

MyCoRe User Guide

Frank Lützenkirchen (Essen/Duisburg)

Jens Kupferschmidt, Ute Starke (Leipzig)

Detlev Degenhardt (Freiburg)

Johannes Bühler (Greifswald)

Ulrike Krönert, Andreas Trappe, Thomas Scheffler (Jena)

Kathleen Krebs (Hamburg)

Anja Schaar (Rostock)

Heiko Helmbrecht (München)

Release 1.3

26. Mai 2006

Inhaltsverzeichnis

1 Allgemeines zu MyCoRe.....	7
1.1 Vorbemerkungen.....	7
1.1.1 Betriebssysteme.....	7
1.1.2 Relationale Datenbanken.....	7
1.1.3 Suchindizes.....	7
1.1.4 Textextraktion.....	8
1.1.5 Video Streamer.....	8
1.1.6 Servlet-Engine.....	8
1.1.7 WEB-Server.....	8
1.2 Die Beispielanwendung.....	8
1.3 Installationswege.....	8
2 Allgemeine Umgebungs- und Softwarevoraussetzungen.....	10
2.1 Erforderliche und zusätzliche Softwareprodukte.....	10
2.2 Die Installation von JAVA, ANT und CVS.....	11
2.2.1 Die Installation von JAVA.....	11
2.2.2 Die Installation von ANT.....	11
2.2.3 Die Installation von CVS.....	11
2.3 Setzen der Umgebungsvariablen.....	11
2.3.1 Für Windows.....	11
2.3.2 Für UNIX.....	12
2.4 Hinweise zur Installation zusätzlicher Komponenten.....	12
2.4.1 Installation von MySQL.....	12
2.4.2 Installation von Tomcat.....	14
2.4.3 Installation des Apache WEB Servers.....	15
3 Download und Installation des MyCoRe Kerns.....	16
3.1 Download des MyCoRe Kerns.....	16
3.2 Konfiguration und Übersetzen des Kerns.....	17
4 Die MyCoRe-Beispielanwendung DocPortal.....	19
4.1 Grundlegender Aufbau und Ziel der Beispielanwendung.....	19
4.1.1 Allgemeines.....	19
4.1.2 Weiterführende Erläuterungen.....	19
4.2 Download der Beispielanwendung.....	20
4.3 Konfiguration und Installation von DocPortal.....	20
4.3.1 Grundlegendes zur Konfiguration.....	20
4.3.2 Initialisieren von DocPortal.....	21
4.3.3 Aufruf der MyCoRe-Kommandozeile.....	21
4.3.4 Erzeugen der Web-Anwendung.....	22
4.4 Laden der Beispieldaten.....	23
5 Die Arbeit mit der Beispielanwendung.....	24
5.1 Benutzer.....	24
6 Vom Beispiel zum eigenen Dokumenten-Server.....	26
6.1 Allgemeines.....	26
6.2 Nutzung anderer Konzepte für die Datenbasis.....	27
6.2.1 MySQL.....	27
6.2.2 Nutzung der Datenbanken ohne Hibernate.....	28
6.3 Nutzung anderer Search-Backends.....	29
6.3.1 Die Installation der freien XML:DB eXist.....	29
6.4 Die Zusammenarbeit mit anderen DocPortal-Installationen.....	31
6.4.1 Die eigene Installation.....	31
6.4.2 Weitere Server benachbarter Einrichtungen.....	31

6.4.3 Standard-Server des MyCoRe-Projektes.....	31
6.5 Einbindung virtueller Host-Namen mit Hilfe des Apache-Web-Server.....	32
6.5.1 Installation von httpd-2.0.54 mit Einbindung von mod_proxy.....	32
6.5.2 Die Verbindung von Tomcat5 und Apache2.....	32
6.6 Nutzung der OAI Schnittstelle.....	33
6.6.1 Grundlagen.....	33
6.6.2 Der OAI Data Provider.....	34
6.6.3 Installation.....	34
6.6.4 Der Deployment Descriptor.....	34
6.6.5 Die mycore.properties.oai.....	35
6.6.6 Instanzunabhängige Properties.....	35
6.6.7 Instanzabhängige Properties.....	35
6.6.8 Test.....	36
6.6.9 Zertifizierung und Registrierung.....	36
7 Weiterführende Informationen zum Aufbau von MyCoRe-Anwendungen.....	37
7.1 XML-Syntax des Datenmodells.....	37
7.1.1 Die MCRObjektID.....	37
7.1.2 Das Klassifikationen-Datenmodell.....	37
7.1.3 Das Metadatenmodell.....	39
7.1.4 Das Speichermodell für die Multimediadaten (IFS).....	54
7.2 Die Verwendung der Kommandozeilenschnittstelle.....	55
7.2.1 Basiskommandos des Kommandozeilensystems.....	55
7.2.2 Kommandos zur Arbeit mit der Benutzerverwaltung.....	55
7.2.3 Kommandos zur Arbeit mit den Daten.....	58
7.2.4 Kommandos zum Verwalten von Klassifikationen.....	58
7.2.5 Kommandos zur Verwaltung der Objekte.....	59
7.2.6 Anfragen an das System per Kommandozeilen.....	60
7.2.7 Sonstige Kommandos.....	60
7.3 Das SimpleAccessControl-System.....	60
7.3.1 Benutzer und Privilegien.....	61
7.3.2 Zugriffsdaten eines Objektes.....	61
7.3.3 Testmechanismen.....	62
7.3.4 Die konkrete Umsetzung im DocPortal.....	63
7.4 Das SimpleWorkflow-System zur interaktiver Autorenenarbeit.....	66
7.4.1 Das MCRStartEditorServlet.....	67
7.4.2 Abläufe für neue Datenobjekte.....	69
7.4.3 Abläufe für Datenobjekte aus dem Server.....	69
7.4.4 Einbindung in das SimpleAccessControl-System.....	69
7.5 Klassifikationseditor.....	70
7.5.1 Start des Klassifikationseditors.....	70
7.5.2 Operationen des Klassifikationseditors.....	71
7.5.3 Klassifikationen.....	71
7.5.4 Kategorien.....	73
8 Hints & Tips / Troubleshooting.....	78
8.1 Eine zweite Instanz von eXist auf einem System einrichten.....	78

Abbildungsverzeichnis

Abbildung 1: MyCoRe-Schichtenmodell.....	19
Abbildung 2: Ausschnitt der httpd.conf.....	32
Abbildung 3: Änderungen in der server.xml.....	33
Abbildung 4: Einbindung des OAI-Data-Providers in web.xml.....	34
Abbildung 5: XML-Syntax eines Klassifikations-Objektes.....	39
Abbildung 6: XML-Syntax eines Metadaten-Objektes.....	40
Abbildung 7: XML-Syntax eines structure XML-Knotens.....	41
Abbildung 8: XML-Syntax eines metadata XML-Knotens.....	42
Abbildung 9: XML-Syntax des Metadaten-Typs MCRMetaAddress.....	43
Abbildung 10: Beispiel des Metadaten-Typs MCRMetaAddress.....	43
Abbildung 11: XML-Syntax des Metadaten-Typs MCRMetaBoolean.....	44
Abbildung 12: Beispiel des Metadaten-Typs MCRMetaBoolean.....	44
Abbildung 13: XML-Syntax des Metadaten-Basistyps MCRMetaClassification.....	44
Abbildung 14: Beispiel des Metadaten-Basistyps MCRMetaClassification.....	45
Abbildung 15: XML-Syntax des Metadaten-Basistyps MCRMetaDate.....	45
Abbildung 16: Beispiel des Metadaten-Basistyps MCRMetaDate.....	45
Abbildung 17: Syntax des Metadaten-Basistyps MCRMetaHistoryDate.....	46
Abbildung 18: Syntax des Metadaten-Basistyps MCRMetaInstitutionName.....	47
Abbildung 19: Syntax des Metadaten-Basistyps MCRMetaISBN.....	47
Abbildung 20: Syntax des Metadaten-Basistyps MCRMetaISO8601Date.....	47
Abbildung 21: Beispiel des Metadaten-Basistyps MCRMetaISO8601Date.....	48
Abbildung 22: XML-Syntax des Metadaten-Basistyps MCRMetaLangText.....	48
Abbildung 23: Beispiel des Metadaten-Basistyps MCRMetaLangText.....	48
Abbildung 24: XML-Syntax des Metadaten-Basistyps MCRMetaLink.....	49
Abbildung 25: Beispiel des Metadaten-Basistyps MCRMetaLink.....	49
Abbildung 26: Syntax des Metadaten-Basistyps MCRMetaNBN.....	50
Abbildung 27: XML-Syntax des Metadaten-Basistyps MCRMetaNumber.....	50
Abbildung 28: Beispiel des Metadaten-Basistyps MCRMetaNumber.....	50
Abbildung 29: XML-Syntax des Metadaten-Basistyps MCRMetaPersonName.....	51
Abbildung 30: Beispiel des Metadaten-Basistyps MCRMetaPersonName.....	52
Abbildung 31: XML-Syntax des Metadaten-Basistyps MCRMetaXML.....	52
Abbildung 32: Beispiel des Metadaten-Basistyps MCRMetaXML.....	53
Abbildung 33: XML-Syntax des service XML-Knotens.....	53
Abbildung 34: XML-Syntax des Derivate-Datenmodells.....	54
Abbildung 35: Das Prinzip des SimpleAccessControl-Systems.....	61
Abbildung 36: Zusammenhang von Gruppen, Benutzern und Privilegien.....	61
Abbildung 37: XML-Syntax des service XML-Knotens in Hinblick auf das SimpleAccessControl-System.....	62
Abbildung 38: Funktionsschema des SimpleWorkflow.....	67

