MyCoRe User Guide

Frank Lützenkirchen
Jens Kupferschmidt
Detlev Degenhardt
Johannes Bühler
Ulrike Krönert
Ute Starke

Release 1.0

3. Februar 2005

Inhaltsverzeichnis

1.Voraussetzungen für eine MyCoRe Anwendung....8
1.1 Vorabbemerkungen....8
1.2 Allgemein benötigte Software....10
1.3 Hinweise zur Installation des IBM Content Manager 8.2....11
1.4 Hinweise zu vorbereitenden Installationen freier Datenbanken, Text-Suchmaschinen und Web–Komponenten....11
1.4.1 Installation der Komponenten unter Linux....11
1.4.1.1 Installation von Java....11
1.4.1.2 Die Installation von MySQL....11
1.4.1.3 Der JDOM-Search-Layer für das Beispiel....12
1.4.1.4 Die Nutzung der freien Text-Suchmaschine Lucene zur Volltextsuche....13
1.4.1.5 Die Installation von Apache 2....13
1.4.1.6 Die Installation von Tomcat 5....13
1.4.2 Installation der Komponenten unter MS Windows....14
1.4.2.1 Installation von Java....14
1.4.2.2 Installation von MySQL....14
1.4.2.3 Der JDOM-Search-Layer für das Beispiel....14
1.4.2.4 Die Nutzung der freien Text-Suchmaschine Lucene zur Volltextsuche....15
1.4.2.5 Installation von Apache 2....15
1.4.2.6 Installation von Tomcat 5....15
2.Download und Installation des MyCoRe Kerns....16
2.1 Download des MyCoRe Kerns....16
2.2 Konfiguration und Übersetzten des Kerns....17
3.Die MyCore Beispielanwendung MyCoReSample....20
3.1 Grundlegender Aufbau und Ziel der Beispielanwendung....20
3.1.1 Allgemeines....20
3.1.2 Weiterführende Erläuterungen....20
3.2 Download der Beispielanwendung....21
3.3 Konfiguration im Applikationsteil DocPortal....21
3.3.1 Allgemeines....21
3.3.2 Installation....21
3.4 Konfiguration im Applikationsteil MyCoReSample....22
3.4.1 Grundlegendes zur Konfiguration....22
3.4.1.1 CM8 Store konfigurieren....23
3.4.1.2 eXist Store Konfigurieren....23
3.4.1.3 JDBC-Treiber konfigurieren....23
3.4.1.4 Die SQL-Tabellennamen....23
3.4.1.5 User-Management....24
3.4.1.6 Remote-Access....24
3.4.1.7 Speicherung von Objekten und Dokumenten konfigurieren....24
3.4.1.7.1 File System Store....24
3.4.1.7.2 Lucene Store....24
3.4.1.8 Editor....25
3.4.1.9 Workflow....25
3.4.1.10 Appletsignatur....25

3.4.1.11 Debug konfigurieren....25
3.4.2 Laden der Beispiel-Inhalte....25
3.5 Arbeiten mit dem MyCoRe Command Line Interface....27
3.5.1 Erzeugen der Skripte mycore.sh / mycore.cmd....27
3.5.2 Aufruf des CommandLineInterface....27
3.6 Erzeugen und Konfigurieren der Web-Anwendung....27
3.6.1 Erzeugen der Web-Anwendung....27
3.6.2 Konfiguration des Web Application Server....28
3.6.2.1 Tomcat....28
3.6.2.2 Websphere....28
4.Vom Sample zum eigenen Dokumenten-Server....30
4.1 Allgemeines....30
4.2 Die Installation der freien XML:DB eXist....31
4.2.1 Allgemeines....31
4.2.2 Vorbedingungen....31
4.2.3 Installation....31
4.2.4 Konfigurationsänderungen in den MyCoRe-Teilen....31
4.2.5 Füllen des Search-Backends für eXist....32
4.3 Die Zusammenarbeit mit anderen MyCoReSample-Installationen....32
4.3.1 Die eigene Installation....33
4.3.2 Weitere Server benachbarter Einrichtungen....33
4.3.3 Standard-Server des MyCoRe-Projektes....33
4.4 Nutzung der OAI Schnittstelle....34
4.4.1 Grundlagen....34
4.4.2 Der OAI Data Provider....34
4.4.3 Installation....34
4.4.4 Der Deployment Descriptor....35
4.4.5 Die mycore.properties.oai....35
4.4.6 Instanzunabhängige Properties....35
4.4.7 Instanzabhängige Properties....36
4.4.8 Test....36
4.4.9 Zertifizierung und Registrierung....36
5.Weiterführende Informationen zum Aufbau von MyCoRe-Anwendungen....37
5.1 XML-Syntax des Datenmodells....37
5.1.1 Das Klassifikationen-Datenmodell....37
5.1.2 Das Metadatenmodell....38
5.1.2.1 XML-Syntax eines Metadatenobjektes....39
5.1.2.2 XML-Synatx des XML-Knotens structure....39
5.1.2.3 XML-Syntax des XML-Knotens metadata....40
5.1.2.4 MyCoRe Metadaten-Basistypen 41
5.1.2.4.1 XML-Syntax des Metadaten-Basistyps MCRMetaAddress....42
5.1.2.4.2 XML-Syntax des Metadaten-Basistyps MCRMetaBoolean....43
5.1.2.4.3 XML-Syntax des Metadaten-Basistyps MCRMetaClassification....43
5.1.2.4.4 XML-Syntax des Metadatentyps MCRMetaCorporation....44
5.1.2.4.5 XML-Syntax des Metadaten-Basistyps MCRMetaDate....45
5.1.2.4.6 XML-Syntax des Metadatentyps MCRMetaLangText....46
5.1.2.4.7 XML-Syntax der Metadatentypen MCRMetaLink und MCRMetaLinkID....46

5.1.2.4.8 XML-Syntax des Metadatentyps MCRMetaNumber....47
5.1.2.4.9 XML-Syntax des Metadatentyps MCRMetaPerson....48
5.1.2.4.10 XML-Syntax des Metadatentyps MCRMetaXML....49
5.1.2.5 XML-Syntax des XML-Knotens service....50
5.1.3 Das Speichermodell für die Multimediadaten (IFS)....50
5.2 Die Verwendung der Kommandozeilenschnittstelle der Benutzerverwaltung....52
5.2.1 Allgemeine Kommandos....52
5.2.2 Kommandos zum Arbeiten mit XML-Dateien....52
5.2.3 Kommandos zum direkten Arbeiten mit Objekten der Benutzerverwaltung....53
5.2.4 Das Sichern und Restaurieren der Benutzerverwaltungsdaten....54
6.Hints & Tips / Troubleshooting....56
6.1 Eine zweite Instanz von eXist auf einem System einrichten....56

Abbildungsverzeichnis

Abbildung 1: Einbindung des OAI-Data-Providers in web.xml....35
Abbildung 2: XML-Syntax eines Klassifikations-Objektes....37
Abbildung 3: XML-Syntax eines Metadaten-Objektes....39
Abbildung 4: XML-Syntax eines structure XML-Knotens....40
Abbildung 5: XML-Syntax eines metadata XML-Knotens....41
Abbildung 6: XML-Syntax des Metadaten-Basistyps MCRMetaAddress....42
Abbildung 7: Beispiel des Metadaten-Basistyps MCRMetaAddress....42
Abbildung 8: XML-Syntax des Metadaten-Basistyps MCRMetaBoolean....43
Abbildung 9: Beispiel des Metadaten-Basistyps MCRMetaBoolean....43
Abbildung 10: XML-Syntax des Metadaten-Basistyps MCRMetaClassification....44
Abbildung 11: Beispiel des Metadaten-Basistyps MCRMetaClassification....44
Abbildung 12: XML-Syntax des Metadaten-Basistyps MCRMetaCorporation....44
Abbildung 13: Beispiel des Metadaten-Basistyps MCRMetaCorporation....45
Abbildung 14: XML-Syntax des Metadaten-Basistyps MCRMetaDate....45
Abbildung 15: Beispiel des Metadaten-Basistyps MCRMetaDate....45
Abbildung 16: XML-Syntax des Metadaten-Basistyps MCRMetaLangText....46
Abbildung 17: Beispiel des Metadaten-Basistyps MCRMetaLangText....46
Abbildung 18: XML-Syntax des Metadaten-Basistyps MCRMetaLink....47
Abbildung 19: Beispiel des Metadaten-Basistyps MCRMetaLink....47
Abbildung 20: XML-Syntax des Metadaten-Basistyps MCRMetaNumber....47
Abbildung 21: Beispiel des Metadaten-Basistyps MCRMetaNumber....48
Abbildung 22: XML-Syntax des Metadaten-Basistyps MCRMetaPerson....48
Abbildung 23: Beispiel des Metadaten-Basistyps MCRMetaPerson....48
Abbildung 24: XML-Syntax des Metadaten-Basistyps MCRMetaXML....49
Abbildung 25: Beispiel des Metadaten-Basistyps MCRMetaXML....49
Abbildung 26: XML-Syntax des service XML-Konotens....50
Abbildung 27: XML-Syntax des Derivate-Datenmodells....51

Tabellenverzeichnis

Tabelle 1: MyCoRe Komponentenübersicht für IBM CM....8
Tabelle 2: MyCoRe Komponentenübersicht für freie Lösung....9
Tabelle 3: MyCoRe Komponentenübersicht für freie Lösung....10
Tabelle 4: Aufbau des MyCoRe-Kerns....17
Tabelle 5: Feste Test-Instanzen für das MyCoRe-Beispiel....33
Tabelle 6: MyCoRe-Datentypen....42

Vorwort

Dieses Dokument ist für einen Start in das MyCoRe-System konzipiert und soll dem Einsteiger den Weg zu einer ersten MyCoRe-Anwendung, einem Dokumenten-Server, aufzeigen. Begonnen wird mit der Beschreibung alle vorab notwendigen Installationen. Weiter geht es über die Installation des MyCoRe-Kerns und des mitgelieferten Beispiels. Ein weiteres Kapitel beschreibt die Gestaltung eines ersten Dokumenten-Servers. Abschließend sind noch einige Hinweise aus der MyCoRe-Gemeinde zu Fehlerfällen und der Umstierg auf ein anderes Search-Backend dokumentiert.

Bitte konsultieren Sie bei Detailfragen auch das mitgelieferte Programmer Guide, hier sind die Einzelheiten von MyCoRe genauer beschrieben.

ACHTUNG, diese Dokumentation basiert auf dem Release 1.0 der MyCoRe-Software!

1.Voraussetzungen für eine MyCoRe Anwendung

1.1 Vorabbemerkungen

Das MyCoRe-Projekt ist so designed, dass es dem einzelnen Anwender frei steht, welche Komponenten er für die Speicherung der Daten verwenden will. Dabei spielt natürlich das verwendete Betriebssystem eine wesentliche Rolle. Jedes System hat seine eigenen Vor- und Nachteile, die an dieser Stelle nur kurz diskutiert werden sollen. Wir wollen es dem Anwender überlassen, in welchem System er für seine Anwendung die größten Vorteile sieht. Nachfolgend finden Sie eine Tabelle der wesentlichen eingesetzten Komponenten entsprechend des gewählten Basissystems.

Allg. Basis	komerzieller IBM Content Manager
Metadaten Store	IBM Content Manager 8.2 mit integrierter Text-Suchmaschiene IBM NSE
Volltextsuche	Diese ist für die Nutzung des IBM CM Ressource Managers voll integriert.
Datenbank	Es wird die mitgelieferte DB2 Datenbank benutzt.
Dokument Stores	Filesystem IBM CM Ressource Manager IBM Video Charger Helix Server
Web-Server	IBM WebSphere
Servlet-Engine	IBM WebSphere
OS	IBM AIX Sun Solaris Linux Microsoft Windows
Vorteile	- Parametrische Metadaten-Werte können ggf. auch volltextindiziert und so gesucht werden werden. - Gute Implementierung der an XPath angelehnten Abfragesprache - Viele Dokumenttypen lassen sich automatisch volltextindizieren. - Das Basisprodukt kommt von einem Hersteller. - Viel unterstützte Dateiformate für die Volltextsuche
Nachteile	- Sehr komplexe Installation des benötigten Content Manager /WebSphere Systems. - Abhängigkeit von den IBM Lizenz-Bedingungen.

Tabelle 1: MyCoRe Komponentenübersicht für IBM CM

Allg. Basis	Lösung auf freien Komponenten
Metadaten Store	Es wird die freie XML:DB eXist benutzt.
Volltextsuche in Datenbank	Es wird die freie Text-Suchmaschine Lucene benutzt. Es wird die freie Datenbank MySQL benutzt.
Dokument Stores	Filesystem Lucene für Textdokumente IBM Video Carger Helix Server
Web-Server	Apache / Tomcat
Servlet-Engine	Tomcat
OS	Linux Microsoft Windows alle weiteren UNIX-Systeme, sofern die benötigten Komponenten vorhanden sind
Vorteile	- Geringe Kosten durch die Nutzung freier Komponeneten. - Keine direkte Bindung an einen Hersteller. - Die Komponenten sind fast plattformunabhängig, einfach zu installieren, gut getestet und dokumentiert. - Es kann schnell eine Anwendung fertiggestellt werden.
Nachteile	- Abhängikeit von der Entwicklung in der OpenSource-Gemeinde. - Keine Produkthaftung für Komponenten. - Derzeit werden nur PDF und TXT für die Volltextsuche unterstützt. - Xpath-Abfragen unter eXist implementieren nicht alle Möglichkeiten der Xquery-Spezifikation und können ggf. falsche Angaben zurückliefern.

Tabelle 2: MyCoRe Komponentenübersicht für freie Lösung

Allg. Basis	Lösung auf freien Komponenten (**Standardinstallation**)
Metadaten Store	Es wird die Metadatensuche via XSLT benutzt.
Volltextsuche in Datenbank	Es wird die freie Text-Suchmaschine Lucene benutzt. Es wird die freie Datenbank MySQL benutzt.
Dokument Stores	Filesystem Lucene für Textdokumente IBM Video Carger Helix Server
Web-Server	Apache / Tomcat
Servlet-Engine	Tomcat

OS	Linux Microsoft Windows alle weiteren UNIX-Systeme, sofern die benötigten Komponenten vorhanden sind
Vorteile	- Geringe Kosten durch die Nutzung freier Komponeneten. - Keine direkte Bindung an einen Hersteller. - Die Komponenten sind fast plattformunabhängig, einfach zu installieren, gut getestet und dokumentiert. - Es kann schnell eine Anwendung fertiggestellt werden.
Nachteile	- Abhängikeit von der Entwicklung in der OpenSource-Gemeinde. - Keine Produkthaftung für Komponenten. - Derzeit werden nur PDF und TXT für die Volltextsuche unterstützt. - **Das System arbeitet nur mit kleinen Datenmengen derzeit vernünftig. Bei einer großen Anzahl von Datensätzen wird die Suche extrem verlangsamt. Diese Variante dient nur Demonstrationszwecken.**

Tabelle 3: MyCoRe Komponentenübersicht für freie Lösung

1.2 Allgemein benötigte Software

Zum Betrieb von MyCoRe sind zuerst einmal die folgenden freien Komponenten nötig:

- Java SDK – es ist mindestens ein SDK 1.4.2 erforderlich. Java wird normalerweise mit dem Betriebssystem ausgeliefert. Eine Installation ist in den entsprechenden Abschnitten beschrieben.
- BASH (erforderlich unter Unix-Systemen) – Alle zusätzlichen Unix-Shell-Scripts basieren auf der bash-Shell. Es ist also sehr sinnvoll, diese auf dem System mit zu installieren, sofern sie noch nicht vorhanden ist.
- CVS-Client (erforderlich) – Dieser dient dem Checkout des aktuellen MyCoRe-Codes und ist für den Einbau von Bug-Fixes unerlässlich. CVS ist in der SuSE-Linux-Distribution enthalten. Für andere Systeme muss es von einem separaten Server[1] nachgeladen werden.
- ANT (erforderlich) – Unter Linux ist dieses Tool in den Distributionen enthalten, für die anderen Systeme muss es vom Server des Apache-Projektes[2] geholt werden. Es sollte mindestens Version 1.5.x zum Einsatz kommen.
- OpenOffice (für Linux erforderlich) – Unter Linux ist dieses Paket mit in der Distribution enthalten. Wichtig ist , dass es als **mcradmin** einmal gestartet wurde, damit lokale Konfigurationen angelegt sind. Das Paket wird zur Volltextindizierung verwendet. Alternativ kann man das Produkt auch direkt beziehen[3].
- XPDF (für Linux) – Das Paket ist in den gängigen Linux-Distributionen in Version 3.00... enthalten.

