browsing und Navigation, Metadaten und Indizierung, Workflow und Formatumwandlung sowie Volltextsuche und Schnittstellen für den Export von Informationen zu externen Diensten.

Darauf aufbauend, werden anschließend im Kapitel 5 die Systemerweiterungen im Allgemeinen herausgearbeitet. Dabei ergeben sich Änderungen in der Referenzierung der Dokumente, wobei die zusätzliche Strukturinformationen in Metadaten abgespeichert werden müssen. Die Suche über die multimedialen Daten muss erweitert und die Schnittstellen zu anderen externen Diensten angepasst werden. Das Vorliegen von zusätzlichen Formaten stellt auch neue Anforderungen an die Konvertierung.

Die Bilddaten werden mit Hilfe des freien Tools *convert* aus dem *ImageMagick Toolkit* umgewandelt. Die Erzeugung der MP3-Daten aus WAV-Dateien erfolgt mit dem *LAME*-Codec, während für die andere Richtung das freie Tool *mpg123* verwendet wird. Da Videodaten nur in einem Format vorliegen sollen, ist hier keine Konvertierung notwendig. Die Autoren sollten dafür sorgen, dass die Daten in den gegebenen Formaten eingebracht werden.

Diese Änderungen haben natürlich auch Einfluss auf das äußere Erscheinungsbild des Dokumentenservers, der sich dem Benutzer leicht verändert präsentiert. Die Sicherung der Dokumente und Metadaten soll weiterhin durch das verwendete Speichermedium sichergestellt werden.

Im Kapitel 6 wird auf die Einzelheiten eingegangen, die im vorherigen Kapitel angesprochen wurden. Zuerst betrifft das die Änderungen im Detail in der *Servlet* – Komponente, durch die die zentrale Steuerung des Dokumentenservers erfolgt. Durch die Erweiterungen sind zusätzliche Anfragen abzuarbeiten. Daran anschließend wird die *Indexing* – Komponente betrachtet, die neue Metadaten mit der Datenbank abgleicht und zusätzliche Konvertierungsaufgaben verwaltet.
Die Komponente für *Document, Metadata und Workflow*, welche die Metadaten und Quellen verwaltet, wurde um Aufgaben bezüglich der zusätzlichen Strukturdaten und Dateiformate erweitert. Daraus ergaben sich auch Änderungen in der Data Store – Komponente, welche die strukturell modifizierten Daten auf das jeweilige Speichermedium schreibt und sie ausliest. Die Konverter - Komponente wurde um zusätzliche Methoden erweitert, um auch die neu hinzugekommenen Formate untereinander konvertieren zu können.
Für die manuelle Konvertierung und das Nachreichen von Dokumenten ergaben sich außerdem Änderungen in der Komponente zur Administration. Schließlich wurde der Export von Metadaten im RDF-Format erweitert, um die zusätzlichen Strukturdaten einzubinden.

Die Aspekte zur Implementierung werden in Kapitel 7 behandelt. Dabei wird die verwendete Software erläutert und die Installation und Konfiguration erklärt. Zusätzlich wird beschrieben, wie der Dokumentenserver an eigene Wünsche angepasst und das Projekt neu übersetzt werden kann. Dazu werden Fehlerquellen angesprochen, die während der Implementierung auftraten.

Kapitel 8 gibt schließlich einen kurzen Ausblick auf mögliche Testszenarien und Beispielanwendungen.

## *Ausblick*

Durch die vorgestellten Erweiterungen ergeben sich natürlich neue Ideen für Verbesserungen und Erweiterungen. So liegt es nahe, eine Auswahl aus beliebigen Teildokumenten in einem Archiv herunterladen zu können. Diese Teildokumente könnten dann auch aus verschiedenen Gesamtdokumenten kombinierbar sein. So wäre es denkbar, die Ergebnisse einer Suche gleich komplett zu übertragen.