Tabellenverzeichnis

Tabelle 1: Aufbau des MyCoRe-Kerns.....	17
Tabelle 2: Feste Test-Instanzen für das MyCoRe-Beispiel.....	31
Tabelle 3: Aufbau der MCRObjectID.....	37
Tabelle 4: MyCoRe-Basisdatentypen.....	42
Tabelle 5: Liste der möglichen administrativen Privilegien.....	63
Tabelle 6: Liste der möglichen Bearbeitungs-Privilegien.....	64
Tabelle 7: Liste der möglichen Lese-Privilegien.....	65
Tabelle 8: Benutzer des DocPortals.....	65
Tabelle 9: Parameter des MCRStartEditorServlets.....	68
Tabelle 10: Mögliche Aktionen mit dem MCRStartEditorServlet auf dem Plattenbereich.....	68
Tabelle 11: Mögliche Aktionen mit dem MCRStartEditorServlet im Server.....	69

Vorwort

Dieses Dokument ist für einen Step-by-Step 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 aller vorab notwendigen Installationen. Weiter geht es über die schrittweise Installation des MyCoRe-Kerns und der Beispielanwendung DocPortal. Ein weiteres Kapitel beschreibt die umfassende Gestaltung eines Dokumenten-Servers auf der Basis von DocPortal. Abschließend sind noch einige Hinweise aus der MyCoRe-Gemeinde zur Nutzung alternativer Speicherkomponenten, zur Integration von MyCoRe in ein Bibliotheksumfeld und zur Nutzung in anderen Umfeldern dokumentiert. Am Schluss wird noch auf bekannte Probleme, kritische Szenarien und die weitere Entwicklung von MyCoRe eingegangen.

Für einen ersten Schnelleinstieg wird eine spezielle Distribution der DocPortal-Anwendung auf den MyCoRe-WEB-Seiten angeboten. Bitte konsultieren Sie bei Detailfragen auch das mitgelieferte Programmer Guide. Hier sind die Einzelheiten von MyCoRe genau beschrieben. Die Beispielanwendung DocPortal ist in einer gesonderten Dokumentation ausführlich beschrieben.

ACHTUNG, diese Dokumentation basiert auf dem Release 1.3 der MyCoRe-Software! Um die aktuellsten Features nutzen zu können, benutzen Sie bitte den aktuellen Code Stand inklusive der aktuellen Fixes!

1 Allgemeines zu MyCoRe

Vorbemerkungen

Das MyCoRe-Projekt ist so gestaltet, dass es dem einzelnen Anwender frei steht, welche Komponenten er aus einer Auswahl für die Speicherung und Verarbeitung 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.

1.1 Betriebssysteme

MyCoRe ist auf Grund seiner vollständigen Implementation in JAVA Betriebssystem-unabhängig. Innerhalb der MyCoRe-Community gibt es Installationen unter Unix (Linux, AIX, Solars) wie auch unter MS Windows und MacOS. Die Entscheidung für ein bestimmtes OS ist stark vom Einsatzzweck und der Verfügbarkeit der zusätzlich benötigten Komponenten abhängig.

1.2 Relationale Datenbanken

Im Bereich der SQL Datenbasis werden ab MyCoRe 1.3 alle Zugriffe über das Hibernate¹-API realisiert. Dieses garantiert eine einheitliche Programmierschnittstelle gegenüber der Anwendung und gestattet die Nutzung einer Reihe von SQL-Datenbanken wie HSQLDB², MySQL³, PostgreSQL⁴, IBM DB2⁵, Oracle⁶, usw. Über die Konfiguration von MyCoRe teilt der Administrator dann der Anwendung mit, welche Datenbank konkret zum Einsatz kommt. Standardmäßig wird für DocPortal die HSQLDB verwendet. Diese erfordert keine zusätzlichen Installationen. Der erforderliche Code ist im MyCoRe-Paket bereits enthalten. Für Produktionsinstallationen sollte jedoch ein den Erfordernissen angepasstes Datenbanksystem gewählt werden.

1.3 Suchindizes

Für die Suche in den abgelegten Daten stehen ebenfalls unterschiedliche Ansätze zur Verfügung. Mit Hilfe einer simplen JDOM-Suchemaschine kann auf einfache Art in den beschreibenden Daten der digitalen Objekte (Metadaten) gesucht werden. Diese Ansatz ist aber eher für Entwicklungszwecke konzipiert. Standard ist der Einsatz einer Volltextsuchmaschine für Metadaten und Texte auf Basis des Apache-Projektes Lucene⁷. Diese ist in MyCoRe fest integriert und bedarf keiner zusätzlichen Installation. Alternativ kann noch die Volltextsuche in der Verwendeten SQL-Datenbank eingesetzt werden. Bisher sind hinsichtlich der Leistungsfähigkeit von Lucene in einer Produktionsumgebung keine Einschränkungen bekannt.

1 siehe <http://www.hibernate.org/>

2 siehe <http://hsqldb.org/>

3 siehe <http://dev.mysql.com/>

4 siehe <http://www.postgresql.org/>

5 siehe <http://www-306.ibm.com/software/data/db2/>

6 siehe <http://www.oracle.com/lang/de/database/index.html>

7 siehe <http://lucene.apache.org/>

1.4 Textextraktion

Zum Extrahieren von durchsuchbaren Volltexten aus Dokumenten werden seitens MyCoRe eine Reihe von externen Anwendungen genutzt. Welche davon bei einer konkreten MyCoRe-Installation zum Einsatz kommen, entscheide der jeweilige Administrator selbst. Über eigene Plugins lassen sich weitere Komponenten problemlos einbinden. Die jeweilige Third-Party-Software muss zusätzlich zu den MyCoRe-Komponenten installiert werden. Derzeit sind für folgende Objekttypen Plugins in MyCoRe integriert: XML, HTML, TXT, PS, PDF, DOC, XLS, PPT, OTD, SXW.

1.5 Video Streamer

MyCoRe kann alternativ für die Speicherung von Audio- und Video-Daten Streaming-Server integrieren und zur Datenhaltung nutzen. Vorgesehen ist der Einsatz eines Helix-Servers⁸ oder des IBM VideoChargers⁹. Beide Produkte müssen extern installiert und dann über die Anwendungskonfiguration eingebunden werden.

1.6 Servlet-Engine

MyCoRe benötigt für eine interaktive Arbeit eine Servlet-Engine. Bekanntester Vertreter hier ist Tomcat¹⁰. Dies ist bei vielen Betriebssystemen heute schon standardmäßig bei einer Installation dabei. Alternativ bietet die MyCoRe-Beispielanwendung DocPortal die Nutzung von Jetty¹¹ an. Besonders für kleinere Anwendungen stellt es eine interessante Alternative dar. Jetty ist in der DocPortal-Distribution mit enthalten. Für große Anwendungen sein noch auf IBM WebSphere¹² verwiesen.

1.7 WEB-Server

Zur Arbeit mit MyCoRe in Produktionsumgebungen empfehlen wir die Nutzung des Apache-WEB-Servers.

Die Beispielanwendung

Zur Demonstration der Leistungsfähigkeit des MyCoRe-Kernsystems wurde von den Entwicklern eine Beispielanwendung konzipiert und umgesetzt. Diese soll anschaulich die Funktionalitäten von MyCoRe aufzeigen und gleichzeitig eine praxisnahe Anwendung darstellen. Aus diesem Grund wurde von den Entwicklern eine Dokumenten-Server-Anwendung mit Namen **DocPortal** geschaffen. Die Installation und Nutzung von MyCoRe und der darauf aufbauenden Applikation **DocPortal** wird im Verlauf dieser Dokumentation in allen Einzelheiten besprochen. Ziel ist es, dem interessierten Anwender von Anfang an einen vollständigen und sofort nutzbaren Dokumenten-Server an die Hand zu geben.

Installationswege

Für die Installation der Beispielanwendung gibt es zwei Wege. Um einen ersten Eindruck vom System zu erhalten, wird von der Entwicklergruppe eine Sample-Distribution auf den Projekt-WEB-

8 siehe <https://helix-server.helixcommunity.org/>

9 siehe <http://www-306.ibm.com/software/data/videocharger/>

10 siehe <http://tomcat.apache.org/>

11 siehe <http://jetty.mortbay.org/jetty/>

12 siehe <http://www-306.ibm.com/software/websphere/>

seiten bereitgestellt. Diese stellt ein fertiges leeres System dar, welche mittels der Software IZPack generiert wurde und selbstinstallierend funktioniert. Eine Anleitung zum Einsatz ist in dem Dokument **QuickInstallationGuide** zu finden.