1 http://cvshome.org/
2 http://ant.apache.org/
3 http://www.openoffice.org/

- GhostScript (für Linux) - Das Paket ist in den gängigen Linux-Distributionen in Version 7.07... enthalten.
- ImageMagick (optional) – Ein mächtiges Bildkonverter-Programm, welches bei der Bearbeitung großer Bildmengen sehr hilfreich ist. Auch dieses Produkt ist in Linux-Distributionen mit enthalten.

Für das Betriebssystem AIX stellt die Firma IBM einen Software-Download unter http://ftp.software.ibm.com/ bereit.

1.3 Hinweise zur Installation des IBM Content Manager 8.2

Die Installationshinweise für den IBM Content Manager sind recht umfangreich. Wir haben uns daher entschieden, diese in einem gesonderten Dokument abzulegen. Es enthält sowohl die Hinweise für die Installation der Komponenten unter AIX wie auch unter Windows und Linux. Die Schrift gehört zur Reihe der MyCoRe Quick Guides und heißt **'Installing IBM CM 8.2'**. Bitte konsultieren Sie erst diese Schrift, bevor sie mit der Installation vom MyCoRe und der Beispielanwendung auf IBM Content Manager Basis fortfahren. Die in der nachfolgenden Beschreibung angegebenen Arbeiten gehen davon aus, dass der Nutzer für des MyCoRe-Kerns und des Beispiels mit **mcradmin** angegeben ist.

1.4 Hinweise zu vorbereitenden Installationen freier Datenbanken, Text-Suchmaschinen und Web–Komponenten

1.4.1 Installation der Komponenten unter Linux

Die folgende Beschreibung erläutert die vorbereitenden Arbeiten unter SuSE Linux. Getestet wurde unter der Version 9.1. Es ist besonders aus Sicherheitsgründen besonders zu empfehlen, auch die vom Distributor angebotenen Updates für die nachfolgenden Komponenten nachzuinstallieren. Sollten für RedHat Abweichungen auftreten, so werden diese gesondert vermerkt.

1.4.1.1 Installation von Java

Als erster Schritt ist der Java-Compiler J2SE 1.4.1 oder höher zu installieren. JRE und SDK befinden sich unter in der SuSE in der Distribution. Zum Test unter 9.1 wurden Java 1.4.2 verwendet. Probleme mit höheren Versionen sind bisher nicht bekannt.

- **java2-jre-1.4.2-...**
- **java2-1.4.2-...**

1.4.1.2 Die Installation von MySQL

MySQL ist eine derzeit freie relationale Datenbank, welche zur schnellen Speicherung von Daten innerhalb des MyCoRe-Projektes benötigt wird. Sie besitzt eine JDBC Schnittstelle und ist SQL konform. Sie könnten MySQL auch durch eine andere verfügbare Datenbank mit gleicher Funktionalität ersetzen. Der Test erfolgte mit einem MySQL 4.x System, höhere Versionen sollten keine Probleme bereiten.

1. Installieren Sie aus Ihrer Distribution die folgenden Pakete und danach ggf. noch vom Hersteller der Distribution per Netz angebotene Updates. Die angegebenen Versionsnummern sind nur exemplarisch.
   - **mysql-4.0.18-...**
   - **mysql-shared-4.0.18-...**
   - **mysql-client-4.0.18-...**
   - **mysqlcc-0.9.4-84**
2. Die Dokumentation steht nun unter */usr/share/doc/packages/mysql*.
3. Führen Sie als **root** das Kommando rcmysql start aus.
4. Führen Sie als **root** das Kommando aus
   /usr/bin/mysqladmin -u root password *new-password*
   /usr/bin/mysqladmin -u root -h <full_host_name> password *new-password*
5. Die folgende Sequenz sorgt dafür, dass der MyCoRe-User **mcradmin** alle Rechte auf der Datenbank hat. Dabei werden bei der Ausführung von Kommandos von **localhost** aus keine Passwörter abgefragt. Von anderen Hosts aus muss *ein-password* eingegeben werden.
   mysql –uroot -p*PASSWORD* mysql
   GRANT ALL PRIVILEGES ON *.* TO mcradmin@localhost WITH GRANT OPTION;
   GRANT ALL PRIVILEGES ON *.* TO mcradmin@'%' IDENTIFIED BY '*ein-passwort*' WITH GRANT OPTION;
   quit
6. Ist das Password einmal gesetzt, müssen Sie zusätzlich die Option -p verwenden.
7. Zum Verifizieren, ob der Server läuft, nutzen Sie folgende Kommandos
   mysqladmin -u mcradmin version
   mysqladmin -u mcradmin variables.
8. Jetzt können Sie die Datenbasis für MyCoRe mit nachstehendem Kommando anlegen.
   mysqladmin -u mcradmin create mycore
9. Falls weitere Benutzer noch das Recht auf Selects von allen Hosts aus haben sollen, verwenden Sie die Kommandos
   mysql –u mcradmin mycore
   GRANT SELECT ON mycore.* TO mcradmin@'%';
   quit

Falls sie keine Connection auf ihren Rechnernamen (nicht localhost) aufbauen können, kann es auch mit ihrer Firewall oder TCPWrapper Einstellung zu tun haben. Bei einer Firewall sollte der Port 3306 freigegeben werden und bei einem TCPWrapper der entsprechende Dienst (**mysqld**) in die Datei */etc/hosts.allow* geschrieben werden.

1.4.1.3 Der JDOM-Search-Layer für das Beispiel

Um eine erste Installation von MyCoRe durchzuführen, ist es erstmal nicht erforderlich, einen externen Search-Layer (also ein Backend für die Suche in den XML-Daten) zu haben. Standardmäßig wird hier das in MyCoRe enthaltene XALAN verwendet. Leider hat die Einfachheit dieser Lösung auch ihre Schattenseiten. Das System ist bei mehr als 100 Datensätzen nicht mehr sehr performant. Ersetzen Sie also für ein Produktionssystem den JDOM-Search-Layer durch eine

XML:DB wie **eXist**[4]. Eine Anleitung zu dieser Migration finden Sie in Kapitel 4.

1.4.1.4 Die Nutzung der freien Text-Suchmaschine Lucene zur Volltextsuche

Um die frei verfügbare Text-Suchmaschine für die Volltextsuche Lucene des Apache Jakarta Projektes[5] zu benutzen, bedarf es keiner zusätzlichen Installationen. Derzeit ist die Version Lucene 1.4.2 aktuell. Die erforderliche ***lucene-1.4.2.jar*** Datei wird unter dem MyCoRe-Kern im Verzeichnis *~/mycore/lib* mitgeliefert. Zur Nutzung ist lediglich die weiter hinten beschriebene Konfiguration in der MyCoRe-Anwendung nötig.

Im derzeitigen Entwicklungsstand können mit der MyCoRe Variante auf Basis freier Software nur nachfolgende Dateitypen textindiziert werden. Hierzu müssen die Tools OpenOffice und xpdf installiert worden sein. Diese sollten in den gängigen Linux-Distributionen enthalten sein.

Achtung, es können selbstverständlich nur PDF-Dateien indiziert werden, welche auch reinen Text und nicht den Text als Bild enthalten! Die Praxis hat gezeigt, dass einige PostScript Ausprägungen nicht gelesen werden können (Fehler beim Öffnen)!

1.4.1.5 Die Installation von Apache 2

Im Linux-Umfeld kann der Apache-Webserver als Standard betrachtet werden. Er ist in allen gängigen Distributionen enthalten. Die allerneuesten Apache 2 Pakete sind zu finden unter ftp://ftp.suse.com/pub/projects/apache/apache2/9.0-i386 Der Test erfolgte unter SuSE 9.1 mit folgenden Versionen:

- **apache2-2.0.49-...**
- **apache2-doc-2.0.49-...** (dieses Paket ist optional)

Mit rcapache2 start wird der Server per Default Einstellungen gestartet und sollte jetzt über `http://localhost` erreichbar sein. Etwaige Fehlermeldungen werden in die Fehler-Datei */var/log/apache2/error_log* geschrieben.
Die zentrale Konfigurationsdatei des Apache2 ist die unter */etc/apache2* liegende Datei *httpd.conf*. Bevor irgendwelche Änderungen vorgenommen werden bitte immer ein Sicherheitskopie anlegen. Der Befehl (als **root**) /usr/sbin/apache2ctl configtest überprüft die Syntax ihrer geänderten Einstellungen. Vorerst sollten jedoch die Standard Einstellungen genügen. Mit rcapache2 [start; stop; restart] als **root** kann der Server gestartet gestoppt bzw. neu gestartet werden und über rcapache2 status wird der aktuelle Staus abgefragt. Die Einbindung von virtuellen Web-Serveren und von Tomcat wird weiter hinten besprochen.

1.4.1.6 Die Installation von Tomcat 5

Wie Apache, dessen Installation im vorigen Abschnitt beschrieben wurde, ist auch Tomcat als Servlet-Engine im Linux-Umfeld ein Quasi-Standard. Für die Arbeit der Web-Anwendung des

4 http://www.exist-db.org/
5http://jakarta.apache.org/lucene/docs/index.html

MyCoRe-Projektes wird dieses Tool zwingend benötigt. Getestet wurde unter SuSE 9.1 mit folgenden Komponenten:

- **jakarta-tomcat-5.0.19-...**
- **apache2-jakarta-tomcat-connectors-5.0.19-...**

Im File */etc/profile.local* ist folgender Eintrag vorzunehmen, damit Tomcat auch von **mcradmin** genutzt werden kann. Weiterhin werden der Memory Size für den Server erweitert.

- export CATALINA_HOME=/opt/share/tomcat
- export CATALINA_OPTS="$CATALINA_OPTS -server -Xms256m -Xmx1800m -Xincgc"

MyCoRe arbeitet beim Encoding der Zeichen satndardmäßig mit UTF-8. Um dies auch dem Tomcat zu vermitteln und damit alle Abfragen und Verarbeitungen, welche Umlaute verwenden, korrekt laufen, müssen Sie die Datei */usr/share/tomcat/conf/server.xml* anpassen. Ergänzen Sie das Tag für den **Connector** um das Attribut **URIEncoding="UTF-8"** .

Mit rctomcat [start; stop; restart] als **root** kann der Server gestartet gestoppt bzw. neu gestartet werden und über rctomcat status wird der aktuelle Staus abgefragt. Die Dokumentation von Tomcat steht unter */usr/share/doc/packages/jakarta-tomcat* .

1.4.2 Installation der Komponenten unter MS Windows

1.4.2.1 Installation von Java

Dieser Teil der Dokumentation folgt zu einem späteren Zeitpunkt.

1.4.2.2 Installation von MySQL

Dieser Teil der Dokumentation folgt zu einem späteren Zeitpunkt.

1.4.2.3 Der JDOM-Search-Layer für das Beispiel

Um eine erste Installation von MyCoRe durchzuführen, ist es erstmal nicht erforderlich, einen externen Search-Layer (also ein Backend für die Suche in den XML-Daten) zu haben. Standardmäßig wird hier das in MyCoRe enthaltene XALAN verwendet. Leider hat die Einfachheit dieser Lösung auch ihre Schattenseiten. Das System ist bei mehr als 100 Datensätzen nicht mehr sehr performant. Ersetzen Sie also für ein Produktionssystem den JDOM-Search-Layer durch eine XML:DB wie **eXist**[6]. Eine Anleitung zu dieser Migration finden Sie in Kapitel 4.

6 http://www.exist-db.org/

1.4.2.4 Die Nutzung der freien Text-Suchmaschine Lucene zur Volltextsuche

Dieser Teil der Dokumentation folgt zu einem späteren Zeitpunkt.

1.4.2.5 Installation von Apache 2

Dieser Teil der Dokumentation folgt zu einem späteren Zeitpunkt.

1.4.2.6 Installation von Tomcat 5

Dieser Teil der Dokumentation folgt zu einem späteren Zeitpunkt.

2.Download und Installation des MyCoRe Kerns

2.1 Download des MyCoRe Kerns

Derzeit steht der MyCoRe-Kern unter server.mycore.de. Sie müssen also unter einem Unix-System zuerst ein kleines Shell-Script *cvsmycore.sh* erstellen, welches die erforderlichen Variablen setzt.

```
# Script zum Zugriff auf das CVS in Essen
export CVS_CLIENT_LOG=~/cvs_log
export TERM=vt100
export CVSROOT=:pserver:anoncvs@server.mycore.de:/cvs
export CVS_RSH=ssh
cvs $*
```

Für MS Windows sollte alternativ folgendes Script als *cvsmycore.cmd* werden:

```
rem Script zum Zugriff auf das CVS in Essen
CVS_CLIENT_LOG=cvs_log
CVSROOT=:pserver:anoncvs@server.mycore.de:/cvs
cvs $*
```

Führen Sie zuerst folgendes Kommando aus (**das Passwort ist anoncvs**)

```
touch ~/.cvspass ;  cvsmycore.sh login
```

oder

```
cvsmycore login
```

Der MyCoRe Kern wird für alle unterstützten Systeme über das CVS Repository ausgeliefert. Das Holen der aktuellen Version erfolgt mit dem Kommando

```
cvsmycore.sh checkout mycore
```

oder

```
cvsmycore checkout mycore
```

Nach dem erfolgreichen CheckOut erhalten Sie unter dem Verzeichnis *~/mycore* folgende Dateistruktur.

Relativer Pfad	**Inhalt**
bin	Das Verzeichnis für die Shell-Scripts des Kerns.
bin/build.cmd	Startet den Build-Prozess unter einem Windows-System.
bin/build.sh	Startet den Build-Prozess unter einem Unix-System.
bin/build.properties.template	Konfigurationsdatei für den Kern, welche die Pfade zu den verwendeten Datenspeichern enthält.
build.xml	Die Konfigurationsdatei für den Build-Prozess.
documentation	Enthält alle MyCoRe-Dokumentationen.

Relativer Pfad	Inhalt
documentation/StartingGuide	Eine kurze Einleitung in das Projekt.
documentation/UserGuide	Das Handbuch für die Standard-Installation von MyCoRe.
documentation/QuickGuide	Einige Kurzbeschreibungen zu ganz bestimmten Spezialthemen rund um MyCoRe.
documentation/ProgGuide	Dieses Handbuch enthält tiefere Hintergrundinformationen zu MyCoRe.
lib	Java-Archive von Komponenten, welche in MyCoRe genutzt werden.
license.txt	Die verbindliche Lizenz von MyCoRe.
modules	Zusätzliche modulare Programmkomponenten.
schema	XML Schema Dateien des MyCoRe-Kerns.
sources//org	Verzeichnis des Java-Quellcodes. Eine Beschreibung der Pakete steht im ProgrammerGuide.
stylesheets	XSLT Stylesheets des MyCoRe-Kerns.