Bei der Einbringung eines Dokumentes sollte man mehr Wert auf Sicherheit legen. Zum Beispiel könnten die Dokumente über eine sichere Verbindung übertragen werden (sicheres HTTP – HTTPS) und mittels einer Prüfsumme versehen werden, um die Authentizität der Dokumente sicherzustellen.

Eine weitere Möglichkeit wäre es, Tondokumente und Videodaten schon während des Downloads betrachten zu können („Streaming“). Die mitunter sehr großen Daten müssen dann nicht komplett heruntergeladen werden. Den Erweiterungen sind aufgrund des modularen Aufbaus des Dokumentenservers in dieser Hinsicht keine Grenzen gesetzt.

# Anhang

## *Literaturverzeichnis und Quellen*


[1] S. Melnik, E. Rahm, D. Sosna "DOL An Interoperable Document Server", http://dol.uni-leipzig.de/pub/2001-27/en , 2001

[2] E. Rahm, K. Schwipper, D. Sosna "Dienst für Online-Dokumente gestartet" http://www.informatik.uni-leipzig.de/ifi/medien/uj980615.html , 1998

[3] Pscript. http://www.ubka.uni-karlsruhe.de/~guenter/pscript , 2000

[4] Adobe. http://www.adobe.com/products/acrobat/main.html , Mai 2003

[5] Suchmaschine Google. http://www.google.de , Mai 2003

[6] Suchmaschine AltaVista. http://www.altavista.de , Mai 2003

[7] MILESS-Projekt der Universität Essen. http://miless.uni-essen.de , April 2003

[8] Digitale Bibliothek Thüringen. http://www.db-thueringen.de , April 2003

[9] Digitales Video- und Audioarchiv DIVA. http://www.ubka.uni-karlsruhe.de/diva/ , April 2003

[10] Mediaserver der Universität Heidelberg. http://www.uni-heidelberg.de/media , April 2003

[11] ddb, Archivserver DEPOSIT.DDB.DE . http://deposit.ddb.de , April 2003

[12] A. Crespo and H. Garcia-Molina. *Archival Storage for Digital Libraries*. Third ACM Conf. on Digital Libraries, Pittsburgh, PA, USA, 1998

[13] Apache Project. http://www.apache.org , Mai 2003

[14] htDig Homepage. http://www.htdig.org , April 2003

[15] NCSTRL Home Page. http://cs-tr.cs.cornell.edu , 2001

[16] R. Lasher and D. Cohen. *A Format for Bibliographic Records*, RFC 1807, Juni 1995

[17] DublinCore Home Page. http://dublincore.org/ , Mai 2001

[18] Dienst Overview and Introduction. htttp://www.cs.cornell.edu/cdlrg/dienst/DienstOverview.htm, April 2003

[19] Image Magick Toolkit. http://www.imagemagick.org , April 2003

[20] Fraunhofer Gesellschaft. http://www.iis.fraunhofer.de/amm/index.html , Mai 2003

[21] mpg123. http://www.mpg123.de/ , Mai 2003

[22] Sun Solaris. http://wwws.sun.com/software/solaris/index.html , Mai 2003

[23] Apache JServ. http://java.apache.org/jserv/ , November 2002

[24] Red Hat Linux. http://www.redhat.de , Mai 2003

[25] Renato Iannella. *Digital Rights Management (DRM) Architectures*, D-Lib Magazine, Juni 2001

## *Wichtige rechtliche Hinweise*

Mit Urteil vom 12. Mai 1998 - 312 O 85/98 - "Haftung für Links" - hat das Landgericht Hamburg entschieden, dass man durch die Anbringung eines Links, die Inhalte der gelinkten Seite gegebenenfalls mit zu verantworten hat. Dies kann nur dadurch verhindert werden, dass man sich ausdrücklich von diesen Inhalten distanziert:

Hiermit distanziere ich mich ausdrücklich von allen Inhalten aller verlinkten Seiten in dieser Arbeit und mache mir diese Inhalte nicht zu eigen. Diese Erklärung gilt für alle in dieser Arbeit veröffentlichen Links.

# Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, 16. Juli 2003 .....
(Thomas Weise)