Die folgende Dokumentation beschäftigt sich damit, wie die Beispielanwendung DocPortal schrittweise zu installieren ist und wie der Administrator per Konfiguration diese Applikation in seine Umgebung einpasst. Hierbei ist stark zu unterscheiden, ob das Zielsystem unter den Betriebssystemen UNIX (Linux) oder MS Windows arbeitet. Hier unterscheiden sich deutlich die Arten der Installation von zusätzlicher Software, aber auch die Gestaltung der Anwendungsumgebung. DocPortal sollte auch unter MacOS laufen, wurde aber von den Entwicklern aus Kapazitätsgründen nicht explizit getestet.

2 Allgemeine Umgebungs- und Softwarevoraussetzungen

In der nachfolgenden Beschreibung wird davon ausgegangen, dass alle Arbeiten unter UNIX/MacOS durch einen anzulegenden Benutzer **mcraadmin** ausgeführt werden. Für die Benutzung von MyCoRe / DocPortal muss der Benutzer nachfolgend gelistete, frei verfügbare Software verwenden können. Als Commandline-Shell ist die **BASH** erforderlich. Hinzu kommt eine Reihe von optionalen Anwendungen, um die volle Leistungsfähigkeit von MyCoRe auszuschöpfen.

Unter MS Windows ist kein spezieller Nutzer vorgesehen. Trotzdem ist eine Reihe von Softwareprodukten erforderlich. Auch Umgebungsvariable sind in diesem Zusammenhang einzustellen. Auch für die Nutzung alternativer Komponenten gibt es in diesem Kapitel Hinweise.

Erforderliche und zusätzliche Softwareprodukte

Produkt	UNIX/MacOS	MS Windows
Java 2 Platform, SE, v1.5.x (J2SE) ¹³ oder höher	erforderlich	erforderlich
CVS-Client ¹⁴	erforderlich	erforderlich
ANT 1.6.x ¹⁵ oder höher	erforderlich	erforderlich
OpenOffice 2.0.x ¹⁶ oder höher	optional, wird zur Textindizierung benötigt	optional, wird zur Textindizierung benötigt, ggf. kann auch die MS Office-Suite benutzt werden
XPDF ¹⁷	optional, wird zur Textindizierung benötigt	
GhostScript ¹⁸	optional, wird zur Textindizierung benötigt	
MySQL	optional, kann HSQLDB ersetzen	optional, kann HSQLDB ersetzen
PostgreSQL	optional, kann HSQLDB ersetzen	optional, kann HSQLDB ersetzen
DB2	optional, kann HSQLDB ersetzen	optional, kann HSQLDB ersetzen
Oracle	optional, kann HSQLDB ersetzen	optional, kann HSQLDB ersetzen
Tomcat	optional, kann die Nutzung von Jetty ersetzen	optional, kann die Nutzung von Jetty ersetzen
Apache	optional, wird zur Umsetzung virtueller Host-Namen verwendet	optional, wird zur Umsetzung virtueller Host-Namen verwendet

Unter Linux sind alle Programme in den gängigen Distributionen enthalten. Für das Betriebssystem AIX stellt die Firma IBM verschiedene Software-Downloads unter <http://ftp.software.ibm.com/> bereit. Unter MS Windows müssen die Installationspakete von den entsprechenden Quellen geholt werden.

¹³ siehe <http://java.sun.com/>

¹⁴ <https://www.cvshome.org/dev/interface.html>

¹⁵ <http://ant.apache.org/>

¹⁶ <http://www.openoffice.org/>

¹⁷ <http://www.foolabs.com/xpdf/download.html>

¹⁸ <http://www.cs.wisc.edu/~ghost/>

Die Installation von JAVA, ANT und CVS

2.1 Die Installation von JAVA

Für Windows

Für UNIX/MacOS

JAVA ist in allen gängigen Linux-Distributionen mit enthalten. Installieren Sie einfach die Pakete

- **java-1_5_0-sun.x**
- **java-1_5_0-sun-devel.x**
- **java-1_5_0-sun-jdbc.x**

2.2 Die Installation von ANT

Für Windows

Für UNIX/MacOS

CVS ist in allen gängigen Linux-Distributionen mit enthalten. Installieren Sie einfach die Pakete

- **ant-1.6.2x**
- **ant-apache-regexp-1.6.2x**

2.3 Die Installation von CVS

Für Windows

Für UNIX/MacOS

CVS ist in allen gängigen Linux-Distributionen mit enthalten. Installieren Sie einfach das Paket

- **cvs-1.12.x**

Setzen der Umgebungsvariablen

2.1 Für Windows

Zuerst sollten Sie ein Script erstellen, welches die Arbeit mit dem CVS-System des MyCoRe-Projektes erleichtert. Alternativ können Sie auch die Umgebungsvariablen im Windows eintragen (siehe Beispiel unten).

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

```
rem Script zum Zugriff auf das CVS in Essen
```

```
set CVS_CLIENT_LOG=cvs_log
set CVSROOT=:pserver:anoncvs@server.mycore.de:/cvs
cvs %1 %2 %3
```

Nun ist zu prüfen, ob die erforderlichen Umgebungsvariablen bereits korrekt gesetzt sind:

Variable	Verweis auf ...
JAVA_HOME	... das Basisverzeichnis der JAVA Installation
ANT_HOME	... das Basisverzeichnis der ANT Installation
MYCORE_HOME	... das Basisverzeichnis des MyCoRe-Quellzweiges (i. d. R. D:\mycore)
DOCPORTAL_HOME	... das Basisverzeichnis des DocPortal-Quellzweiges (i. d. R. D:\docportal)

2.2 Für UNIX

Zuerst sollten Sie ein Script erstellen, welches die Arbeit mit dem CVS-System des MyCoRe-Projektes erleichtert. Alternativ können Sie auch die Umgebungsvariablen in der Datei `.bashrc` eintragen.

Für MS Windows sollte folgendes Script als `cvsmycore.cmd` angelegt werden:

```
# 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 $*
```

Variable	Verweis auf ...
JAVA_HOME	... das Basisverzeichnis der JAVA Installation
MYCORE_HOME	... das Basisverzeichnis des MyCoRe-Quellzweiges (i. d. R. ~/.mycore)
DOCPORTAL_HOME	... das Basisverzeichnis des DocPortal-Quellzweiges (i. d. R. ~/.docportal)

Bitte setzen Sie folgende Einträge in der `.bashrc`:

```
export JAVA_HOME=/usr/lib/java
export MYCORE_HOME=/home/mcraadmin/mycore
export DOCPORTAL_HOME=/home/mcraadmin/docportal
```

Hinweise zur Installation zusätzlicher Komponenten

2.1 Installation von MySQL

MySQL kann als Alternative zu standardmäßig angebotenen Datenbank HSQLDB verwendet werden. Diese Kapitel beschreibt die Installation und Grundkonfiguration von MySQL.

Installation unter Windows

MySQL kann von <http://www.mysql.de> bezogen werden.

1. mysql-essential-4.1.9-win32 oder höher laut mitgelieferter Dokumentation installieren
2. MySQL starten
3. 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 *newpassword* eingegeben werden.

```
mysql -uroot -prootpassword mysql
GRANT ALL PRIVILEGES ON *.* TO mcradmin@localhost WITH GRANT
OPTION;
quit
```

4. Sollen auch externe Hosts zugreifen dürfen, sind noch die folgenden Kommandos erforderlich.

```
mysql -uroot -prootpassword mysql
GRANT ALL PRIVILEGES ON *.* TO mcradmin@'%' IDENTIFIED BY
'newpassword' WITH GRANT OPTION;
quit
```

5. Ist das Passwort einmal gesetzt, müssen Sie zusätzlich die Option *-p* verwenden. Zum Verifizieren, ob der Server läuft, nutzen Sie folgende Kommandos

```
mysqladmin -u mcradmin version
mysqladmin -u mcradmin variables
```

6. Jetzt können Sie die Datenbasis für MyCoRe mit nachstehendem Kommando anlegen.

```
mysqladmin -u mcradmin create mycore
```

7. 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
```

Installation unter UNIX/MacOS

MySQL kann aus der Linux-Distribution oder direkt von <http://www.mysql.de> bezogen werden. Am günstigsten installieren Sie die folgenden Pakete direkt von der Distribution:

- **mysql-4.1.x**
- **mysql-shared-4.1.x**
- **mysql-client-4.1.x**
- **mysql-administrator-1.0.x**
- **mysql-connector-java-3.1.x**

1. MySQL-Server mit `/etc/init.d/mysql start` starten und ggf. in den Systemstart eintragen.
2. 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 *newpassword* eingegeben werden.

```
mysql -uroot -prootpassword mysql
```

```
GRANT ALL PRIVILEGES ON *.* TO mcradmin@localhost WITH GRANT
OPTION;
quit
```

3. Sollen auch externe Hosts zugreifen dürfen, sind noch die folgenden Kommandos erforderlich.

```
mysql -uroot -prootpassword mysql
GRANT ALL PRIVILEGES ON *.* TO mcradmin@'%' IDENTIFIED BY
'newpassword' WITH GRANT OPTION;
quit
```

4. Ist das Passwort einmal gesetzt, müssen Sie zusätzlich die Option `-p` verwenden. Zum Verifizieren, ob der Server läuft, nutzen Sie folgende Kommandos

```
mysqladmin -u mcradmin version
mysqladmin -u mcradmin variables
```

5. Jetzt können Sie die Datenbasis für MyCoRe mit nachstehendem Kommando anlegen.