Tabelle 4: Aufbau des MyCoRe-Kerns

Es ist erforderlich, diese Pfadangabe in der Umgebungsvariablen MYCORE_HOME abzulegen.

export MYCORE_HOME=~/mycore

oder

MYCORE_HOME=mycore

Es wird immer wieder Korrekturen am Code geben, welche bestehende Fehler beseitigen. Es ist daher immer mal sinnvoll, ein Update zu fahren. Das folgend Kommando kann in abgewandelter Form auch für die anderen MyCoRe-Komponenten benutzt werden.

```
cvsmycore.sh update -dP mycore
```

oder

cvsmycore update -dP mycore

2.2 Konfiguration und Übersetzten des Kerns

1. MyCoRe verwendet das Apache Ant Build-Tool, um den Quellcode zu übersetzen und eine vollständige Beispiel-Applikation zu erzeugen. Entsprechend der Installationsanleitung des Ant-Paketes sollten Sie zunächst die Umgebungsvariable `JAVA_HOME` und `ANT_HOME` gesetzt haben. Sollten diese Variablen auf Ihrem System noch nicht gesetzt sein, können Sie dies in der Datei `build.sh` (Unix) bzw. `build.cmd` (Windows) nachholen und korrigieren.
2. Es ist nicht nötig, weitere Umgebungsvariablen wie etwa den Java CLASSPATH zu setzen. Das MyCoRe Ant Build-Skript ignoriert den lokal gesetzten `CLASSPATH` völlig und generiert stattdessen einen eigenen `CLASSPATH` entsprechend Ihrer Konfiguration. Somit können wir sicherstellen, dass nur die erforderlichen Pakete und Klassen in der richtigen Version verwendet werden. Die Konfiguration der Systemumgebung der verwendeten Datenbanken für die XML-

Speicherung (JDOM, IBM CM8, eXist) und die Speicherung von Tabellen über JDBC in einer relationalen Datenbank (IBM DB2, MySQL, optional auch andere) wird in der Datei *bin/build.properties* festgelegt.

3. In der Regel werden Sie nur die beiden entsprechenden Blöcke für die verwendete XML-Datenbank (`MCR.XMLStore.*`) - **Standard ist JDOM** - und die verwendete relationale Datenbank (`MCR.JDBCStore.*`) - **Standard ist MySQL** - durch kommentieren bzw. auskommentieren der vorgegebenen Zeilen und Anpassen der beiden Variablen `MCR.XMLStore.BaseDir` und `MCR.JDBCStore.BaseDir` an die lokalen Installationsverzeichnisse Ihrer Datenbanksysteme anpassen müssen. Zu Testzwecken können die vorgegebenen Standards verwendet werden. Die weiteren Variablen steuern die für den Betrieb notwendigen JAR-Dateien (`MCR.*Store.Jars`), eventuell zusätzlich in den CLASSPATH einzubindende class-Dateien oder Ressourcen (`MCR.*Store.ClassesDirs`) und zur Laufzeit erforderliche native Libraries bzw. DLLs (`MCR.*Store.LibPath`). Passen Sie die Werte entsprechend der Dokumentation Ihres Datenbankproduktes und der Kommentare in der Datei selbst an.
4. Sie sollten zunächst prüfen, ob ihre Systemumgebung korrekt eingerichtet ist, indem Sie

   bin/build.sh info bzw. bin\build.cmd info

   ausführen. Das Ant Build Tool zeigt Ihnen daraufhin die verwendeten JDK- und Ant-Software-Versionen und den generierten CLASSPATH und LIBPATH (für Unix Systeme) an.
5. Sollten Sie festgestellt haben, dass Ihr JDK ab 1.4.x eine andere Xalan-Version benutzt (ab 1.4.x ist Xalan im SDK), führen Sie bitte folgende Kommandos aus und prüfen Sie danach Ihr System erneut.
   - cd $JAVA_HOME/jre/lib
   - mkdir endorsed
   - cd endorsed
   - cp $MYCORE_HOME/lib/xerces* .
   - cp $MYCORE_HOME/lib/xalan* .
   - cd $JAVA_HOME/lib
   - ln -s ../jre/lib/endorsed endorsed
6. Eine Übersicht über alle wesentlichen Build-Ziele erhalten Sie mit

   bin/build.sh usage bzw. bin\build.cmd usage
7. Übersetzen Sie alle MyCoRe Quellcode-Dateien mit dem Befehl

   bin/build.sh jar bzw. bin\build.cmd jar

   Dabei entsteht, abhängig von dem von Ihnen gewählten Datenbank-System zur Speicherung der XML-Daten eine Jar-Datei *lib/mycore-for-[jdom-cm8—xmldb].jar*.
8. Optional können Sie auch JavaDoc Quellcode-Dokumentation im HTML-Format generieren lassen, indem Sie

   `bin/build.sh javadocs` bzw. `bin\build.cmd javadocs`

   aufrufen. Dabei entstehen HTML-Dateien im Verzeichnis *documentation/html*.

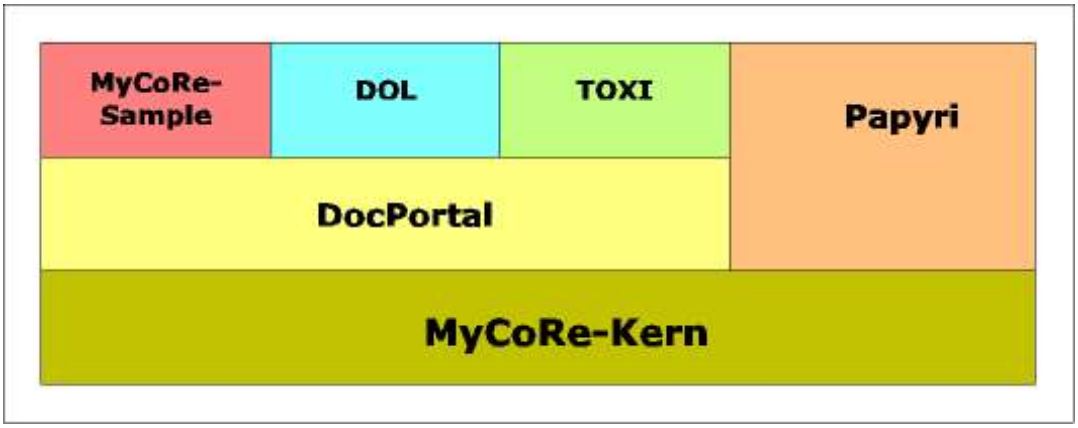
3.Die MyCore Beispielanwendung MyCoReSample

3.1 Grundlegender Aufbau und Ziel der Beispielanwendung

3.1.1 Allgemeines

Um die Funktionsweise des MyCoRe-Kernes zu testen wurde eine Beispiel-Anwendung basiered auf diesem Kern entwickelt. Sie soll dem Anwender eine voll funktionsfähige Applikation in die Hand geben, welche eine Vielzahl von Komponenten des MyCoRe-Projektes nutzt und deren Arbeitsweise klar werden lässt. Um die Anwendung, im weiteren MyCoReSample genannt, gleichzeitig sinnvoll einsetzten zu können, wurde als Beispielszenario ein Dokumenten-Server gewählt, wie er bei vielen potentiellen Nutzern zur Anwendung kommt. Auch soll das MyCoReSample die Nachfolge des erfolgreichen MILESS-Dokumenten-Servers sein und den Migrationspfad zu MyCoRe hin aufzeigen.

Die gesammte Applikation ist in mehrere Ebenen aufgeteilt. Dies gestattet eine flexible Konfiguration auf einem Zielsystem bei gleichzeitiger Nutzung mehrerer gleichartiger Anwendungen auf diesem System. Beispielsweise wollen Sie einen Dissertations-Server neben einem Server für eine Bildsammlung und einem allgemeinen Dokumenten-Server auf dem selben System laufen lassen. Alle drei basieren auf DocPortal und nutzen große Teile wie Datenmodell oder Editormasken davon gemeinsam. Diese Komponenten werden in der DocPortal-Basis untergebracht, während die Konfiguration der Tabellennamen jeweils unterschiedlich ist. Die möglichen Szenarien verdeutlicht die folgende Grafik. Für jede Anwendung gibt es im CVS noch eine zugehörige Tomcat-Anwendung welche die vollständige Installation beschleunigen soll.



3.1.2 Weiterführende Erläuterungen

Die weiterführenden Erläuterungen zum Datenmodell und der Struktur des MyCoReSamples finden Sie in der Dokumentation zu DocPortal. In dieser Schrift ist das Konzept des Dokumenten-Servers und seiner individuellen Gestaltung als MyCoreSample beschrieben.

Auf der offiziellen Web-Seite des MyCoRe-Projektes[7] finden sie die in den nachfolgenden Abschnitten zu installierende Anwendung bereits lauffähig vor. Nach Abschluss der Arbeiten sollte

7 http://www.mycore.de

auch Ihr System so funktionieren.

3.2 Download der Beispielanwendung

Nachdem Sie den MyCoRe-Kern erfolgreich installiert haben, ist nun die Installation der mitgelieferten Beispiel-Anwendung sinnvoll. Hier können Sie ein erstes Gefühl dafür gewinnen, wie eine eigene Anwendung gestaltet sein könnte. Das MyCoReSample wird für alle unterstützten Systeme über das CVS Repository ausgeliefert.

Das Holen der aktuellen Version von **DocPortal** und **MyCoReSample** erfolgt mit den Kommandos

```
cvsmycore.sh checkout docportal
cvsmycore.sh checkout mycoresample
```

oder

```
cvsmycore checkout docportal
cvsmycore checkout mycoresample
```

Nach erfolgreichem CheckOut finden Sie nun zwei Verzeichnisse – docportal für die DocPortal-Basis-Applikation und mycoresample als spezielle Applikation. Der folgende Abschnitt beschreibt deren Konfiguration. Für Updates dieser Komponenten verfahren Sie wie in Abschnitt 2.1 beschrieben.

3.3 Konfiguration im Applikationsteil DocPortal

3.3.1 Allgemeines

Die Anwendungskomponente DocPortal enthält alle Dateien, welche für alle Applikationen der Klasse Dokumenten-Server gemeinsam genutzt werden sollen. Dies sind u. a. die Datenmodelldefinition, allgemeine Web-Seiten, Stylesheets zur Darstellung von Suche und Ergebnisanzeige, allgemeingültige Klassifikationen.

3.3.2 Installation

Die Installation der Zwischenschicht DocPortal gestaltet sich denkbar einfach. Führen Sie die nachfolgenden Arbeitsschritte einfach nacheinander durch.

1. Setzen Sie die Umgebungsvariable DOCPORTAL_HOME in Ihrem System.

```
export DOCPORTAL_HOME=~/docportal
```

oder

```
DOCPORTAL_HOME=docportal
```

2. Prüfen Sie die Systemumgebung in DocPortal mit

   bin/build.sh info bzw. bin\build.cmd info

3. Compilieren Sie die zusätzlichen Java-Klassen mit dem Kommando

   bin/build.sh jar bzw. bin\build.cmd jar

4. Generieren Sie die XML Schema Dateien mit dem Kommando

   bin/build.sh schema bzw. bin\build.cmd schema

3.4 Konfiguration im Applikationsteil MyCoReSample

3.4.1 Grundlegendes zur Konfiguration

Bevor Sie beginnen, sollte als erstes die Umgebungsvariable für das MyCoRSample-Home-Verzeichnis gesetzt werden.

export MYCORESAMPLE_HOME=~/mycoresample

oder

MYCORESAMPLE_HOME=mycoresample

Die Konfiguration der Beispielanwendung MyCoReSample ist in zwei Verzeichnissen zu finden. Im Verzeichnis $DOCPORTAL_HOME/config sind alle Dateien untergebracht, die Sie für eine erste oder einfache Installation der Anwendung nicht ändern müssen. Die Voreinstellungen entsprechen dem Standard und sollten ohne Probleme funktionieren.

Im Verzeichnis $MYCORESAMPLE_HOME/config hingegen sind die zu verändernden Konfigurationen hinterlegt. Kopieren Sie als erstes die Datei *mycore.properties.private.template nach mycore.properties.private* . Dieses File enthält für den Anfang alle einzustellenden Werte, die an Ihre spezielle Systemumgebung angepasst werden müssen. Damit dies privaten Einstellungen nicht durch ein Update überschrieben werden, wurde auf die Template-Variante zurückgegriffen. Achten Sie bei Updates stets darauf, ob sich im template etwas geändert hat!

Die anschließenden Abschnitte des Kapitels beschäftigen sich mit der Bearbeitung der Konfigurationsdatei und behandeln jeweils einen Komplex wie JDBC, Logger, usw. Es ist aber sehr empfehlenswert, wenn Sie sich die ganze Datei sorgsam durchsehen!

Die MyCoReSample Gesamtanwendung baut sich aus den drei Komponenten MyCoRe-Kern, DocPortal und MyCoReSample auf. Beim Zusammenbau der Anwendung wird die jeweils niedrigere Komponente durch gleichnamige Dateien einer höheren Komponente überlagert. Das bedeutet, ggf. können z. B. Standardanpassungen aus DocPortal mit denen aus dem MyCoreSample überlagert werden. Ein Beispiel hierfür sind die Suchmaskendefinitionen (SearchMask...).

Die MyCoRe Beispiel-Anwendung verwendet die folgenden Dateien aus dem MyCore Kern.

- das File *$MYCORE_HOME/bin/build.properties*

- alle *.jar Files aus dem Verzeichnis *$MYCORE_HOME/lib*
- alle Stylesheets aus *$MYCORE_HOME/stylesheets*
- Konfigurationsdaten aus den Modulen

3.4.1.1 CM8 Store konfigurieren

Dieser Abschnitt ist nur für Sie interessant, wenn Sie sich für den IBM Content Manager als Backend entschieden haben.

Die Konfiguration geht von einer Standardinstallation des IBM Content Managers aus. Sie sollten nur folgende Einträge anpassen müssen:

- MCR.persistence_cm8_password
- MCR.persistence_cm8_textsearch_indexdir
- MCR.persistence_cm8_textsearch_workingdir

3.4.1.2 eXist Store Konfigurieren

Dieser Teil wird in Kapitel 4 beschrieben. Von der Auslieferung her benutzt das MyCoreSample den JDBC Store, welcher nicht konfiguriert werden muss.

3.4.1.3 JDBC-Treiber konfigurieren

Im MyCoRe-Projekt werden ein Teil der Organisations- und Metadaten in klassischen relationalen Datenbanken gespeichert. Um die Arbeit mit verschiedenen Anbietern möglichst einfach zu gestalten, wurde die Arbeit mit dieser Datenbank gegen die JDBC-Schnittstellen programmiert.

In der Konfigurationsdatei *mycore.properties.private* legen Sie im Parameter **MCR.persistence_sql_driver** fest, welcher JDBC-Treiber verwendet werden soll. Weiterhin müssen Sie die Variable **MCR.persistence_sql_database_url** anpassen, die die JDBC URL für Verbindungen zu Ihrer Datenbank festlegt. Achten Sie bei Verwendung von MySQL darauf, dass der richtige Datenbank-User (per Default **mcradmin**) angegeben ist. In einigen Fällen , z. B. unter SuSE 9.1, kann es notwendig sein, **localhost durch 127.0.0.1** zu **ersetzen**!