```
mysqladmin -u mcradmin create mycore
```

6. 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
```

2.2 Installation von Tomcat

Tomcat kann als Alternative zum in DocPortal mitgelieferten Jetty eingesetzt werden. Tomcat ist ebenfalls eine Servlet-Engine, die besonders bei größeren Installationen Anwendung findet.

Installation unter Windows

Tomcat kann über <http://jakarta.apache.org/tomcat/index.html> geholt werden. Die Installation ist laut mitgelieferter Dokumentation vorzunehmen.

Die Umgebungsvariablen sind wie folgt beschrieben zu setzen, damit Tomcat auch von **mcradmin** genutzt werden kann. Weiterhin werden der Memory Size für den Server erweitert.

```
set CATALINA_HOME = Pfad zum Home-Verzeichnis
set CATALINA_OPTS = "%CATALINA_OPTS -server -Xms256m -Xmx512m -Xincgc"
```

MyCoRe arbeitet beim Encoding der Zeichen standardmäßig 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 `%CATALINA_HOME%/conf/server.xml` anpassen. Ergänzen Sie das Tag für den **Connector** um das Attribut **URIEncoding='UTF-8'**.

Installation unter UNIX/MacOS

Tomcat finden Sie auf jeder gängigen Linux Distribution. Die Dokumentation von Tomcat steht unter `/usr/share/doc/packages/tomcat5`. Installieren Sie bitte die Pakete

- **tomcat5-5.x**
- **tomcat5-admin-webapps-5.x**

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/tomcat5
export CATALINA_OPTS="$CATALINA_OPTS -server -Xms256m -Xmx512m
-Xincgc"
```

MyCoRe arbeitet beim Encoding der Zeichen standardmäßig 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"**.

2.3 Installation des Apache WEB Servers

Die Installation des Apache Web Servers ist eigentlich für das MyCoRe/DocPortal-Beispiel nicht zwingend erforderlich. Sollten Sie aus diesem Beispiel jedoch eine eigene Produktionsinstallation ableiten wollen, so benötigen Sie den Server zur Gestaltung virtueller Hosts. Hier wird aber nur der Proxy-Modul verwendet. Die Konfiguration wird weiter hinten in diesem Buch besprochen.

Installation unter Windows

Die Installation des Apachen WEB Servers ist relativ einfach und in der Dokumentation gut beschrieben.

1. Apache2 für Windows oder höher über <http://httpd.apache.org/> holen
2. laut mitgelieferter Dokumentation installieren und testen

Installation unter UNIX/MacOS

Der Apache WEB Server ist in jeder gängigen Linux-Distribution enthalten und sollte in seiner Grundkonfiguration kein Problem darstellen. Es wird empfohlen, den Apachen ab Version 2.x zu verwenden. Installieren Sie die Pakete

- **apache2-2.0.x**
- **apache2-devel-2.0.x**

3 Download und Installation des MyCoRe Kerns

Download des MyCoRe Kerns

Für eine Step-by-Step Installation der DocPortal-Anwendung benötigen Sie zuerst das MyCoRe-Kernsystem. DocPortal wird sich im weiteren Verlauf der Installation darauf beziehen und sich die erforderlichen Komponenten daraus laden. Die Installation des Kerns muss als Root-Verzeichnis das unter der Umgebungsvariablen **MYCORE_HOME** angegebene Directory nutzen. Derzeit steht der MyCoRe-Kern auf dem CVS-Server server.mycore.de zum Download bereit.

Um Zugang zum CVS-System zu erlangen, führen Sie zuerst folgendes Kommando aus (**das Passwort ist anoncvs**)

```
touch ~/.cvspass ; cvsmymcore.sh login
oder
cvsmymcore login
```

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

```
cvsmymcore.sh checkout -r release_1_3_fixes mycore
oder
cvsmymcore checkout -r release_1_3_fixes mycore
```

Um den aktuellen Arbeitsstand der Entwickler zu erhalten, lassen Sie bitte einfach die -r Option bei dem Kommando weg. Achtung, es kann sein, dass sie so eine fehlerhafte Version herunterladen. Setzen Sie diese **nicht für Produktionssysteme** ein!

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

Relativer Pfad	Inhalt
bin	Das Verzeichnis für die Shell-Skripts des Kerns.
build.xml	Die Konfigurationsdatei für den Build-Prozess.
changelog.txt	Enthält Hinweise zu Produktänderungen.
config	Einige wenige Konfigurationsdateien für den Kern
documentation	Enthält alle MyCoRe-Dokumentationen.
documentation/StartingGuide	Eine kurze Einleitung in das Projekt.
documentation/UserGuide	Das Handbuch für die Standard-Installation von 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	Verzeichnis des Java-Quellcodes. Eine Beschreibung der

Relativer Pfad	Inhalt
	Pakete steht im ProgrammerGuide.
stylesheets	XSLT Stylesheets des MyCoRe-Kerns.

Tabelle 1: Aufbau des MyCoRe-Kerns

Es wird immer wieder Korrekturen am Code geben, welche bestehende Fehler beseitigen. Es ist daher immer mal sinnvoll, ein Update zu fahren. Ggf. sollten Sie vorher eine Anfrage per Mail an mycore-user@lists.sourceforge.net stellen, ob derzeit ein stabiler Code-Stand vorhanden ist. 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
```

Konfiguration und Übersetzen 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 sollten Sie zunächst diese Software installieren. Der Aufruf `ant usage` im `mycore`-Verzeichnis zeigt bei richtiger Konfiguration, welche Aufrufe möglich sind.
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, Lucene, SQL) und die Speicherung von Tabellen über JDBC in einer relationalen Datenbank (Hibernate, IBM DB2, MySQL, optional auch andere) wird in der Datei `config/build.properties` festgelegt. Nach dem erstmaligen Checkout des Kerns befindet sich im `config`-Verzeichnis nur das Template `build.properties.template`. Diese Datei muss nach `build.properties` kopiert werden.
3. In der Regel werden Sie nur die beiden entsprechenden Blöcke für die verwendete XML-Datenbank (`MCR.XMLStore.*`) - **Standard ist Lucene** - und die verwendete relationale Datenbank (`MCR.JDBCStore.*`) - **Standard ist Hibernate** - 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

```
ant 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.

1. Eine Übersicht über alle wesentlichen Build-Ziele erhalten Sie mit

```
ant usage
```

2. Übersetzen Sie alle MyCoRe Quellcode-Dateien mit dem Befehl

```
ant jar
```

Dabei entsteht eine Jar-Datei *lib/mycore.jar*.

3. Optional können Sie auch JavaDoc Quellcode-Dokumentation im HTML-Format generieren lassen, indem Sie

```
ant javadocs
```

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

Damit sind alle Arbeiten im Kernsystem abgeschlossen und sie können nun die eigentliche Anwendung, in diesem Fall die Beispiel-Applikation DocPortal installieren.

4 Die MyCoRe-Beispielanwendung DocPortal

Grundlegender Aufbau und Ziel der Beispielanwendung

4.1 Allgemeines

Um die Funktionsweise des MyCoRe-Kernes zu testen wurde eine Beispiel-Anwendung basierend 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 Docportal genannt, gleichzeitig sinnvoll einsetzen zu können, wurde als Beispielszenario ein Dokumenten-Server gewählt, wie er bei vielen potentiellen Nutzern zur Anwendung kommt. Auch soll das Docportal die Nachfolge des erfolgreichen MILESS-Dokumenten-Servers sein und den Migrationspfad zu MyCoRe hin aufzeigen.

Die Beispiel-Applikation ist in zwei Ebenen aufgeteilt. Für eine Nachnutzung von DocPortal ist es auch möglich eine weitere Anwendungsschicht einzufügen. 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 einem gemeinsamen Anwendungskern 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.

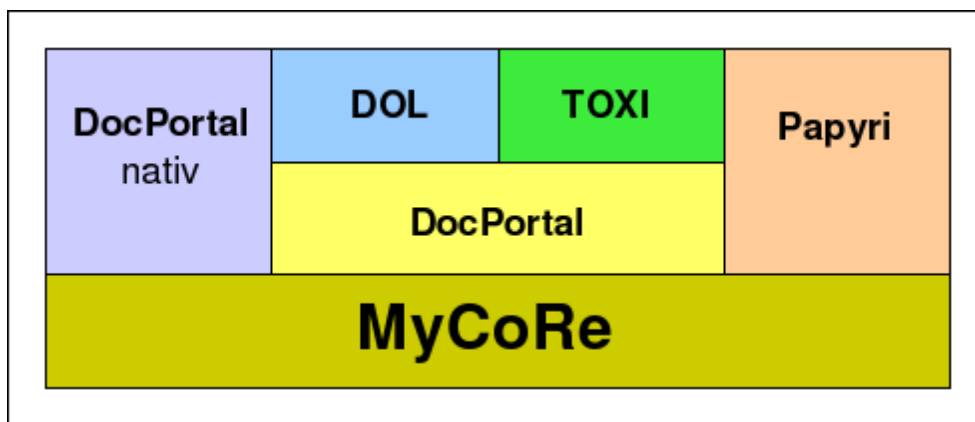


Abbildung 1: MyCoRe-Schichtenmodell

4.2 Weiterführende Erläuterungen

Die weiterführenden Erläuterungen zum Datenmodell und der Struktur der Beispielanwendung finden Sie in der Dokumentation zu DocPortal¹⁹. In dieser Schrift ist das Konzept des Dokumenten-Servers und seiner individuellen Gestaltung als Beispielanwendung beschrieben.

Auf der offiziellen Web-Seite des MyCoRe-Projektes²⁰ finden sie die in den nachfolgenden Abschnitten zu installierende Anwendung bereits lauffähig vor. Nach Abschluss der Arbeiten sollte auch Ihr System so funktionieren.

¹⁹ <http://www.mycore.de/content/main/documentation.xml>

²⁰ <http://mycoresamplelinux.dl.uni-leipzig.de/>

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 DocPortal wird für alle unterstützten Systeme über das CVS Repository ausgeliefert.

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

```
cvsmycore.sh checkout -r release_1_3_fixes docportal
oder
cvsmycore checkout -r release_1_3_fixes docportal
```

Nach erfolgreichem CheckOut finden Sie nun ein Verzeichniss – docportal für die DocPortal-Basis-Applikation inklusive aller Daten für die Beispielanwendung. Der Pfad zu diesem Verzeichnis sollte dem Inhalt der Umgebungsvariable `DOCPORTAL_HOME` entsprechen. Der folgende Abschnitt beschreibt deren Konfiguration. Für Updates dieser Komponenten verfahren Sie wie in Abschnitt 3.1 beschrieben.

Konfiguration und Installation von DocPortal

4.1 Grundlegendes zur Konfiguration

Die Konfiguration der Beispielanwendung ist im Verzeichniss `$DOCPORTAL_HOME/config` zu finden. Hier 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.

Weiterhin sind hier 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!**

mycore.properties.private

Die Datei `mycore.properties.private` enthält eine ganze Reihe von Konfigurationswerten. Für ein erstes System müssen Sie nur folgende Werte anpassen:

- **MCR.basedir** – Pfad zur DoCportal-Root bzw. entspricht `DOCPORTAL_HOME`
- **MCR.editor_mail** – Mail-Adresse, an die alle Nachrichten gehen sollen
- **MCR.mail.server** – Mail-Server, der die Nachrichten versenden soll

mycore.properties.wcms

Docportal enthält ein Modul für das dynamische Verwalten der Webanwendung - WCMS (Web Content Management System). Um den Module in Betrieb zu nehmen, reicht es vorerst, in das Verzeichnis `docportal/modules/module-wcms/aif/config/` zu wechseln und die Datei `mycore.properties.wcms.template` nach `mycore.properties.wcms` zu kopieren.

Eine ausführlichere Beschreibung zum WCMS unter anderem für das Ändern der Passwörter befindet sich unter `$MYCORE_HOME/modules/module-wcms/documentation/Documentation.pdf`

4.2 Initialisieren von DocPortal

Nun gilt es, die Beispielanwendung mit Leben zu füllen. Gehen Sie nacheinander folgende Arbeitsschritte durch. Sollte ein Fehler auftreten, beseitigen Sie diesen, bevor sie weiter arbeiten.

1. Prüfen Sie die Systemumgebung in DocPortal mit

```
ant info
```

2. Erzeugen Sie die erforderlichen Arbeitsverzeichnisse mit

```
ant create.directories
```

3. Generieren Sie die XML Schema Dateien für das Datenmodell mit dem Kommando

```
ant create.schema
```

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

```
ant jar
```

5. Erzeugen Sie die CommandLineTools mit Hilfe des Kommandos

```
ant create.scripts
```

6. Der nächste Schritt ist das Anlegen der Datenbankstrukturen (Data Stores). In der freien Variante werden hier nur die SQL-Tabellen inklusive der Indexe angelegt.

```
ant create.metastore
```

7. Laden Sie die Standard-ACL's, Gruppen und Benutzer für die Beispielanwendung mit

```
ant create.users
```

8. Laden Sie abschließend noch alle mitgelieferten Klassifikationen. Diese werden für ein reibungsloses Funktionieren der Anwendung sowie zum Einspielen der Beispieldaten benötigt.

```
ant create.class
```

Sie sollten nun eine leere, aber vollständige Beispielanwendung haben, in welche **per CommandLineInterface** Daten eingestellt werden könnten. Wie dies geht, wird weiter hinten beschrieben. Die Inbetriebnahme der Web-Anwendung ist in einem der nächsten Kapitel detailliert erklärt.

4.3 Aufruf der MyCoRe-Kommandozeile

Starten Sie die MyCoRe-Kommandozeile, auch CommandLineInterface genannt, durch Aufruf von `bin/mycore.sh` (UNIX/MacOS) bzw. `bin\mycore.cmd` (Windows). Eine kurze Übersicht aller Befehle erhalten Sie durch Eingabe von **help**. Sie verlassen die MyCoRe-Kommandozeile durch Eingabe von **quit** oder **exit**. Mit `help [Kommandoteil]` erhalten Sie einen kurzen Hilfetext. Eine ausführliche Dokumentation enthält Abschnitt 7.2 auf Seite 44.

Sie können natürlich auch die Aufrufe in beliebige Skripte usw. einbinden, um eine Batch-Verarbeitung zu organisieren. Das Laden der Beispieldaten erfolgte auf diese Weise.

4.4 Erzeugen der Web-Anwendung

Dieser Abschnitt beschäftigt sich mit der Inbetriebnahme der DocPortal-Anwendung als WEB-Applikation. Um den Installationsaufwand in Grenzen zu halten, enthält der DocPortal-Code-Baum schon eine fertige Servlet-Engine namens Jetty. Diese ist zwar nicht so mächtig wie Tomcat, ist aber für kleinere und Mittlere Anwendungen und Demonstrationszwecke sehr gut geeignet. Bevor sie jedoch die Jetty-Engine starten sind noch diese Schritte zu tun.

1. Zuerst müssen Sie **einmal** Schlüssel für die Applets in der Anwendung erzeugen. Dies geschieht durch die Eingabe des Kommandos

```
ant create.genkeys
```

2. Nun kann die Webanwendung erstellt werden.

```
ant webapps
```

3. Alternativ können Sie auch ein Web Application Archive (war) erzeugen.

```
ant war
```

Das MyCoRe Build-Script kopiert beim Erzeugen der Web Applikation auch alle externen, erforderlichen *.jar-Dateien Ihrer verwendeten Datenbank-Systeme (DB2, MySQL, ...) in das Verzeichnis *WEB-INF/lib*, entsprechend den Vorgaben Ihrer Konfiguration in *build.properties*.

Nun können Sie die Webanwendung ein erstes mal starten. Benutzen Sie dazu das generierte Script , welches die Servlet-Engine Jetty aktiviert.

```
bin/jettystart.sh bzw. bin/jettystart
```

Starten Sie nun einen Web-Browser und rufen Sie diese URL auf:

<http://localhost:8291/>

Damit Ihre Anwendung auch über die WebServices erreichbar ist (entfernte Anfragen), müssen sie diese Funktionalität noch gesondert aktivieren. Geben Sie bitte **bei laufendem Jetty** noch das Kommando

```
ant webservice.deploy
```

Die nun verfügbaren WebService-Dienste sehen Sie bei eingabe der URL in Ihren Web-Browser.

<http://localhost:8291/servlets/AxisServlet>

Um den Dienst abzuschalten muss ebenfalls **unter laufendem Jetty** das folgende Kommando gegeben werden.

```
ant webservice.undeploy
```

GRATULATION, SIE HABEN NUN ERFOLGREICH DocPortal INSTALLIERT!

Laden der Beispieldaten

MyCoRe stellt eine umfangreiche Sammlung von Beispieldaten bereit. Diese wurde aus Performance-Gründen in ein extra Paket gepackt. Sie können diese aus dem CVS beziehen und in Ihre fertige Anwendung laden. Die Beispieldaten sind in mehrere Blöcke gegliedert, welche einzeln geladen werden können und bestimmte Aspekte des MyCoRe-Projektes verdeutlichen sollen.

Zweig	Beschreibung
audosample	Einige größere Audiodateien
defaultsample	Einige kleiner Beispiel mit Bildern, HTML-Seiten, Text
kochbuch	Teddies Kochbuch von Ursula Reber als hierarchischer Content
theses	Einige Diplomarbeiten im Zusammenhang mit MyCoRe
video	Einige größere Videosequenzen

Das Laden der Daten einer Beispielgruppe in die DocPortal-Anwendung geht wie folgt:

1. Download der Daten direkt neben die Verzeichnisse *mycore* und *docportal* mit

```
cvsmycore.sh checkout -r release_1_3_fixes content
```


bzw.

```
cvsmycore checkout -r release_1_3_fixes content
```
2. Stoppen Sie die Servlet-Engine Jetty. Dies ist nur erforderlich, wenn Sie die HSQLDB mit der Standardkonfiguration benutzen. Sollten Sie HSQLDB im Server-Modus betreiben, ist ein stoppen von Jetty nicht notwendig.