Weiterhin können Sie die minimale und maximale Anzahl der gleichzeitigen Verbindungen zur Datenbank in den beiden Parametern **MCR.persistence_sql_init_connections** und **MCR.persistence_sql_max_connections** festlegen.

3.4.1.4 Die SQL-Tabellennamen

Alle Namen der verwendeten SQL-Tabellen sind im darauf folgenden Abschnitt gelistet. Für eine erste Inbetriebnahme des MyCoReSamples müssen diese nicht geändert werden.

3.4.1.5 User-Management

Auch die Einträge für das User-Management können vorerst so bleiben. Das MyCoreSample liefert eine ganze Reihe von Privilegien, Gruppen und Benutzern mit, welche im Sample zur Darstellung der Funktionalitäten benötigt werden.

3.4.1.6 Remote-Access

MyCoRe bietet die Möglichkeit, mehrere Systeme einer Applikationsgruppe zusammen zuschalten und die MyCore-Oberfläche wie ein Portal zu betreiben. Im Simplen Fall können Sie also Ihr Testsystem auch über den Remote-Zugang abfragen. Dazu müssen Sie mindestens den Hostnamen unter **MCR.remoteaccess_selfremote_host** und die Portnummer unter **MCR.remoteaccess_selfremote_port** entsprechend Ihrer Systeminstallation eintragen. Es ist sinnvol hier den vorgesehenen Tomcat oder WebSphere-Port zu nehmen, da dies den Umweg über einen Apache-Zugriff erspart.

3.4.1.7 Speicherung von Objekten und Dokumenten konfigurieren

Neben den Metadaten sind im MyCoReSample auch eine Reihe von Dokumenten und Bildern zum Laden in die Beispielanwendung abgelegt. Je nach Eintrag in der Datei *ContentStoreSelectionRules.xml* werden zur Speicherung der Objekte und Dokumente entsprechende Stores herangezogen.

Sollten Sie sich für den IBM Content Manager entschieden haben, so kopieren sie die datei nach *$MYCORESAMPLE_HOME/config* und passen Sie sie entsprechend an. Der Syntax ist selbsterklärend. Für die Nutzung von CM8 sind keine weiteren Anpassungen nötig, die Textindizierung wird bei richtiger Konfiguration des IBM Content Managers automatisch durchgeführt.

3.4.1.7.1 File System Store

Standardmäßig wird alles in ein Plattenverzeichnis abgelegt, welches mit dem Parameter **MCR.IFS.ContentStore.FS.BaseDirectory** definiert wird. MyCoReSample benutzt ein Verzeichnis *$MYCORESAMPLE_HOME/filestore,* welches automatisch angelegt wird.

3.4.1.7.2 Lucene Store

Die auf der freien Komponente Lucene basierende Variante der Textindizierung basiert auf Tools wie OpenOffice, GhostScript und XPDF. Sie ist Standardmäßig in der Konfigurationsdatei *ContentStoreSelectionRules.xml* vordefiniert. es Können *.pdf, *.doc, *.txt und *.ps Objekte indiziert werden.

Lucene benötigt einen Speicherberich für die Ablage der Indizes. Hier sollte das Verzeichnis $MYCORESAMPLE_HOME/lucenestore genutzt werden, welches automatisch angelegt wird. Die Eigentlichen Objekte sollten auch nach *$MYCORESAMPLE_HOME/filestore* gespeichert werden. Bearbeiten Sie also die Werte von *MCR.IFS.ContentStore.Lucene.IndexDirectory* und *MCR.IFS.ContentStore.Lucene.BaseDirectory* entsprechend.

3.4.1.8 Editor

Hinsichtlich der Konfiguration des Editors ist nur zu beachten, dass das im Parameter **MCR.Editor.FileUpload.TempStoragePath** angegebene Verzeichnis auch angelegt ist.

3.4.1.9 Workflow

Im Konfigurationsabschnitt Workflow sind eine ganze Reihe von Einstellungen angegeben. Für den Anfang sollten Sie nur die Werte für die Parameter **MCR.editor_..._directory**, **MCR.editor_author_..._mail** und **MCR.editor_mail_sender** anpassen.

3.4.1.10 Appletsignatur

Zum signieren des FileUpload-Applets benötigen Sie einen Schlüssel, welchen Sie im weiteren Verlauf der Installation erzeugen müssen. Die Parameter hierfür können Sie übernehmen, nur das Verzeichnis der Keys sollten Sie mit dem Parameter **SIGN.KeyStore** anpassen. Das Verzeichnis wird automatisch erzeugt.

3.4.1.11 Debug konfigurieren

Innerhalb des MyCoRe-Projektes wird zum Erzeugen aller Print-Ausgaben für das Commandline-Tool und/oder die Stdout-Logs das externe Paket **log4j**des Apache-Jakarta-Projektes benutzt[8] . Dieses ist mittlerweile ein Quasistandard und ermöglicht eine gezielte Steuerung der Informationen, welche man erhalten möchte. In der Grundkonfiguration in mycore.properties.private ist der Output-Level INFO eingestellt. Eine zweite Standardvorgabe ist DEBUG, diese ist auskommentiert und kann alternativ bei Problemen genommen werden. **log4j** bietet jedoch darüber hinaus noch viele weitere Möglichkeiten, die Sie bitte der Dokumentation zu diesem Produkt entnehmen. Ergänzend sei auch auf das MyCoRe-Konfigurationsfile mycore.properties.logger hingewiesen.

3.4.2 Laden der Beispiel-Inhalte

Nachdem Sie nun die Konfiguration abgeschlossen haben, können Sie beginnen, die Beispieldaten in das System zu laden. Gehen Sie Schritt für Schritt vor und prüfen Sie das Ergebnis. Oft kommt es zu Fehlern, die auf den ersten Blick unerklärlich erscheinen. prüfen Sie in dem Fall zu allererst Ihre Konfigurationsparameter kritischq. Im Zweifelsfall erhalten Sie Hilfe über die MyCoRe-Mailing-Liste mycore-user@lists.sourceforge.net .

1. Erzeugen Sie die Commandline Tools mit Hilfe des Kommandos

   bin/build.sh create.unixtools

   bzw.

   bin\build.cmd create.unixtools

2. Wenn Sie es nicht schon getan haben, sollten Sie nun die SQL Datenbank anlegen.

8http://jakarta.apache.org/log4j/docs/index2.html

- mysql
- create database mycore
- quit

3. Laden Sie die Benutzer, Gruppen und Privilegien für die Beispielanwendung mit
   bin/build.sh create.users bzw. bin\ build.cmd create.users
4. Wenn Sie sich für das IBM Content Manager System als Backend entschieden haben, ist es jetzt erforderlich, die ItemTypes im LibraryServer anzulegen. Das geschieht mittlerweile

   bin/build.sh create.metastore bzw. bin\ build.cmd create.metastore

Nun können Sie die Beispieldaten unter Unix laden. Dazu wechseln Sie in das Verzeichnis unixtools.

1. cd $MYCORESAMPL_HOME/unixtools
2. Laden der mitgelieferten Klassifikationen mit ./ClassLoad.sh
3. Laden der mitgelieferten Institutionen mit ./SInstLoad.sh
4. Laden der mitgelieferten Authorendaten mit ./SAuthLoad.sh
5. Laden der mitgelieferten Dokumentdaten mit ./SDocLoad.sh
6. Laden der mitgelieferten Objekte mit ./SDocDerLoad.sh
7. Modifizieren Sie nun die Scripts ./Query....sh (Kommentar für jeweils eine Query entfernen) und lassen Sie einige Anfragen an das System durchlaufen. Hier können Sie testen, ob bisher alles korrekt läuft.

Das laden der Beispieldaten unter Window geht genauso. Dazu wechseln Sie in das Verzeichnis dostools.

1. cd $MYCORESAMPL_HOME/dostools
2. Laden der mitgelieferten Klassifikationen mit ./ClassLoad.cmd
3. Laden der mitgelieferten Institutionen mit ./SInstLoad.cmd
4. Laden der mitgelieferten Authorendaten mit ./SAuthLoad.cmd
5. Laden der mitgelieferten Dokumentdaten mit ./SDocLoad.cmd
6. Laden der mitgelieferten Objekte mit ./SDocDerLoad.cmd
7. Modifizieren Sie nun die Scripts ./Query....cmd (Kommentar für jeweils eine Query entfernen) und lassen Sie einige Anfragen an das System durchlaufen. Hier können Sie testen, ob bisher alles korrekt läuft.

Achtung, beim Laden der Klassifikationen kommt es zu einer Fehlermeldung, dass aus dem Verzeichnis $MYCORESAMPLE_HOME/content/classifications keine Dateien geladen werden. Die Meldung kann ignoriert werden, Hier würden nur zusätzliche Klassifikationen stehen, welche in erweiterten Dokumenten-Servern Anwendung finden.

3.5 Arbeiten mit dem MyCoRe Command Line Interface

3.5.1 Erzeugen der Skripte mycore.sh / mycore.cmd

Wenn Sie dieser Anleitung gefolgt sind, sollten Sie jetzt bereits alle erforderlichen Scripts für die Arbeit auf der Kommandozeile verfügbar haben. Es gib jedoc einige Fälle in denen das Basis-Script mycore.sh bzw. mycore.cmd noch einmal erzeugt werden muss, damit alle Classpath-Einträge richtig sind. Das kann z. B. sein, wenn bei einem Update ein *.jar-File in $MYCORE_HOME/lib ausgetaust wurde. Führen Sie in diesem Falle nachfolgendes Kommando aus.

bin/build.sh scripts bzw. bin/build.cmd scripts

Dieser Aufruf generiert die Shell-Skripte *bin/mycore.sh* (Unix) bzw. *bin/mycore.cmd* (Windows) neu.

3.5.2 Aufruf des CommandLineInterface

Starten Sie das MyCoRe Command Line Interface durch Aufruf von bin/mycore.sh (Unix) bzw. bin/mycore.cmd (Windows). Sie erhalten eine Übersicht über die verfügbaren Befehle durch Eingabe von **`help`** . Sie verlassen das CommandLineInterface durch Eingabe von **`quit`** oder **`exit`** .

Sie können natürlich auch den Aufruf in beliebige Scripts usw. einbinden, um eine Batch-Verarbeitung zu organisieren. Auch das laden der beispieldaten erfolgte auf diese Weise.

3.6 Erzeugen und Konfigurieren der Web-Anwendung

3.6.1 Erzeugen der Web-Anwendung

Durch Eingabe von

bin/build.sh webapps bzw. bin/build.cmd webapps

wird die MyCoRe Sample Web Application im Verzeichnis `webapps` erzeugt. Alternativ können Sie auch ein Web Application Archive (war) erzeugen, indem Sie

bin/build.sh war bzw. bin/build.cmd war

aufrufen.

Das MyCoRe Build-Script kopiert beim Erzeugen der Web Applikation auch alle externen, erforderlichen *.jar-Dateien Ihrer verwendeten Datenbank-Systeme (IBM Content Manager / DB2, MySQL, eXist) in das Verzeichnis `WEB-INF/lib`, entsprechend den Vorgaben Ihrer Konfiguration in `build.properties`. Beachten Sie dazu bitte die Hinweise in der Ausgabe beim Erzeugen der Web Application.

3.6.2 Konfiguration des Web Application Server

3.6.2.1 Tomcat

Die grundlegende Installation von Tomcat wurde bereits beschrieben. Nun soll auf dieser Basis das die WEB-Anwendung des MyCoReSamples installiert werden. Dabei ist an dieser Stelle nur ein einfaches Szenario auf der Basis der Tomcat-Grundinstallation beschrieben. Für die Konfiguration komplexerer Modelle, z. B. mehrere Applikationen nebeneinander, gibt es weiter hinten in diesem Dokument eine ausführliche Anleitung.

Folgende Schritte sind auszuführen:
1. bin/build... war
2. su -
3. cd $CATALINA_HOME/webapps
4. cp $MYCORESAMPLE_HOME/mycoresample.war
5. rctomcat restart

Nun sollten Sie auf die Beispielanwendung mit der URL http://localhost:8080/mycoresample zugreifen können. Testen Sie nun die Anwendung!

3.6.2.2 Websphere

In der gesonderten Dokumentation zur Installation des IBM Content Managers wurde ja bereits beschrieben, wie die Anwendung IBM WebSphere zu installieren ist. Diese soll als Servlet-Engine zur Anwendung kommen, wenn der IBM Content Manager 8 als Persitence-Layer verwendet wird. Die Konfiguration von WebSphere erfolgt via Web-Anwendung. Starten Sie dazu den Adminserver mittels

```
/usr/WebSphere/AppServer/bin/startServer.sh server1
```

Öffnen Sie nun eine Web-Browser mit der URL `http://<hostname>:9090/admin` und melden Sie sich an. Nun sind folgende Schritte durchzuführen:
1. (linke Seite) **Server Application Server**
2. (rechte Seite) **NEW**
3. (rechte Seite) Server Name **mycoresample NEXT**
4. (rechte Seite) **FINISH**
5. (linke Seite) **Applications Install New Applications**
6. (rechte Seite) **Server Path** Pfad zum File mycoresample.war eintagen **/mycoresample** im Feld Context Root eintragen **NEXT**
7. (rechte Seite) Preparing for application installation **NEXT**
8. (rechte Seite) Step 1 **NEXT**
9. (rechte Seite) Step 2 **NEXT**
10. (rechte Seite) Step 3 Auswählen **mycoresample** in der Checkbox dann auswählen der Zeile mit **server=mycoresample APPLY**
11. (rechte Seite) Step 3 **NEXT**
12. (rechte Seite) Step 4 **FINISH**
13. (linke Seite) **Server Application Server**
14. (rechte Seite) **mycoresample Process Definition Process Execution**

15.(rechte Seite) User auf **mcradmin** setzen
16.(rechte Seite) Group auf **mcr** setzen
17.(rechte Seite) **APPLY OK**
18.(rechte Seite) **Java Virtual Machine Classpath**
19.(rechte Seite) einfügen der Pfade
/home/db2inst1/sqllib/java12/db2java.zip:
/usr/lpp/cmb/lib/cmbsdk81.jar:
/usr/lpp/cmb/cmgmt
20.(rechte Seite) **APPLY OK**
21.(rechte Seite) oben auf den Text **save** klicken
22.(rechte Seite) **SAVE**
23.(linke Seite) **Environment Update Web Server Plugin OK**
24.**Logout**

Nun muss der Application Server gestartet werden:

```
/usr/WebSphere/AppServer/bin/startServer.sh mycoresample
```

Da die Anwendung als **mcradmin** ausgeführt wird, kommt es zu einem Schreibfehler in den Log-Files. Hier ist nun folgendes zu tun:

1. chown -R mcradmin:mcr /usr/WebSphere/AppServer/logs/mycoresample
2. chmod 666 /usr/WebSphere/AppServer/logs/activity.log
3. cd /usr/WebSphere/AppServer/temp/<hostname>
4. mkdir mycoresample
5. chown -R mcradmin:mcr mycoresample

Danach ist der Server nocheinmal zu stoppen und neu zu starten:

```
/usr/WebSphere/AppServer/bin/stopServer.sh mycoresample
/usr/WebSphere/AppServer/bin/startServer.sh mycoresample
```

Jetzt sollten Sie auf das MyCoRe-Sample unter der URL http://<hostname>/mycoresample zugreifen können.