```
bin/jettystop.sh
```

 bzw.

```
bin/jettystop
```
3. Wechseln Sie in das gewünschte Datenverzeichnis, z. B. *defaultsample*
4. Testen Sie die Umgebung mit

```
ant info
```
5. Laden Sie die Daten mit

```
ant load
```
6. Sie können die Daten auch wieder aus Ihrem System entfernen. Dies geht mit

```
ant remove
```
7. Als letztes ist Jetty neu zu starten. Die Daten sind nun auch direkt in Ihrer Web-Anwendung verfügbar.

5 Die Arbeit mit der Beispielanwendung

Benutzer

Um nun eigene Dokumente einstellen zu können oder im WCMS Seiten verändern zu können, benötigen Sie entsprechende Logins. Einige sind schon bei dieser Installation eingerichtet worden.

Benutzer der Beispielanwendung

Die Beispielanwendung bringt zur Demonstration bereits eine Reihe von Benutzern und Gruppen mit. Einer Gruppe können dabei mehrere Benutzer angehören. Die Vergabe von Rechten auf Objekte kann dann sowohl für einen einzelnen Benutzer wie auch für eine ganze Gruppe erfolgen. Im Kapitel xxx wird diese Zugriffssystem (ACL-System) näher erläutert.

Gruppe	Beschreibung
root	Die Superuser-Gruppe
admingroup	Die Gruppe der Systemadministratoren
readergroup1	Eine Gruppe von Lesern der 1.Einrichtung (nur Leserechte für Objekte der Gruppe)
readergroup2	Eine Gruppe von Lesern der 2.Einrichtung (nur Leserechte für Objekte der Gruppe)
authorgroup1	Eine Gruppe von Autoren der 1.Einrichtung (Autorenrechte nur auf die eigenen Objekte)
authorgroup2	Eine Gruppe von Autoren der 2.Einrichtung (Autorenrechte nur auf die eigenen Objekte)
editorgroup1	Eine Gruppe von Editoren der 1.Einrichtung (Bearbeiter aller Objekte einer Einrichtung)
editorgroup2	Eine Gruppe von Editoren der 2.Einrichtung (Bearbeiter aller Objekte einer Einrichtung)
gastgroup	Die Gast-Gruppe

Benutzer	Gruppe	Password
root	rootgroup	alleswirdgut
administrator	admingroup	alleswirdgut
editor1A	editorgroup1	editor1A
editor1B	editorgroup1	editor1B
editor2A	editorgroup2	editor2A
editor2B	editorgroup2	editor2B
author1A	authorgroup1	author1A
author1B	authorgroup1	author1B
author2A	authorgroup2	author2A
author2B	authorgroup2	author2B
reader1A	readergroup1	reader1A
reader2A	readergroup2	reader2A
gast	gastgroup	gast

Benutzer des WCMS

Aktuell ist nur der Benutzer **admin** mit dem Password **wcms** eingerichtet. Im Laufe des kommenden Release wird diese separate Lösung entfallen und es wird grundsätzlich mit dem ACL-System gearbeitet.

Die Verwendung der Kommandozeilenschnittstelle

Mit Hilfe der Kommandozeilenschnittstelle können Sie administrative Aufgaben für ihre MyCoRe-Anwendung auf Kommandozeilenebene durchführen. Dies ermöglicht auch die Abarbeitung von Scripts zu Massenverwaltung der eingestellten Daten. So können Sie etwa Objekte (Dokumente, Derivate, Personen, u.s.w.) 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.

Die nachfolgenden Abschnitte sollen Ihnen eine Übersicht der möglichen Kommandos geben. Die einzelnen Kommandogruppen werden dabei intern in den Kommandokern importiert und stehen dem Benutzer zur Verfügung. Die mit dem MyCoRe-System ausgelieferten Kommandos können beliebig erweitert werden. Konsultieren Sie hierzu das MyCoRe Programmer Guide.

5.1 Basiskommandos des Kommandozeilensystems

Das Kommandozeilensystem enthält eine kleine Zahl an Basiskommandos. Diese sollen im folgenden Abschnitt beschrieben werden. Sie werden ergänzt durch eine Vielzahl von weiteren Kommandos für die einzelnen Bereiche der Arbeit mit MyCoRe. Es besteht auch die Möglichkeit, eigene Anwendungskommandos als Java-Klassen zu entwickeln und zu integrieren. Das Kommandozeilen-Tool ist **mycore.sh** bzw. **mycore.cmd** und steht nach dem Erzeugen mittels `ant create.scripts` im *bin*-Verzeichnis der Anwendung. In den unten stehenden Beschreibungen sind alle erforderlichen Parameter in der Form **{n}** notiert. **n** gibt dabei die Nummer des Parameters an. Die Zählung beginnt bei 0.

Basiskommandos sind:

- help** – zeigt alle möglichen Kommandos, auch die von der Anwendung hinzugefügten.
- help {0}** – zeigt alle Kommandos, die mit {0} beginnen
- process {0}** – führt das im Parameter 0 angegebene System-Shell-Kommando aus.
- ! {0}** – führt das im Parameter 0 angegebene System-Shell-Kommando aus.
- whoami** – zeigt den aktuellen Benutzer an.
- login {0}** – startet einen Benutzerwechsel-Dialog für den Benutzer {0}.
- change to user {0} with {1}** – Wechselt den Benutzer {0} mit dem Passwort {1}.
- exit** – beendet das Kommandozeilensystem.
- quit** – beendet das Kommandozeilensystem.

5.2 Kommandos zur Arbeit mit der Benutzerverwaltung

Eine Gruppe der verfügbaren Kommandos der Kommandozeilenschnittstelle ermöglicht die Verwaltung von Benutzern und Gruppen. Diese Kommandos werden im folgenden vorgestellt. Oft werden bei den Kommandos XML-Dateien mit Definitionen von Benutzern oder Gruppen erwartet. Die Syntax dieser XML-Beschreibungen finden Sie im Programmer Guide. Es werden derzeit nur die wichtigsten Geschäftsprozesse der Benutzerverwaltung in den folgenden Kommandos abgebildet. Der Schwerpunkt liegt auf einem Management-Interface für administrativen Zugriff. Die vollständige GUI der Benutzerverwaltung (geplant für eine spätere Version) wird die Möglichkeit einer Bearbeitung aller Geschäftsprozesse bieten.

Allgemeine Verwaltungskommandos

Die folgenden Kommandos sind allgemeiner Natur.

init superuser – Dieses Kommando wird nur bei der Installation und Konfiguration des MyCoRe-Systems einmalig verwendet. Dabei werden Daten über den zu verwendenden Administrations-Account und den Gast-Account aus den Konfigurationsdateien gelesen sowie das Benutzersystem initialisiert.

check user data consistency – Dieses Kommando dient zur Kontrolle der Konsistenz des Benutzersystems. Alle Verbindungen zwischen Benutzern und Gruppen werden kontrolliert und Unregelmäßigkeiten, die eventuell durch den Import von Daten (siehe weiter unten) entstanden 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 während des Exports Daten ändern oder Objekte angelegt werden.

Kommandos zum Arbeiten mit XML-Dateien

Diese Kommandos dienen dem Anlegen und Ändern von Gruppen und Benutzern aus XML-Dateien heraus.

create user data from file {0}

create group data from file {0} – Diese Kommandos erwarten eine XML-Datei als Parameter. In der 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 Programmer Guide). Ist die Passwortverschlüsselung eingeschaltet (siehe *mycore.properties.private*), 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 IDs doppelt vergeben wurden.

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} – Mit diesen Befehlen werden bereits vorhandene Benutzer 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. Sollen 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 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. Die so entstanden Dateien können beispielsweise mit den `import`-Kommandos wieder geladen werden.

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.

Kommandos zum direkten Arbeiten mit Objekten der Benutzerverwaltung

delete user {0}

delete group {0} – Durch Angabe des Benutzer- oder Gruppennamens werden die Objekte mit diesen Kommandos aus dem System entfernt (und abhängige Objekte aktualisiert).

list all users

list user {0}

list all groups

list group {0} – Die Kommandos dienen dem Auflisten der Objekte der Benutzerverwaltung und sind selbsterklärend.

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

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 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 zugegriffen werden, indem Mitglieder (sowohl Gruppen als auch Benutzer) hinzugefügt oder gelöscht werden können.

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>
```