4.Vom Sample zum eigenen Dokumenten-Server

4.1 Allgemeines

Nachdem das MyCoReSample bei Ihnen nun läuft und erste Erfahrungen damit gesammelt wurden, soll nun auf dieser Grundlage aufbauend ein produktiver Dokumenten-Server aufgesetzt werden. Hier gilt es, das MyCoRe Modell in eine praxisorientierte Anwendung umzusetzen. Sicher werden bei jedem Anwender weitere zusätzliche Anforderungen auftreten, welche innerhalb des Beispiels oder der nachfolgenden Ausführungen keine Beachtung fanden. Die MyCoRe-Gruppe ist gern bereit, allgemeingültige Ergänzungen in das Kern-Projekt mit aufzunehmen.

Für die Erstellung einer Dokument- und Multimedia-Server-Anwendung sind im groben die folgenden Schritte erforderlich. Modifizieren Sie das Beispiel schrittweise hin zu Ihrer eigenen Applikation. Prüfen Sie in Zwischenschritten immer, dass die gewünschte Funktionalität noch erhalten geblieben ist.

- Kopieren des MyCoReSamples (alles unter $MYCORESAMPLE_HOME) in einen gesonderten Verzeichniszweig (z. B. *~/docserv*. Es ist von Vorteil diesen Pfad auch in einer Environment Variable abzulegen (z. B. export DOCSERV_HOME=~/docserv).
- Legen Sie für Ihre Anwendung die Namen für die Datenbank-Tabellen, XML- und IFS-Stores fest und ändern Sie die Konfigurationsdateien *$DOCSERV_HOME/config/mycore.properties.private* entsprechend.
- Kopieren Sie sich die Privileg-, Group- und User-Daten aus *$DOCPORTAL_HOME/config/user und* modifizieren Sie die User-Daten entsprechend ihren Anforderungen. Entfernen Sie die nicht benötigten Anweisungen aus *$DOCSERV_HOME/build.xml,* so dass nur noch die von Ihnen gewünschten Benutzer geladen werden. Laden Sie das User-System.
- Erstellen Sie sich alle benötigten Klassifikationen (z. B. eine aller Ihrer Einrichtungen) und laden Sie diese.
- Erzeugen Sie sich einen Satz Testdaten (pro MCRObject-Typ mindestens eine Datei) und laden Sie diese mit dem Commandline Tools. Nun können Sie auf Commandline-Ebene schon mittels mycore.sh bzw. mycore.cmd Anfragen an das System stellen und erste Test durchführen.
- Legen Sie nun eine URL für ihren Server fest und setzen Sie einen Web-Server (WebSphere oder Apache/Tomcat) auf. Dieser Server sollte als eigenständige Einheit arbeiten und vom MyCoreSample entkoppelt sein. Am Besten ist, unter **mcradmin** eine Tomcat-Instanz auf einem freien Port anzulegen.
- Installiern Sie Ihre Anwendung im Web-Server und modifizieren Sie schrittweise die Präsentation nach Ihren Bedürfnissen.
- Testen und integrieren Sie die in diesem Kapitel beschriebenen weiterfführenden Funktionalitäten, welche nicht im MyCoReSample enthalten sind.

Habe Sie all die Schritte bewältigt, sollte Ihnen nun ein ansprechender Dokument-Server zur Verfügung stehen. Sollten Sie andere Applikationen, wie Sammlungen usw. aufbauen wollen, konsultieren Sie bitte auch das ProgrammerGuide, wo auf derartige MyCoRe-Erweiterungen näher

eingegangen wir.

4.2 Die Installation der freien XML:DB eXist

4.2.1 Allgemeines

eXist ist eine frei verfügbare XML:DB, welche die entsprechenden Interfaces implementiert. Für MyCoRe wurde ein auf diesen Schnittstellen basierendes Search-Backend implementiert. So ist die Nutzung von eXist direkt möglich. Für Produktionsumgebungen, welche auf Linux basieren ist es Sinnvoll, das im MyCoReSample verwendete JDOM-Search-Backend gegen das von eXist auszutauschen, da die JDOM-Applikation mit ca. 200 Dokumenten ihre Grenze erreicht hat. Wie Sie das backend austauchen können, ohne einen Datenverlust zu erleiden, soll dieses Kapitel zeigen.

4.2.2 Vorbedingungen

Für das folgende Szenario ist darauf zu achten, dass der Tomcat bzw. Websphere Server nicht auf die Ports 8080 und 8081 hören, da es ansonsten zu einem Konflikt mit der hier vorgestellten Installation kommt, weil der eXist servlet Container ebenfalls standardmäßig auf den port 8080 hört. Den Port für Tomact ändern sie in der *tomcatinstalldir/conf/server.xml*. Wir haben uns in der Praxis entschlossen, mit eXist auf Ports mit den Nummern 8090, 8091 ... auszuweichen.

4.2.3 Installation

1. Download der aktuellen Version von eXist[9]. Zum Einsatz kam Version eXist-1.0b2 .
2. Entpacken Sie die Distribution in ein entsprechendes Verzeichnis, z. B. unter */home/mcradmin* (eXist-installdir).
3. Entfernen Sie zur Nutzung des Stand-Alone-Servers den Kommentar aus der Zeile `uri=xmldb:exist://localhost:8081` im File *client.properties*
4. Kommentieren Sie folgende Zeile aus.
   `uri=xmldb:exist://localhost:8080/exist/xmlrpc`
   Dies Einstellung wird zum Beispiel für die direkte Einbindung in Tomcat verwendet, wenn man über xmlrpc auf eXist zugreifen möchte.
5. Ändern Sie ggf. die Portnummern.
6. Unter Linux müssen noch die Shell-Scripts ausführbar gemacht werden.
   chmod 755 <eXist-installdir>/bin/*.sh
7. Starten Sie nun den Server mit <eXist-installdir>/bin/server.sh bzw. <eXist-installdir>/bin/server.bat
8. Anschließend können Sie auch den Client mit <eXist-installdir>/bin/client.sh bzw. <eXist-installdir>/bin/client.bat starten.
9. Hier sollten Sie dem **admin**- und **guest**-User ein Password spendieren.

4.2.4 Konfigurationsänderungen in den MyCoRe-Teilen

Um das eXist-backend als aktives zu integrieren, müssen Sie ein paar Änderungen an Ihrer

9 http://exist-db.org

MyCoRe-Installation vornehmen. Diese sollen in der folgenden Liste Schritt für Schritt beschrieben sein.

1. Bearbeiten Sie die Datei *$MYCORE_HOME/bin/build.properties* und kommentieren Sie ganz unten die Nutzung der JDOM-Variante aus.
2. Entfernen Sie die Kommentare in den Zeilen für eXist (Linux oder Windows). Achten Sie darauf, dass die Einträge hinsichtlich der Verzeichnisnamen richtig sind.
3. Löschen Sie die Datei *$MYCORE_HOME/lib/mycore-for-jdom.jar*
4. Löschen Sie das Verzeichnis *$MYCORE_HOME/classes*
5. Bauen Sie ein neues *.jar-File mit bin/build.sh jar bzw bin/build.cmd jar
6. Ergänzen Sie die Datei *$DOCSERV/config/mycore.properties.private* im Abschnitt für eXist um folgende Zeilen und passen Sie die anderen Einträge entsprechend an

   MCR.persistence_xmldb_user=admin

   MCR.persistence_xmldb_passwd=??????? (Ihr Password)
7. Erzeugen Sie mit Hilfe des eXist-Client eine Collection mit dem Namen, der in der Konfiguration angegeben ist (in der Regel **mycore** oder für eine Dokumenten-server **docserv**)
8. Erzeigen Sie ein neues mycore.sh / mycore.cmd über den build-Aufruf (siehe oben)

4.2.5 Füllen des Search-Backends für eXist

Nachdem Sie nun das System für die Arbeit mit eXist vorbereitet haben, gilt es nun noch das backend mit Daten zu füllen. MyCore speichert alle Daten (XML-Dateien) in einer SQL-Tabelle als BLOB's. Dies schafft eine hohe Performance bei der Auslieferung der Dateien. Zur Suche werden die Daten dann aufbereitet und noch einmal in eine Suchmaschine gepackt, in diesem Fall ist das jetzt eXist. Es ist also möglich auf einfache Art, aus den SQL-Tabellen die Daten in den entsprechenden Search-Store wieder zu laden. Dies geht wie folgend:

1. $DOCSERV_HOME/bin/RepairExist.sh
2. Starten Sie nun eine Test-Suche, indem Sie das Script DocQuery.sh bzw DocQuery.cmd aktivieren.

Nun sollte Ihr System auf die Benutzung von eXist umgestellt sein und Sie können nun Tomcat starten um auch interaktiv zu tetsten.

4.3 Die Zusammenarbeit mit anderen MyCoReSample-Installationen

Das MyCoRe-System ist so konzipiert, dass hinsichtlich der Metadaten gleichartige Installationen miteinander arbeiten können und von einer gemeinsamen Oberfächer (Frontend) abgefragt werden können. Hierzu müssen die Remote-Instanzen definiert werden. Auch die eigene Installation kann über diesen Weg abgefragt werden. Voraussetzung ist die im Abschitt ’Erzeugen und Konfigurieren der Web-Anwendung’ beschriebene Installation eines Web Application Servers, welcher für die

Remote-Zugriffe via Servlets zuständig ist.

4.3.1 Die eigene Installation

Die Konfiguration für den Zugriff auf die eigene Installation finden Sie im File mycore.properties.private in dem Abschnitt MCR.remoteaccess_selfremote. Hier muss im Normalfall nur die Hostadresse und ggf. der Port geändert werden, alle anderen Angaben sollten übernommen werden können.

```
MCR.remoteaccess_selfremote_host=myhost.de
MCR.remoteaccess_selfremote_port=8080
```

Nun sollten Sie Ihre Anwendung auch über die Remote-Schiene abfragen können.

4.3.2 Weitere Server benachbarter Einrichtungen

Wenn Sie MyCoRe-Installationen anderer Community-Mitglieder integrieren wollen, ist folgendes zu tun:

1. Kopieren Sie den MCR.remoteaccess_selfremote... Abschnitt in der Konfiguration in einen Abschnitt MCR.remoteaccess_otherhost... für den neuen Serverzugang.
2. Tragen Sie die korrekten Netzzugänge ein.
3. Ergänzen sie den Hostalias in der Konfigurationszeile MCR.remoteaccess_hostaliases

Wenn Sie nun die Suchmaskenkonfiguration unter $DOCSERV/config/SearchMask... noch um die Zeilen für einen weiteren entfernten Rechner (z. B. otherhost) erweitern, ist das system in Ihre Suche integriert.

4.3.3 Standard-Server des MyCoRe-Projektes

Von den Entwicklern des MyCoRe-Projektes werden exemplarisch einige MyCoRe-Sample-Installationen bereitgehalten. Diese sind im Konfigurationsfile mycore.properties.remote notiert und sollten in der Regel verfügbar sein. Sie repräsentieren eine Auswahl der verschieden Persistence-Layer. Auch die Auswahl für die Suche in diesen Instanzen ist bereits in das Sample integriert und solle nach dem erfolgreichen Start der Web Applikation aktiv sein.

Alias	URL	Port	Standort
mcrLpzMS	http://mycoresamplelinux.dl.uni-leipzig.de/	8291	Uni Leipzig

Tabelle 5: Feste Test-Instanzen für das MyCoRe-Beispiel

4.4 Nutzung der OAI Schnittstelle

4.4.1 Grundlagen

Die Open Archives Initiative[10] hat 2001 ein offenes Protokoll für das Sammeln (Harvesting) von Metadaten vorgestellt. Dies geschah vor dem Hintergrund, dass gängige Suchmaschinen im WWW für die wissenschaftliche Nutzung wegen der i.d.R. unüberschaubaren Treffermenge und der fehlenden Qualität der angebotenen Treffer kaum nutzbar sind. Das **Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)** liegt mittlerweile in der Version 2.0 vor. Das OAI-PMH dient zur Kommunikation zwischen **Data Providern** und **Service Providern**. Unter einem Data Provider versteht man hierbei ein Archivierungssystem, dessen Metadaten von einem (oder mehreren) Service Provider(n) abgerufen werden, der diese als Basis zur Bildung von Mehrwertdiensten benutzt (z.B. der Suche über viele Archive gleichzeitig).

Zum besseren Verständnis der weiteren Dokumentation führe ich hier die wichtigsten Definitionen kurz an:

- Ein **Harvester** ist ein Client, der OAI-PMH Anfragen stellt. Ein Harvester wird von einem Service Provider betrieben, um Metadaten aus Repositories zu sammeln.
- Ein **Repository** ist ein über das Netzwerk zugänglicher Server, der OAI-PMH Anfragen verarbeiten kann, wie sie im Open Archives Initiative Protocol for Metadata Harvesting 2.0 vom 2002-06-14 beschrieben werden[11]. Ein Repository wird von einem Data Provider betrieben, um Harvestern den Zugriff auf Metadaten zu ermöglichen.

Der für MyCoRe und Miless implementierte OAI Data Provider ist zertifiziert und erfüllt den OAI-PMH 2.0 Standard.

4.4.2 Der OAI Data Provider

MyCoRe bietet ein extrem flexibles Klassifikations-/Kategoriensystem. Der MyCoRe OAI Data Provider benutzt eine frei definierbare Teilmenge dieser Klassifikationen zur Strukturierung der Metadaten gemäß der Definition von **Set**s in OAI 2.0. An den Harvester werden also nur Metadaten ausgeliefert, die in diesen Klassifikationen erfasst sind, wobei die Klassifikationen eine Set-Struktur erzeugen. Zur weiteren Einschränkung kann eine zusätzliche Klassifikation (restriction classification) angegeben werden, in der die Elemente klassifiziert sein müssen, die für den OAI Data Provider aber nicht strukturbildend ist.

Sollen weitere Daten über OAI zugänglich gemacht werden, so bietet der OAI Data Provider die Möglichkeit, unter verschiedenen Namen mehrere Servlet-Instanzen zu betreiben, wobei eine Instanz jeweils ein OAI-Repository darstellt.

4.4.3 Installation

Zur Einbindung des OAI Data Providers müssen Eintragungen in den Deployment Descriptor des Servletcontainers und in die mycore.properties erfolgen.

10 http://www.openarchives.org/
11 http://www.openarchives.org/OAI/openarchivesprotocol.html

4.4.4 Der Deployment Descriptor

Für jedes OAI-Repository muss eine Servlet-Instanz in den Deployment Descriptor nach folgendem Muster eingetragen werden:

```
<servlet id="OAIDataProvider">
    <servlet-name>
      OAIDataProvider
    </servlet-name>
    <servlet-class>
      org.mycore.services.oai.MCROAIDataProvider
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>
      OAIDataProvider
    </servlet-name>
    <url-pattern>
      /servlets/OAIDataProvider
    </url-pattern>
  </servlet-mapping>
```

Abbildung 1: Einbindung des OAI-Data-Providers in web.xml

4.4.5 Die mycore.properties.oai

Bei den einzurichtenden Properties ist zwischen *instanzunabhängigen* und *instanzabhängigen* Properties zu unterscheiden. Instanzunabhängige Properties sind hierbei für jedes angebotene OAI-Repository gültig, instanzabhängige Properties beziehen sich auf das jeweilige OAI-Repository.