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:> 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
```

5.3 Kommandos zur Administration des ACL-Systems

load permissions data from file {0} - das Kommando lädt alle statischen Permissions.

update permissions data from file {0} - das Kommando ersetzt alle statischen Permissions.

list all permissions – das Kommando listet alle statischen Permissions.

delete all permissions - das Kommando löscht alle statischen Permissions.

save all permissions to file {0} - das Kommando sichert alle statischen Permissions.

update permission {0} for id {1} with rulefile {2} -

update permission {0} for id {1} with rulefile {2} described by {3} -

update permission {0} for documentType {1} with rulefile {2} -

update permission {0} for documentType {1} with rulefile {2} described by {3} -

5.4 Kommandos zum Verwalten von Klassifikationen

load classification from file {0} – es wird eine Klassifikation in Form einer XML-Definition gelesen und in das System geladen.

load all classifications from directory {0} – es werden alle XML-Definitionen von Klassifikationen aus einem Verzeichnis gelesen und in das System geladen.

update classification from file {0} – es wird eine Klassifikation in Form einer XML-Definition gelesen. Diese überschreibt die im System bereits geladene Klassifikation. Achtung, lassen Sie keine Kategorien einer bestehenden Klassifikation weg, das sonst Ihr System ggf. in einen inkonsistenten Zustand kommen kann!

update all classifications from directory {0} – es werden alle XML-Definitionen von Klassifikationen aus einem Verzeichnis gelesen. Diese überschreiben die im

System bereits geladene Klassifikationen. Achtung, lassen Sie keine Kategorien einen bestehenden Klassifikation weg, das sonst Ihr System ggf. in einen inkonsistenten Zustand kommen kann!

delete classification {0} – es wird eine Klassifikation mit der im Parameter {0} angegebenen MCRObjektID gelöscht.

save classification {0} to {1} – es wird eine Klassifikation mit der im Parameter {0} angegebenen MCRObjektID in eine Datei mit dem im Parameter {1} angegebenen Namen gespeichert.

5.5 Kommandos zur Verwaltung der Objekte

load object from file {0}

load all objects from directory {0}

load derivate from file {0}

load all derivatives from directory {0} – die Kommandos laden die Metadaten-Objekte oder die Derivate inklusive ihrer Bilder und Dokumente von einer Quelldatei oder einem Quellverzeichnis in das System.

update object from file {0}

update all objects from directory {0}

update derivate from file {0}

update all derivatives from directory {0} – die Kommandos laden die Metadaten-Objekte oder die Derivate inklusive ihrer Bilder und Dokumente von einer Quelldatei oder einem Quellverzeichnis in das System. Dabei werden die bestehenden Daten bis auf Strukturinformationen überschrieben. Strukturinformationen sind ggf. Daten über Vater-Kind- oder Objekt-Derivate-Beziehungen.

export object {0} to directory {1} with {2}

export object from {0} to {1} to directory {2} with {3}

export all objects of type {0} to directory {1} with {2}

export derivate of {0} to directory {1}

export derivate from {0} to {1} to directory {2}

export all derivatives to directory {0} with {1} – die Kommandos speichern ein oder mehrere Metadaten-Objekte oder Derivate in ein Verzeichnis ab. Für die Parameter {0} und {1} bzw. {0} sind MCRObjektIDs anzugeben. Es werden zur Transformation der zu speichernden Daten die Stylesheets *mcr_{3}-object.xsl* und *mcr_{3}-derivate.xsl* unter *\$DOCPORTAL_HOME/stylesheets* verwendet.

save object {0} to directory {1}

save object from {0} to {1} to directory {2}

save derivate {0} to directory {1}

save derivate from {0} to {1} to directory {2} – die Kommandos speichern ein oder mehrere Metadaten-Objekte oder Derivate in ein Verzeichnis ab. Für die Parameter {0} und {1} bzw. {0} sind MCRObjektIDs anzugeben. Es werden zur Transformation der zu speichernden Daten die Stylesheets *mcr_save-object.xsl* und *mcr_save-derivate.xsl* unter *\$DOCPORTAL_HOME/stylesheets* verwendet.

delete object {0}

delete object from {0} to {1}

delete derivate {0}

delete derivate from {0} to {1} – die Kommandos löschen einzelnen Metadaten-Objekte oder Derivate oder ganze Gruppen von selbigen. Alle Parameter müssen dabei MCRObjektID's sein.

delete all objects of type {0} – das Kommando löscht alle eingestellten Objekte eines bestimmten Datentypes.

delete all derivatives – das Kommando löscht alle eingestellten Derivate.

show loadable derivate of {0} to directory {1} – das Kommando speichert den Content des Derivates vom Objekt mit der MCRObjektID {0} in das Verzeichnis {1}.

5.6 Anfragen an das System per Kommandozeilen

query host {0} {1} {2} – das Kommando gestattet eine Anfrage an die Datenbasis. Parameter {0} ist dabei ein bekannter Hostalias oder eines der Schlüsselworte **local** oder **remote**. Parameter {1} enthält den MCRObjektID-type bzw. den Layout-type (document oder alldocs). Der letzte Parameter beinhaltet die eigentliche Anfrage im Query-Syntax.

query local {0} {1}

query remote {0} {1} – in beiden Kommandos ist der Host schon gesetzt. Die Anfragen laufen gegen das lokale System oder gegen die remote-Schnittstellen.

run query from file {0} – das Kommando startet eine Anfrage mit einer Query, welche im File {0} steht.

5.7 Sonstige Kommandos

repair metadata search of type {0}

repair metadata search of ID {0} – die Kommandos lesen die XML-SQL-Tabellen und reparieren zerstörte Metadaten-Suchindizes. Das Kommando kann auch beim Wechsel eines SearchStore angewendet werden.

repair derivate search of type derivate

repair derivate search of ID {0} – die Kommandos lesen die gespeicherten Datenobjekte und reparieren zerstörte Volltext-Suchindizes. Das Kommando kann auch beim Wechsel eines SearchStore angewendet werden.

get last object ID for base {0}

get next object ID for base {0}

get next derivate ID for base {0}

show last object ID for base {0}

show next object ID for base {0} – die Kommandos liefern die nächste freie oder die letzte MCRObjektID für eine vorgegebene MCRObjektID-Basis zurück. Die Basis besteht aus Project- und Type-String in der Form *project_type*.²¹

check file {0} – prüft eine im Parameter {0} angegebene Datei mittels XML-Parser.

init hibernate – das Kommando initialisiert den Hibernate-SQL-Store. Das Kommando ist nur einmal erforderlich!

²¹ Siehe Abschnitt zur MCRObjektID.

Das SimpleWorkflow-System zur interaktiven Autorensarbeit

Das **SimpleWorkflow**-System wurde entwickelt, um mit einem einfachen Werkzeug die interaktive Autoren- und Editorenarbeit zu ermöglichen und damit eine sinnvolle Arbeit mit einer MyCoRe-Applikation zu ermöglichen. Es ist jedoch so konzipiert, dass es auch über eine Servlet-Schnittstelle in größere Workflow-Engines eingebunden werden kann. Einen Workflow im eigentlichen Sinne gibt es nur sehr eingeschränkt und in einfachem Ablauf. Weiterführende organisatorische Maßnahmen waren auch nicht Ziel dieser Entwicklung.

Die Komponente wurde in einen Modul verlagert und ist somit durch andere Komponenten ersetzbar. Eine genaue Beschreibung der Details zur Integration finden Sie im Programmer Guide. Die wichtigsten Merkmale dieses Moduls sind:

Mit dem System kann ein einfacher Eingabe- und Bearbeitungs-Dialog realisiert werden.

Eingabe und Bearbeitung werden durch eine Rechtekontrolle mittels des **ACL**-System realisiert. Nur berechtigte Benutzer dürfen die Daten manipulieren.