4.4.6 Instanzunabhängige Properties

- MCR.oai.adminemail=admin@uni-irgendwo.de **(notwendig)** Der Administrator der OAI-Repositories.
- MCR.oai.resumptiontoken.dir=/mycore/temp **(notwendig)** Ein Verzeichnis, in welches der OAI Data Provider Informationen über Resumption Token ablegt.
- MCR.oai.resumptiontoken.timeout=48 **(notwendig)** Die Zeit (in Stunden), für die die Informationen über die Resumption Token nicht gelöscht werden. Da das Löschen nur erfolgt, wenn auf ein OAI-Repository zugegriffen wird, können die Dateien evtl. auch länger aufgehoben werden.
- MCR.oai.maxreturns=50 **(notwendig)** Die maximale Länge von Ergebnislisten, die an einen Harvester zurückgegeben werden. Überschreitet eine Ergebnisliste diesen Wert, so wird ein Resumption Token angelegt.
- MCR.oai.queryservice=org.mycore.services.oai.MCROAIQueryService **(notwendig)** Die Klasse, die für das Archiv das Query-Interface implementiert. Für Miless ist dies `oai`OAIService .

- MCR.oai.metadata.transformer.oai_dc=MyCoReOAI-mycore2dc.xsl **(notwendig)** Das Stylesheet, das die Transformation aus dem im Archiv benutzten Metadatenschema in das für OAI benutzte OAI Dublin Core Metadatenschema durchführt. Wenn sich das im Archiv benutzte Metadatenschema ändert, muss dieses Stylesheet angepasst werden. Optional können weitere Stylesheets angegeben werden, die einen Harvester mit anderen Metadatenformaten versorgen, z.B. MCR.oai.metadata.transformer.rfc1806=MyCoReOAI-mycore2rfc.xsl. Diese Stylesheets benutzen als Eingabe das Ergebnis des ersten Stylesheets.

4.4.7 Instanzabhängige Properties

Bei instanzabhängigen Properties wird der im Deployment Descriptor verwendete Servletname zur Unterscheidung für die einzelnen Repositories verwendet.

- MCR.oai.repositoryname.OAIDataProvider=Test-Repository **(notwendig)** Der Name des OAI-Repositories.
- MCR.oai.repositoryidentifier.OAIDataProvider=mycore.de **(notwendig)** Der Identifier des OAI-Repositories (wird vom Harvester abgefragt).
- MCR.oai.setscheme.OAIDataProvider=DocPortal_class_00000004 DocPortal_class_00000005 **(notwendig)** Die MyCoRe-Klassifikation, die zur Bildung der Struktur des OAI-Repositories verwendet wird.
- MCR.oai.restriction.classification.OAIDataProvider=DocPortal_class_00000006 **(optional)** Die MyCoRe-Klassifikation, die zur Beschränkung der Suche verwendet wird.
- MCR.oai.restriction.category.OAIDataProvider=dk01 **(optional)** Die MyCoRe-Kategorie, die zur Beschränkung der Suche verwendet wird.
- MCR.oai.friends.OAIDataProvider=miami.uni-muenster.de/servlets/OAIDataProvider **(optional)** Unter dieser Property können weitere (bekannte und zertifizierte) OAI-Repositories angegeben werden, um den Harvestern die Suche nach weiteren Datenquellen zu vereinfachen.

4.4.8 Test

Um zu testen, ob das eigene OAI-Repository funktioniert, kann man sich des Tools bedienen, das von der *Open Archives Initiative* unter `http://www.openarchives.org` zur Verfügung gestellt wird. Unter dem Menüpunkt **Tools** wird der **OAI Repository Explorer** angeboten.

4.4.9 Zertifizierung und Registrierung

Ebenfalls auf der oben angegebenen Website findet sich unter dem Menüpunkt **Community** der Eintrag **Register as a data provider**. Dort kann man anfordern, das eigene Repository zu zertifizieren und zu registrieren. Die Antwort wird an die in den Properties eingetragene EMail-Adresse geschickt.

5. Weiterführende Informationen zum Aufbau von MyCoRe-Anwendungen

5.1 XML-Syntax des Datenmodells

In diesem Abschnitt wird der Syntax der einzelnen XML-Daten-Dateien und der MyCoRe-Standard-Datentypen näher beschrieben. Die Kenntnis des Syntax benötigen Sie um eigene Datensätze für Ihren Dokumenten-Server zu erstellen. Eine umfassende Beschreibung der zugehörigen Klassen finden Sie im Programmer Guide. In den folgenden Abschnitten wird lediglich auf die XML-Syntax der Daten eingegangen.

5.1.1 Das Klassifikationen-Datenmodell

Wie bereits erwähnt dienen Klassifikationen der einheitlichen Gliederung bestimmter Fakten. Sie sorgen dafür, dass eine einheitliche Schreibweise für bestimmte Begriffe verwendet wird. Diese Einzelbegriffe werden als Kategorien bezeichnet. Innerhalb einer Kategorie kann der Begriff in verschiedenen Sprachen aufgezeichnet sein. Die eindeutige Zuordnung zu einer Kategorie erfolgt über einen Bezeichner. Dieser besteht aus der Klassifikations- und Kategorie-ID und muss eindeutig sein.

Klassifikationen werden im MyCore-Sample als extra XML-Datei erstellt, in die Anwendung importiert und in Form einer Datenbank gespeichert. Dies ist für den Nutzer transparent und erfolgt mittels Schnittstellen. Der Zugriff auf die Daten erfolgt dann durch den oben genannten Bezeichner. Die Klassifikations-ID ist eine MCRObjectID mit dem Typ class. Die Kategorie-ID ist dagegen frei wählbar. Sie darf mehrstufig ein, jede Stufe spiegelt eine Hierarchieebene wieder. Die Stufen in der ID werden mit einem Punkt voneinander getrennt, 'Uni.URZ'. Das wiederum gestattet eine Abfrage nach allen untergeordneten Stufen bzw. Sub-Kategorien wie 'Uni.*'. **Achtung, sollten Sie Zahlen als Kategorie-ID's mit verwenden, so planen Sie entsprechende Vornullen ein, andernfalls wird das Suchergebnis fehlerhaft! Weiterhin ist es sehr zu empfehlen, dieser Zahlenfolge einen Buchstaben voranzusetzen, damit die ID nicht als Zahl interpretiert wird (z. B. beim Content Manager 8.2)**.

Im ID Attribut einer category ist der eindeutige Bezeichner anzugeben. Das darunter befindliche label Tag bietet die Möglichkeit, eine Kurzbezeichnung anzugeben. Mehrsprachige Ausführungen sind erlaubt. Dasselbe gilt für das Tag description. Beide Werte werden als Strings aufgefasst. Eine category kann wiederum category Tags beinhalten.

Abbildung 2: XML-Syntax eines Klassifikations-Objektes

```
<?xml version="1.0" cncoding="ISO-8859-1" ?>
 <mycoreclass
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="MCRClassification.xsd"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  ID="..."
  >
  <label xml:lang="..." text="..." description="..."/>
  ...
  <categories>
   <category ID="...">
    <label xml:lang="..." text="..." description="..."/>
    ...
    <category ID="...">
     <label xml:lang="..." text="..." description="..."/>
     ...
    </category>
    <category ID="...">
     <label xml:lang="..." text="..." description="..."/>
     ...
    </category>
   </category>
   <category ID="...">
    <label xml:lang="..." text="..." description="..."/>
    ...
   </category>
  </categories>
 </mycoreclass>
```

5.1.2 Das Metadatenmodell

Die zu speichernden Daten des Beispiels teilen sich in unserem Modell in Metadaten und digitale Objekte. Dies gilt auch für die vom Anwender entwickelten Applikationen. Digitale Objekte sind Gegenstand des Abschnitts **'IFS und Content Store'**. Unter Metadaten verstehen wir in MyCoRe alle beschreibenden Daten des Objektes, die extern hinzugefügt, separat gespeichert und gesucht werden können. Dem gegenüber stehen Daten welche die digitalen Objekte selbst mitbringen. In diesem Abschnitt werden nur erstere behandelt.

Um die Metadaten besser auf unterschiedlichen Datenspeichern ablegen zu können, wurde ein System von XML-Strukturen entwickelt, das es gestattet, neben den eigentlichen Daten wie Titel, Autor usw. auch Struktur- und Service-Informationen mit abzulegen. Die eigentlichen Nutzerdaten sind wiederum typisiert, was deren speicherunabhängige Aufzeichnung erheblich vereinfacht. Es

steht dem Entwickler einer Anwendung jedoch frei, hier bei Bedarf weitere hinzuzufügen. Im Folgenden soll nun der Aufbau der Metadatenobjekte im Detail beschrieben werden. Zum Verständnis des Samples sei hier auch auf den vorigen Abschnitt verwiesen. Die Metadaten werden komplett in XML erfasst und verarbeitet. Für die Grundstrukturen und Standardmetadatentypen werden seitens MyCoRe bereits XMLSchema-Dateien mitgeliefert.

5.1.2.1 XML-Syntax eines Metadatenobjektes

```
<?xml version="1.0" cncoding="ISO-8859-1" ?>
 <mycoreobject
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="....xsd"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  ID="..."
  label="..."
  >
  <structure>
   ...
  </structure>
  <metadata xml:lang="de">
   ...
  </metadata>
  <service>
   ...
  </service>
 </mycoreobject>
```

Abbildung 3: XML-Syntax eines Metadaten-Objektes

- Für **xsi:noNamespaceSchemaLocation** ist das entsprechende XMLSchema-File des Metadatentyps anzugeben (document.xsd)
- Die **ID** ist die eindeutige MCRObjectID.
- Der **label** ist ein kurzer Text-String, der bei administrativen Arbeiten an der Datenbasis das Identifizieren einzelner Datensätze erleichtern soll. Er kann maximal 256 Zeichen lang sein.
- Innerhalb der XML-Datenstruktur gibt es die Abschnitte **structure**, **metadata** und **service** zur Trennung von Struktur-, Beschreibungs- und Wartungsdaten. Diese Tag-Namen sind reserviert und **dürfen NICHT anderweitig verwendet werden!**

5.1.2.2 XML-Synatx des XML-Knotens structure

Im XML-Knoten **structure** sind alle Informationen über die Beziehung des Metadatenobjektes zu anderen Objekten abgelegt. Es werden derzeit die folgenden XML-Daten unter diesem Knoten

abgelegt. Die Tag-Namen parents/parent, children/child und derobjects/derobject sind reserviert und **dürfen NICHT anderweitig verwendet werden!** Alle Sub-Knoten haben einen Aufbau wie für MCRMetaLinkID beschieben.

In **parents** wird ein Link zu einem Elternobjekt gespeichert, sofern das referenzierende Objekt Eltern hat. Ob dies der Fall ist, bestimmt die Anwendung. Das Tag dient der Gestaltung von Vererbungsbäumen und kann durch den Anweder festgelegt werden. Siehe auch 'Programmers Guide', Abschnitt Vererbung. Die Werte für xlink:title und xlink:label werden beim Laden der Daten automatisch ergänzt.

Die Informationen über die **children** hingegen werden durch das MyCoRe-System beim Laden der Daten **automatisch** erzeugt und **dürfen nicht per Hand geändert werden**, da sonst das Gesamtsystem nicht mehr konsistent ist. Werden die Metadaten eines Kindes oder eines Baumes von Kindern gelöscht, so wird in diesem Teil des XML-Knotens der Eintrag durch die Software entfernt.

Dasselbe gilt auch für den XML-Unterknoten derobjects. In diesem Bereich werden alle Verweise auf die an das Metadatenobjekt angehängten digitalen Objekte gespeichert. Jeder Eintrag verweist mittels einer Referenz auf ein Datenobjekt vom Typ mycorederivate, wie es im nachfolgenden Abschnitt 'IFS und Content Store' näher erläutert ist.

```
<structure>
 <parents class="MCRMetaLinkID">
  <parent xlink:type="locator" xlink:href="...mcr_id..." />
 </parents>
 <children class="MCRMetaLinkID">
  <child xlink:type="locator" xlink:href="...mcr_id..." xlink:label="..." xlink:title="..." />
  ...
 </children>
 <derobjects class="MCRMetaLinkID">
  <derobject xlink:type="locator" xlink:href="...mcr_id..." xlink:label="..." xlink:title="..." />
  ...
 </derobjects>
</structure>
```

Abbildung 4: XML-Syntax eines structure XML-Knotens

5.1.2.3 XML-Syntax des XML-Knotens metadata

Der Abschnitt **metadata** des MyCoRe-Metadatenobjektes nimmt alle Beschreibungsdaten des

eigentlichen Datenmodells auf. Diese werden ihrerseits in vordefinierten Datentyp-Strukturen mit festgelegter Syntax abgelegt. Die Liste der Einzelelemente und die Reihenfolge der Typen ist dabei quasi beliebig in Anordnung und Länge. Wichtig ist nur, dass alle Datentypen bestimmte gemeinsame Eigenschaften haben. Es ist auch jederzeit möglich, weitere Typen den Projekten der Anwender hinzuzufügen (siehe dazu das Dokument MyCoRe Programmer Guide).

Die Metadaten bestehen aus einer Ansammlung von Informationen rund um das multimediale Objekt. Vorrangig wird dieser Teil in der Suche abgefragt. Jedes Metadatum (auch Metadaten-Tag) enthält im class Attribut den Namen des MCRMeta-Typs bzw. der gleichnamigen MCRMeta-Java Klasse. Daneben gibt es noch ein Attribut heritable, in dem festgelegt wird, ob diese Metadaten vererbbar sein sollen. Weiterhin können noch die Attribute papasearch für die Einbindung in die parametrische Suche und textsearch für die Volltext-Suche über das gesamte Metadatenobjekt angegeben werden. Es sind jeweils die boolschen Werte true oder false möglich. Die mit der Vererbung verbundenen Mechanismen sind in dieser Dokumentation weiter hinten beschrieben.

Für MyCoRe wurden einige Basismetadatentypen festgelegt, mit denen die Mehrzahl der bisher in Betracht kommenden Anwendungen gelöst werden können. Die einzelnen Daten treten dabei als Liste auf, in denen mehrere Elemente des gleichen Typs erscheinen können, beispielsweise ein Titel in verschiedenen Sprachen. Jedes Listenelement hat wiederum per Default ein **type** Attribut und eine gemäß W3C spezifizierte Sprache im Attribut **xml:lang**. Die Angabe der Sprache im Tag metadata ist für alle eingschlossenen Metadatentypen der Default-Wert. Die Liste der aktuell unterstützten Sprach-Codes entnehmen Sie bitte der Java-Quelldatei

~/mycore/sources/org/mycore/common/MCRDefaults.java

Für interne Zwecke wurde ein weiteres Attribut inherited eingeführt. Dieses ist NICHT durch den Anwender zu verändern! Es wird gesetzt, wenn das betreffende Metadatum von einem Elternteil geerbt wurde (siehe Vererbung). Diese Information ist für die Datenpräsentation sehr hilfreich.

Für das MyCoRe-Beispiel mit einem Dublin Core Datenmodell werden bereits einige Metadatentypen verwendet, welche dem MyCoRe-Kern beigefügt sind. Die Syntax der einzelnen Typen wird in den nachfolgenden Absätzen genau beschrieben.

```
<metadata xml:lang="...">
  <... class="MCRMeta..." heritabel="..." parasearch="..." textsearch="...">
    ...
  </...>
  ...
 </metadata>
```

Abbildung 5: XML-Syntax eines metadata XML-Knotens

5.1.2.4 MyCoRe Metadaten-Basistypen

In MyCoRe gibt es eine Reihe von vordefinierten XML-Datenstrukturen zur Abbildung bestimmter mehr oder minder komplexer Daten. Diese Strukturen bilden die MyCoRe-Datentypen, welche von der Dateneingabe bis hin zur Validierung und Datenpräsentation für einen einheitlichen Umgang mit den Daten sorgen. Dabei ist zwischen einfachen, recht atomaren Typen und anwendungsspezifischen komplexen

Typen zu unterscheiden. Eine Auflistung finden Sie in nachfolgender Tabelle.

einfache Typen	**komplexe Typen**
MCRMetaBoolean	MCRMetaAddress
MCRMetaClassification	MCRMetaCorporation
MCRMetaDate	MCRMetaPerson
MCRMetsLangText	MCRMetaIFS
MCRMetaLink	
MCRMetaLinkID	
MCRMetaNumber	
MCRMetaXML	

Tabelle 6: MyCoRe-Datentypen

5.1.2.4.1 XML-Syntax des Metadaten-Basistyps MCRMetaAddress

Der Basistyp MCRMetaAddress beinhaltet eine Liste von postalischen Anschriften in der Ausprägung eines XML-Abschnittes. Dabei wird berücksichtigt, dass die Anschrift in verschiedenen Sprachen und in international gängigen Formen gespeichert werden soll. Die einzelnen Subtags sind dabei selbsterklärend. Die Angaben zu **type** und **xml:lang** sind optional, ebenso die unter subtag liegenden Tags, jedoch muss mindestens eines ausgefüllt sein. Alle Werte werden als Text betrachtet. Das optionale Attribut textsearch hat keinen Effekt bei diesem Typ.

```
<tag class="MCRMetaAddress" heritable="..." parasearch="...">
  <subtag type="..." xml:lang="...">
   <country>...</country>
   <state>...</state>
   <zipcode>...</zipcode>
   <city>...</city>
   <street>...</street>
   <number>...</number>
  </subtab>
  ...
 </tag>
```

Abbildung 6: XML-Syntax des Metadaten-Basistyps MCRMetaAddress

Abbildung 7: Beispiel des Metadaten-Basistyps MCRMetaAddress

```
<addresses class="MCRMetaAddress" heritable="false" parasearch="true">
 <address type="Work" xml:lang="de">
  <country>Deutschland</country>
  <state>Sachsen</state>
  <zipcode>04109</zipcode>
  <city>Leipzig</city>
  <street>Augustuspaltz</street>
  <number>10/11</number>
 </address>
 ...
</addresses>
```

5.1.2.4.2 XML-Syntax des Metadaten-Basistyps MCRMetaBoolean

Der Basistyp MCRMetaBoolean beinhaltet eine Liste von Wahrheitswerten mit zugehörigen **type** Attributen. Das optionale Attribut **textsearch** hat keinen Effekt bei diesem Typ. Folgende Werte sind zulässig:

- für **true** - 'true', 'yes', 'wahr' und 'ja'
- für **false** - 'false', 'no', 'falsch' und 'nein'

```
<tag class="MCRMetaBoolean" heritable="..." parasearch="...">
  <subtag type="..." xml:lang="...">
   ...
  </subtab>
  ...
 </tag>
```

Abbildung 8: XML-Syntax des Metadaten-Basistyps MCRMetaBoolean

```
<publishes class="MCRMetaBoolean" heritable="true" parasearch="true">
  <publish type="Ausgabe_1" xml:lang="de">ja</publish>
  <publish type="Ausgabe_2" xml:lang="de">nein</publish>
  ...
 </publishes>
```

Abbildung 9: Beispiel des Metadaten-Basistyps MCRMetaBoolean

5.1.2.4.3 XML-Syntax des Metadaten-Basistyps MCRMetaClassification

Der Basistyp MCRMetaClassification dient der Einbindung von Klassifikationen[12] und deren

12siehe voriges Kapitel

Kategorien in die Metadaten. Beide Identifizierer zusammen beschreiben einen Kategorieeintrag vollständig. Dabei ist für die categid eine, ggf. mehrstufige, Kategorie-ID einzutragen. Die classid muss vom Typ MCRObjectID sein. Das optionale Attribut textsearch hat keinen Effekt bei diesem Typ.

```
<tag class="MCRMetaClassification" heritable="..." parasearch="...">
  <subtag classid="..." categid="..."/>
  ...
 </tag>
```

Abbildung 10: XML-Syntax des Metadaten-Basistyps MCRMetaClassification

```
<origins class="MCRMetaClassification" heritable="false" parasearch="true">
  <origin classid="MyCoReDemoDC_class_1" categid="Unis.Leipzig.URZ"/>
  ...
</origins>
```

Abbildung 11: Beispiel des Metadaten-Basistyps MCRMetaClassification

5.1.2.4.4 XML-Syntax des Metadatentyps MCRMetaCorporation

Der Basistyp MCRMetaCorporation beinhaltet eine Liste von Namen einer Firma oder Einrichtung. Dabei soll berücksichtigt werden, dass die Name in verschiedenen Sprachen und in international gängigen Formen gespeichert werden sollen. Über das Attribut type ist eine Differenzierung der verschiedenen Namen (Abteilungen, Institute, Kurznamen usw.) möglich. name beinhaltet den vollständigen Namen, nickname das Pseudonym, parent den Namen der übergeordneten Einrichtung und property den rechtlichen Stand, GmbH. Das optionale Attribut textsearch hat keinen Effekt bei diesem Typ.

```
<tag class="MCRMetaCorporation" heritable="..." parasearch="...">
  <subtag type="..." xml:lang="...">
   <name>...</name>
   <nickname>...</nickname>
   <parent>...</parent>
   <property>...</property>
  </subtab>
  ...
 </tag>
```

Abbildung 12: XML-Syntax des Metadaten-Basistyps MCRMetaCorporation

```
<firmas class="MCRMetaCorporation" heritable="true" parasearch="true">
  <firma type="Uni" xml:lang="de">
   <name>Universität Leipzig</name>
   <nickname>Uni Lpz</nickname>
   <property>Universität</property>
  </firma>
  <firma type="URZ" xml:lang="de">
   <name>Universitätsrechenzentrum</name>
   <nickname>URZ</nickname>
   <parent>Universität Leipzig</parent>
   <property>Einrichtung</property>
  ...
 </firmas>
```

Abbildung 13: Beispiel des Metadaten-Basistyps MCRMetaCorporation

5.1.2.4.5 XML-Syntax des Metadaten-Basistyps MCRMetaDate

Der Basistyp MCRMetaDate beschreibt eine Liste von Datumsangaben welche zusätzlich mit einem type Attribut versehen werden können. Das Darstellungsformat muss der angegebenen Sprache oder der ISO 8601 Notation folgen. Innerhalb von MyCoRe weden dann alle Datumsangaben in das ISO 8601 Format umgewandelt. Das optionale Attribut textsearch hat keinen Effekt bei diesem Typ.

```
<tag class="MCRMetaDate" heritable="..." parasearch="...">
  <subtag type="..." xml:lang="...">
   ...
  </subtab>
  ...
</tag>
```

Abbildung 14: XML-Syntax des Metadaten-Basistyps MCRMetaDate

```
<dates class="MCRMetaDate" heritable="false" parasearch="true">
  <date type="heute" xml:lang="de">15.10.2003</date>
  <date type="morgen" xml:lang="us">2003/16/10</date>
   ...
  </date>
  ...
 </dates>
```

Abbildung 15: Beispiel des Metadaten-Basistyps MCRMetaDate

5.1.2.4.6 XML-Syntax des Metadatentyps MCRMetaLangText

Der Basistyp MCRMetaLangText dient der Speicherung einer Liste von Textabschnitten mit zugehöriger Sprachangabe. Das Attribut **textsearch** bewirkt, dass alle Text-Values in einen gemeinsamen Textindex des Metadatenobjektes abgelegt werden. Über das **form** Attribut kann noch spezifizier werden, in welcher Form der Text geschrieben ist.

```
<tag class="MCRMetaLangText" heritable="..." parasearch="..."
textsearch="...">
  <subtag type="..." xml:lang="..." form="...">
   ...
  </subtab>
  ...
</tag>
```

Abbildung 16: XML-Syntax des Metadaten-Basistyps MCRMetaLangText

```
<titles class="MCRMetaLangText" heritable="true" parasearch="true" textsearch="true">
 <title type="maintitle" xml:lang="de" form="plain">
  Mein Leben als MyCoRe-Entwickler
 </title>
 ...
</titles>
```

Abbildung 17: Beispiel des Metadaten-Basistyps MCRMetaLangText

5.1.2.4.7 XML-Syntax der Metadatentypen MCRMetaLink und MCRMetaLinkID

Der Basistyp MCRMetaLink wurde geschaffen, um eine Verküpfung verschiedener MyCoRe-Objekte untereinander zu realisieren. Ausserden können hier genauso Verweise auf beliebige externe Referenzen abgelegt werden. Der Typ MCRMetaLink ist eine Implementation des W3C XLink Standards[13] . Auf dieser Basis enthält der MyCoRe-Metadatentyp zwei Arten von Links - eine Referenz und einen bidirektionalen Link. Bei beiden werden jedoch in MCRMetaLink nicht alle Möglichkeiten der XLink Spezifikation ausgeschöpft, da dies für die in MyCoRe benötigten Funktionalitäten nicht erforderlich ist.

Im Referenztyp ist das Attribut **xlink:type='locator'** immer anzugeben. Die eigentliche Referenz wird im **xlink:href** Attribut notiert. Dabei kann die Referenz eine URL oder eine MCRObjectID sein. Daneben können noch weitere Informationen im **xlink:label** angegeben werden, die Rolle einer verlinkten Person. Der Referenztyp kommt im MyCoRe-Sample bei der Verlinkung von Dokumenten und Personen zum Einsatz. Um den Update-Aufwand in Grenzen zu halten, wurde die genannte Verbindung als Referenz konzipiert. Somit weiß das referenzierte Objekt im Sample nichts über den Verweis.

13siehe 'XLM Linking Language (XLink) Version 1.0'

Alternativ dazu besteht die Möglichkeit eines bidirektionalen Links. Dieser wird sowohl in der Link-Quelle wie auch im Link-Ziel eingetragen. Der Typ ist in diesem Fall **xlink:type='arc'**. Weiterhin sind die Attribute **xlink:from** und **xlink:to** erforderlich. Optional kann noch ein Titel in xlink:title mitgegeben werden. Das optionale Attribut textsearch hat keinen Effekt bei diesem Typ.

Der Basistyp MCRMetaLinkID entspricht im Aufbau dem MCRMetaLink. Der einzige Unterschied ist, dass die Attribute **xlink:href**, **xlink:from** und **xlink:to** nur mit MCRObjectID's belegt werden dürfen.

```
<tag class="MCRMetaLink" heritable="..." parasearch="...">
  <subtag xlink:type="locator" xlink:href="..." xlink:label="..."
xlink:title="..."\>
  <subtag xlink:type="arc" xlink:from="..." xlink:to="..." xlink:title="..."\>
  ...
</tag>
```

Abbildung 18: XML-Syntax des Metadaten-Basistyps MCRMetaLink

```
<urls class="MCRMetaLink" heritable="false" parasearch="...">
  <url xlink:type="locator" xlink:href="http://www.zoje.de" xlink:label="ZOJE"
xlink:title="Eine externe URL"\>
  <url xlink:type="arc" xlink:from="mcr_object_id_1"
xlink:to="mcr_object_id_2" xlink:title="Link zwischen Objekten"\>
  ...
 </urls>
```

Abbildung 19: Beispiel des Metadaten-Basistyps MCRMetaLink

5.1.2.4.8 XML-Syntax des Metadatentyps MCRMetaNumber

Der Basistyp MCRMetaNumber ermöglicht das Speichern und Suchen von Zahlenwerten. Die Zahlendarstellung kann je nach Sprache, d. h. im Deutschen mit Komma und im Englischen mit Punkt, angegeben werden. Weiterhin sind die zusätzlichen Attribute **dimension** und **measurement** möglich. Beide Attribute sind optional, ebenso wie das Default-Attribut **type**. Das optionale Attribut textsearch hat keinen Effekt bei diesem Typ.

```
<tag class="MCRMetaNumber" heritable="..." parasearch="...">
  <subtag xml:lang="..." dimension="..." mesurement="...">
   ...
  </subtab>
  ...
</tag>
```

Abbildung 20: XML-Syntax des Metadaten-Basistyps MCRMetaNumber

```
<masse class="MCRMetaNumber" heritable="false" parasearch="true">
  <mass xml:lang="de" dimension="Breite" mesurement="cm">
   12,1
  </mass>
  <mass xml:lang="en" type="neu" dimension="width" mesurement="ft">
   12.2
  </mass>
  ...
 </masse>
```

Abbildung 21: Beispiel des Metadaten-Basistyps MCRMetaNumber

5.1.2.4.9 XML-Syntax des Metadatentyps MCRMetaPerson

Der Basistyp MCRMetaPerson beinhaltet eine Liste von Namen für natürliche Personen. Dabei wird berücksichtigt, dass die Namen in verschiedenen Sprachen und international gängigen Formen auftreten können. Das Attribut **type** dient der Differenzierung der verschiedenen Namen einer Person, Geburtsname, Synonym, Kosename usw. **firstname** repräsentiert den/die Vornamen, **callname** den Rufnamen, **surname** den Familiennamen, **academic** den akademischen Titel und **peerage** den Adelstitel. Das optionale Attribut **textsearch** hat den Effekt, dass alle textlichen Werte des Namens in das allgemeine Feld zur Textsuche des Metadatenobjektes übernommen werden.

```
<tag class="MCRMetaPerson" heritable="..." parasearch="...">
  <subtag type="..." xml:lang="..">
   <firstname>...</firstname>
   <callname>...</callname>
   <surname>...<surname>
   <academic>...</academic>
   <peerage>...</peerage>
  </subtag>
  ...
</tag>
```

Abbildung 22: XML-Syntax des Metadaten-Basistyps MCRMetaPerson

Abbildung 23: Beispiel des Metadaten-Basistyps MCRMetaPerson

```
<tag class="MCRMetaPerson" heritable="true" parasearch="false">
  <subtag type="geburtsname" xml:lang="de">
   <firstname>Lisa Marie</firstname>
   <callname>Lisa</callname>
   <surname>Schnell<surname>
  </subtag>
  <subtag type="familienname" xml:lang="de">
   <firstname>Lisa Marie</firstname>
   <callname>Lisa</callname>
   <surname>Schmidt<surname>
   <academic>Dr. phil.</academic>
   <peerage>von</peerage>
  </subtag>
  ...
</tag>
```

5.1.2.4.10 XML-Syntax des Metadatentyps MCRMetaXML

Der Basistyp MCRMetaXML wurde zusätzlich als Container für einen beliebigen XML-Baum in das Projekt integriert. Dieser wird in den Textknoten des Subtags gestellt und kann dort theoretisch beliebig groß sein. Achten Sie aber darauf, dass entsprechend vies Speicherplatz in dem XML-SQL-Store vorgesehen wird. Die Tag-Attribute **parasearch** und **textsearch** haben keine Wirkung.

```
<tag class="MCRMetaXML" heritable="..." parasearch="...">
  <subtag type="..." >
   ...
  </subtag>
  ...
</tag>
```

Abbildung 24: XML-Syntax des Metadaten-Basistyps MCRMetaXML

```
<tag class="MCRMetaPerson" heritable="true" parasearch="false">
  <subtag type="tab1" >
   <table width="100"><tr><td>Col A</td><td>Col B</td></tr></table>
  </subtag>
  ...
 </tag>
```

Abbildung 25: Beispiel des Metadaten-Basistyps MCRMetaXML

5.1.2.5 XML-Syntax des XML-Knotens service

Für die Einrichtung eines Workflow und um die Wartung großer Datenmengen zu vereinfachen, wurde der XML-Abschnitt service in das Metadatenobjekt integriert. Hier sind Informationen wie Datumsangaben und Flags für die Verarbeitung im Batch-Betrieb enthalten. **Achtung, die Tag-Namen sind fest vorgegeben und dürfen nicht anderweitig verwendet werden!**

Die Datumsangaben servdates verhalten sich analog zu denen in MCRMetaDate. Folgende Möglichkeiten für das Attribut type sind vorgesehen. Weitere Typen sind jedoch integrierbar.

- **acceptdate** - Datum aus dem Dublin Core, kann frei verwendet werden.
- **createdate** - Das Erzeugungsdatum des Objektes, dieser Wert wird **automatisch** beim Anlegen des Objektes erzeugt und **bleibt immer erhalten**!
- **modifydate** - Das Datum des letzten Update, dieser Wert wird **automatisch** beim Update des Objektes erzeugt und **bleibt immer erhalten**!
- **submitdate** - Datum aus dem Dublin Core, kann frei verwendet werden.
- **validfromdate** - Datum aus dem Dublin Core, kann frei verwendet werden.
- **validtodate** - Datum aus dem Dublin Core, kann frei verwendet werden.

Im servflags Teil können kurze Texte untergebracht werden. Die Anzahl der servflag Elemente ist **theoretisch** unbegrenzt.

```
<service>
  <servdates class="MCRMetaDate">
   <servdate type="...">...</servdate>
   ...
  </servdates>
  <servflags class="MCRMetaLangText">
   <servflag>...</servflag>
   ...
  </servflag>
</service>
```

Abbildung 26: XML-Syntax des service XML-Konotens

5.1.3 Das Speichermodell für die Multimediadaten (IFS)

Im bisherigen Verlauf dieses Kapitels wurden nur die beschreibenden Daten des multimedialen Objektes erläutert. Dieser Abschnitt beschäftigt sich damit, wie die eigentlichen Objekte dem Gesamtsystem hinzugefügt werden können. Im MyCoRe Projekt wurde zur Ablage der digitalen Objekte das Konzept des **IFS** entwickelt. Hier ist es möglich, über spezielle Konfigurationen festzulegen, in welchen Speicher (Store) die einzelnen Dateien gespeichert werden sollen.

Das Laden von Objekten erfolgt mittels einer Metadaten-Datei, welche alle Informationen über die zu speichernde(n) Datei(en) und ihre Beziehung(en) zu den Metadaten enthält. Die zu speichernden multimedialen Objekte werden im Weiteren als Derivate, also Abkömmlinge, bezeichnet, da ein Objekt in mehreren Formen, Grafikformaten, auftreten kann. Die Struktur der XML-Datei für

Derivate ist fest vorgegeben, alle Felder, die nutzerseitig geändert werden können, sind unten beschrieben.

```
<?xml version="1.0" cncoding="ISO-8859-1" ?>
 <mycorederivate
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="....xsd"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  ID="..."
  label="..."
  >
  <derivate>
   <linkmetas class="MCRMetaLinkID">
    <linkmeta xlink:type="locator" xlink:href="..." />
   </linkmetas>
   <internals class="MCRMetaIFS">
    <internal
     sourcepath="..."
     maindoc="..."
     />
   </internals>
  </derivate>
  <service>
   ...
  </service>
</mycoreobject>
```

Abbildung 27: XML-Syntax des Derivate-Datenmodells

- Für **xsi:noNamespaceSchemaLocation** ist die entsprechende XMLSchema-Datei anzugeben (Derivate.xsd)
- Die **ID** ist die eindeutige MCRObjectID.
- Der **label** ist ein kurzer Text-String, der bei administrativen Arbeiten an der Datenbasis das Identifizieren einzelner Datensätze erleichtern soll. Er kann maximal 256 Zeichen lang sein.
- Die Referenz in **linkmeta** ist die MCRObjectID des Metadatensatzes, an den das/die Objekte angehängt werden sollen.
- Das Attribut **sourcepath** enthält die Pfadangabe zu einer Datei oder zu einem Verzeichnis, welches als Quelle dienen soll. Aus diesen Dateien kann nun eine Datei ausgewählt werden, welche den Einstiegspunkt bei HTML-Seiten darstellen soll. Bei einzelnen Bildern ist hier noch einmal der Dateiname anzugeben. Ist nichts angegeben, so wird versucht Dateien wie index.html usw. zu finden.

5.1.4

5.2 Die Verwendung der Kommandozeilenschnittstelle der Benutzerverwaltung

Mit Hilfe der Kommandozeilenschnittstelle können Sie administrative Aufgaben für ihre MyCoRe-Anwendung durchführen. So können Sie etwa Objekte (Dokumente, Derivate, Legal Entities, ...) oder Klassifikationen in das Repository einlesen, aktualisieren und löschen, Suchen durchführen und Objekte in Dateien exportieren. Diese Funktionalitäten sind insbesondere bei einer Migration eines bestehenden Systems zur Sicherung der Daten sehr sinnvoll.

Ein Teil der verfügbaren Kommandos der Kommandozeilenschnittstelle ermöglicht die Verwaltung von Benutzern, Gruppen und Privilegien. Diese Kommandos werden im folgenden vorgestellt. Oft werden bei den Kommandos XML-Dateien mit Definitionen von Benutzern, Gruppen und Privilegien erwartet. Die Syntaxen dieser XML-Beschreibungen finden Sie im ProgrammerGuide.

Es werden nicht alle Geschäftsprozesse der Benutzerverwaltung in den folgenden Kommandos abgebildet. Der Schwerpunkt liegt auf einem Management-Interface für administrativen Zugriff. Die GUI der Benutzerverwaltung (ab Version 1.0 verfügbar) wird die restlichen Geschäftsprozesse behandeln.

5.2.1 Allgemeine Kommandos

Die folgenden Kommandos sind allgemeiner Natur.

init superuser – Dieses Kommando wird nur bei der Installation und Konfiguration des MyCoRe-Systems verwendet. Dabei werden Daten über den zu verwendenen Administrationsaccount und den Gastaccount aus den Konfigurationsdateien gelesen, die grundlegenden Privilegien installiert und das Benutzersystem initialisiert.

check user data consistency – Dieses Kommando dient zur Kontrolle der Konsistenz des Benutzersystems. Alle Verbindungen zwischen Benutzern und Gruppen sowie zwischen Gruppen untereinander werden kontrolliert und Unregelmässigkeiten, die eventuell durch den Import von Daten (siehe weiter unten) entstenden sind, werden ausgegeben.

set user management to read only mode

set user management to read/write mode – Mit diesen Kommandos können die Daten der Benutzerverwaltung eingefroren werden. Dies sollte vor dem Exportieren von Daten in XML-Dateien geschehen, damit sich nicht wärend des Exports Daten ändern oder Objekte angelegt werden.

5.2.2 Kommandos zum Arbeiten mit XML-Dateien

create user data from file {0}

create group data from file {0} – Diese Kommandos erwarten eine XML-Datei als Parameter. In dieser Datei müssen ein oder mehrere Definitionen von Benutzern oder Gruppen existieren, die dann in das System integriert werden. Ein Benutzerpasswort muss im Klartext in der definierenden XML-Datei vorliegen (für die Syntax siehe den ProgrammerGuide). Ist die Passwortverschlüsselung eingeschaltet (siehe mycore.properties.user), so wird das Passwort bei der Ablage in der Datenbank automatisch verschlüsselt. Bei der Erzeugung der Objekte wird die

Korrektheit der Eingaben bezüglich vorhandener Regeln überprüft. So wird z.B. getestet, ob sich eine Gruppe durch schon vorhandene Gruppen im System implizit selbst enthält.

import user data from file {0}

import group data from file {0}- Diese Kommandos verhalten sich ähnlich den vorhergehenden Befehlen, mit dem Unterschied, dass Daten ohne Logiküberprüfung eingelesen werden und Benutzerpasswörter bei eingeschalteter Passwortverschlüsselung verschlüsselt in den XML-Dateien vorliegen müssen. Die Import-Befehle werden üblicherweise benutzt, wenn Objekte zuvor in XML-Dateien exportiert wurden und wieder eingelesen werden sollen.

update user data from file {0}

update group data from file {0}

update privileges data from file {0} – Mit diesen Befehlen werden bereits vorhandene Benutzer, Gruppen und Privilegien aktualisiert. Dabei ist zu bedenken, dass „update" im Sinne von „festsetzen auf neue Werte" zu verstehen ist, die Objekte also nach dem update genau die Definitionen haben, die in den XML-Dateien festgelegt werden. Einige der Attribute können allerdings nicht verändert werden, z.B. die Erzeuger-Accounts oder das Datum der Erzeugung. Müssen diese Daten unbedingt verändert werden, dann müssen die Objekte vorher gelöscht und neu angelegt werden.

save all users to file {0}

save all groups to file {0}

save all privileges to file {0}

save user {0} to file {1}

save group {0} to file {1} – Mit diesen Kommandos werden alle oder einzelne Objekte der Benutzerverwaltung in XML-Dateien gespeichert. Passwörter von Benutzern werden bei eingeschalteter Verschlüsselung verschlüsselt abgelegt.

encrypt passwords in user xml file {0} to file {1} – Passwortverschlüsselung kann durch einen Konfigurationsparameter in der Datei mycore.properties.user aktiviert oder deaktiviert werden. Dieses Kommando wird benötigt, wenn man ein bestehendes System mit nicht eingeschalteter Verschlüsselung auf ein System mit Verschlüsselung migrieren will. Dabei verfährt man folgendermaßen: Zunächst werden alle Benutzer des alten Systems mit dem Kommando (siehe oben) *save all users to file* in eine XML-Datei exportiert. Daraufhin wendet man *encrypt passwords in user xml file {0} to file {1}* auf diese Datei an und erhält damit verschlüsselte Passwörter in den XML-Dateien. Mit dem Kommando (siehe oben) *update user data from file* können diese Daten in das System reintegriert werden. Danach muss die Kommandozeilenschnittstelle geschlossen und die Verschlüsselung in mycore.properties.user eingeschaltet werden.

5.2.3 Kommandos zum direkten Arbeiten mit Objekten der Benutzerverwaltung

delete user {0}

delete group {0} – Durch Angabe der ID von Benutzern oder Gruppen werden die Objekte mit diesen Kommandos aus dem System entfernt (und abhängige Objekte aktualisiert).

list all users

list all groups

list all privileges- Die Kommandos dienen dem Auflisten der Objekte der Benutzerverwaltung und sind selbsterklärend.

set password for user {0} to {1} – Mit Hilfedieses Befehls kann das Passwort eines Benutzers direkt über die Kommandozeile gesetzt werden. Voraussetzung ist, dass die notwendigen Privilegien vorliegen.

show user {0}

show group {0} – Mit diesen Kommandos werden ein Benutzer oder eine Gruppe, bestimmt durch Angabe der zugehörigen ID, in XML-Syntax dargestellt.Damit sind alle zugehörigen Daten einsehbar.

enable user {0}

disable user {0} – Mit Hilfe dieser Kommandos können einzelne Benutzer temporär deaktiviert und wieder aktiviert werden. Ist ein Benutzer disabled, so kann er oder sie sich nicht mehr am System anmelden.

add group {0} as member to group {1}

remove group {0} as member from group {1}

add user {0} as member to group {1}

remove user {0} as member from group {1} – Mit diesen Kommandos kann direkt auf die Mitgliederlisten von Gruppen zugrgriffen werden, indem Mitglieder (sowohl Gruppen als auch Benutzer) hinzugefügt oder gelöscht werden können.

5.2.4 Das Sichern und Restaurieren der Benutzerverwaltungsdaten

Während der Initialisierung eines MyCoRe-Systems werden ein Administrationsaccount und ein Gastzugang eingerichtet zusammen mit den zugehörigen primären Gruppen (siehe Kommando *init superuser*). Dadurch ist das Sichern und Reimportieren der gesamten Daten der Benutzerverwaltung mit etwas mehr Handarbeit verbunden, weil der Administrationsaccount und Gastzugang zwar mit gesichert werden, aber vor einer Restauration der Daten z.B. nach einem Crash der SQL-Datenbank neu initialisiert werden müssen. Das bedeutet, dass sie bereits vorhanden sind und ein *import user data from file* deswegen nicht geht. Andererseits können sich die Daten dieser beiden Benutzer natürlich auch verändert haben, so dass die alten Daten wieder hergestellt werden müssen. Der folgende Ablauf führt zum Ziel. Dabei stehen <superuser> und <superuser-group> bzw. <guest> und <guest-group> für die in mycore.properties.private eingetragenen Parameter für den Administrations- und Gastzugang. In der MyCoRe-Kommandozeile werden die folgenden Befehle durchgeführt:

MyCoRe:> save user <superuser> to file <superuser.xml>
MyCoRe:> save user <guest> to file <guest.xml>
MyCoRe:> save group <superuser-group> to file <superuser-group.xml>
MyCoRe:> save group <guest-group> to file <guest-group.xml>
MyCoRe:> save all users to file <all-users.xml>
MyCoRe:> save all groups to file <all-groups.xml>

MyCoRe:> save all privileges to file <privileges.xml>

Die Benutzer <superuser> und <guest> sowie die zugehörigen Gruppen müssen aus den Dateien <all-users.xml> bzw. <all-groups.xml> manuell entfernt werden. Dann können alle Daten in einer neu erstellten SQL-Datenbank folgendermassen importiert werden:

MyCoRe:> init superuser
MyCoRe:> update privileges data from file <privileges.xml>
MyCoRe:> import user data from file <all-users.xml>
MyCoRe:> import group data from file <all-groups.xml>
MyCoRe:> update user data from file <superuser.xml>
MyCoRe:> update user data from file <guest.xml>
MyCoRe:> update group data from file <superuser-group.xml>
MyCoRe:> update group data from file <guest-group.xml>
MyCoRe:> check user data consistency

6.Hints & Tips / Troubleshooting

6.1 Eine zweite Instanz von eXist auf einem System einrichten

Wenn Sie eine zweite Instanz von eXist auf Ihrem System (Computer) einrichten möchten, so ist es erforderlich, dass diese mit anderen Port-Nummern arbeitet. Andernfalls kommt es zum Konflikt und eXist lässt sich nicht starten. Dies ist unabhängig von der User-ID, unter der eXist laufen soll.

Folgende Arbeiten sind durchzuführen:

- Packen Sie eine Original-eXist-Distribution aus.
- Führen Sie sie alle Anpassungen wie oben beschrieben durch.
- Im Wurzel-Verzeichnis von eXist ($EXIST_HOME) ist in der Datei *client.properties* der Port von 8081 auf 8091 (oder einen anderen) zu ändern. Auch im Wert der **alternateURI** ist dies erforderlich.
- In *build.xml* ist ebendfalls der Port unter dem Wert **location** auf 8090 (oder einen passenden) zu ändern.
- Um die Änderungen vollständig zu machen, sollten Sie auch die Datei *backup.properies* entsprechend anpassen.
- Im Source-Code sind die Klassen org.exist.Server.java , org.exist.ServerShutdown.java, org/exist/exist.xsl , org.exist.InteractivClient.java und org.exist.client.InteractivClient.java anzupassen und mit build.sh neu zu compilieren.

Nach dem Start des eXist-Servers steht dieser nun unter dem neuen Port zur Arbeit bereit.