Die Zwischenspeicherung aller eingehenden Daten erfolgt zuerst auf einem Plattenbereich, so dass bei Fehlern ggf. auch der Administrator direkt eingreifen kann. Daten die erfolgreich in den Server geladen wurden, werden aus diesem Plattenbereich gelöscht.

Das System benutzt die MyCoRe-interne Editor-Komponente.

Das System basiert auf einer Reihe von Servlets, XML-Seiten und Stylesheets, sowie der Einbindung in die Editor-Formulare.

Alle Funktionen werden über ein einheitliches Servlet initiiert (MCRStartEditorServlet). Die möglichen Aufrufe sind weiter unten notiert.

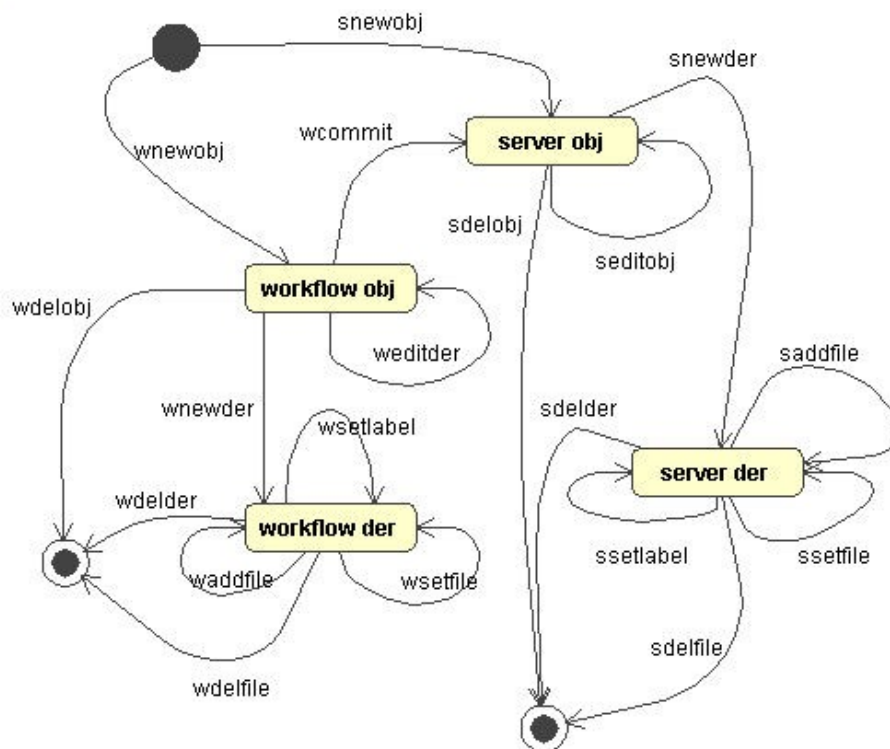


Abbildung 2: Funktionsschema des SimpleWorkflow

Das MCRStartEditorServlet

Dieses Servlet ist der Einstiegspunkt für die Nutzung des SimpleWorkflow-Systems. Von ihm aus werden alle Verarbeitungsprozesse angeschoben. Das Servlet seinerseits startet dann wieder Web-Dialoge oder führt selbstständig Aktionen aus. Dabei sind die folgenden Startparameter von Interesse:

Parameter	Bedeutung
todo	Zeigt an, welche Aktion auszuführen ist.
type	Gibt den Datenmodell-Typ des Metadaten-Objektes an.
step	Gibt den Verarbeitungsschritt an (z. B. author, editor, commit).
layout	Gestattet eine verfeinerte Angabe des Verarbeitungsschrittes (ist optional).
tf_mcrid	Enthält eine MCRObjektID, welche neu hinzugekommen und/oder dem System noch nicht bekannt ist. Die Gültigkeit wird geprüft.
se_mcrid	Enthält eine MCRObjektID, welche aus einem Datensatz oder ähnlichen Quellen extrahiert wurde und gültig sein sollte.
re_mcrid	Enthält eine weitere MCRObjektID, welche aus einem Datensatz oder ähnlichen Quellen extrahiert wurde und gültig sein sollte (z. B. zugehöriger Metdatensatz).
extparm	Erweiterungsparameter, wie in einigen wenigen Fällen benutzt.

Tabelle 2: Parameter des MCRStartEditorServlets

Die nächsten Tabellen sollen nun eine Übersicht der möglichen Aktionen geben. Jede Aktion ist dabei an eine entsprechende Permission gebunden, welche der aktuelle Benutzer gerade haben muss. Hat er das nicht, so wird seine Aktion abgewiesen und gelangt nicht zur Ausführung. Dabei wird noch nach dem Datenmodell-**type** unterschieden, d. h. ein Benutzer muss für genau diesen **type** auch die Berechtigung haben. Die Aktionen unterscheiden sich in dem Ziel-Store, *todo=w...* steht für den Plattenbereich; *todo=s...* arbeitet mit den bereits eingestellten Server-Daten. Der Parameter **layout** ist optional und dient der Verfeinerung der möglichen Arbeitsschritte. Während alle Aktionen, die mit einem **w** beginnen auf dem Plattenbereich (workflow) arbeiten, veranlassen alles Aktionen mit **s** einen Zugriff und Änderungen im Server-System.

Aktion	todo	ID	Permission	ruft
Anlegen neuer Metadaten	wnewobj	tf_mcrid	create- <i>type</i>	editor_form_ <i>type</i> _[<i>layout</i> _]author.xml
Anlegen eines Neuen Derivates	wnewder	se_mcrid	create- <i>type</i>	MCRStartEditorServlet?todo=waddfile
Hinzufügen neuer Dateien aus dem Upload	waddfile	se_mcrid re_mcrid	writewf	fileupload_new.xml
Bearbeiten von Metadaten	weditobj	se_mcrid	writewf	editor_form_ <i>type</i> _[<i>layout</i> _]editor.xml
Bearbeiten des Label eines Derivate-Metadaten-Satzes	weditder	se_mcrid re_mcrid	writewf	editor_form_derivate_editor.xml
Löschen aller Daten eines Objektes	wdelobj	se_mcrid	deletewf	editor_ <i>type</i> _editor.xml
Löschen eines Derivates	wdelder	se_mcrid	deletewf	editor_ <i>type</i> _editor.xml
Löschen einer Datei aus einem Derivate	wdelfile	se_mcrid	writewf	editor_ <i>type</i> _editor.xml
Setzen der Hauptdatei in einem Derivate	wsetfile	se_mcrid	writewf	editor_ <i>type</i> _editor.xml
Hochladen eines Datensatzes vom Plattenbereich zum Server	wcommit	se_mcrid	writedb	MCRQueryServlet mit der ID mit mode=ObjectMetadata

Tabelle 3: Mögliche Aktionen mit dem MCRStartEditorServlet auf dem Plattenbereich

Aktion	todo	ID	Permission	ruft
Bearbeiten der Metadaten	seditobj	se_mcrid	writedb	MCRQueryServlet mit der ID mit mode=ObjectMetadata
Bearbeiten des Label der Derivate-Metadaten	seditder	se_mcrid re_mcrid	writedb	editor_form_commit_derivate.xml
Löschen eines Datenobjekts	sdelobj	se_mcrid	deletedb	editor_deleted.xml
Löschen eines Derivates von einem Datenobjekt	sdelder	se_mcrid re_mcrid	deletedb	MCRQueryServlet mit der ID mit mode=ObjectMetadata
Hinzufügen eines neuen Derivates zu einem Datenobjekt des Servers; Zwischenablage der Daten auf dem Plattenbereich	snewder	re_mcrid	writedb	MCRStartEditorServlet?todo=saddfile
Hinzufügen von Daten zu einem Derivate aus dem Server; Zwischenablage der Daten auf dem Plattenbereich	snewfile	se_mcrid re_mcrid	writedb	MCRStartEditorServlet?todo=saddfile
Upload von Datenobjekten in die Zwischenablage	saddfile	se_mcrid re_mcrid	writedb	MCRStartEditorServlet?todo=scommitder
Setzen Des Label in einem Derivate	ssetlabel	se_mcrid re_mcrid	writedb	MCRQueryServlet mit der ID mit mode=ObjectMetadata
Setzen der Main File Markierung im Derivate	ssetfile	se_mcrid re_mcrid	writedb	MCRFileNodeServlet mit der ID des Derivates
löschen einer Datei aus einem Derivate	sdelfile	se_mcrid re_mcrid	writedb	MCRFileNodeServlet mit der ID des Derivates

Tabelle 4: Mögliche Aktionen mit dem MCRStartEditorServlet im Server

Abläufe für neue Datenobjekte

Die Abläufe für die Eingabe neuer Datensätze sind praktisch für alle Datenmodelle gleich. Lediglich die Anbindung der Derivate an die Metadaten-Objekte ist nicht immer gegeben. Das hängt allein an

der Gestaltung des jeweiligen Datenmodell-Konzeptes für ein Projekt (z. B. haben Personendaten im DocPortal-Projekt keine eigenen Derivate). Wird beim SimpleWorkflow ein neues Objekt eingestellt, so befinden sich alle relevanten Daten vorerst auf einem Plattenbereich, der über die Konfiguration festgelegt wird. Erst wenn das **Commit** (Laden) zum Server-System ausgeführt wurde, werden die Daten von diesem Zwischenspeicher wieder gelöscht. Jeder Datenmodell-*type* hat dabei in der Regel ein eigenes Verzeichnis innerhalb des Workflow-Plattenbereiches.

Abläufe für Datenobjekte aus dem Server

Wurden Datenobjekte in den Server eingebracht, so steht Benutzern, welche berechtigt sind, die Möglichkeit einer Änderung der Daten und/oder das Löschen der selben frei. Für das Bearbeiten der Daten werden diese zwischenzeitlich auf dem Plattenbereich gespeichert. Bei erfolgreicher Beendigung einer Aktion werden die temporären Daten wieder vom Plattenbereich gelöscht. Im Falle eines Fehlers kann über den Zugriff auf den Plattenbereich (Workflow) und entsprechender Aktionen der Fehler behoben werden. Alle Commits sind als Update ausgelegt, so dass ältere Versionen im Server auch bei einem Commit vom Workflow als Folge eines Fehlers überschrieben werden. Einzelheiten zu den Abläufen finden Sie im Programmer Guide.

Einbindung in das SimpleAccessControl-System

In den Bearbeitungsstufen gibt es jeweils einen Button, welcher es gestattet, die ACL's (Access Control Lists – Zugriffslisten) zu ändern. Dies geschieht über zusätzliche Eingabemasken. Alle Änderungen wirken direkt und sofort.

5.1 Der Klassifikationseditor

Eine Klassifikation in MyCoRe ist eine hierarchische festgeschriebene Darstellung von Kategorien, welche nach bestimmten Ordnungsprinzipien und Eigenschaften gestaltet ist.














Die Menge der Klassen/Kategorienamen bilden ein kontrolliertes Vokabular, das es ermöglicht Dokumente nach den Kriterien der Klassifikation zu ordnen. In Bibliotheken spielen Klassifikationen eine große Rolle.

In den Metadaten der Dokumente können beliebige Klassifikationen und Kategorien dem Dokument zugeordnet werden.

Typische Klassifikationen wie Dokumentformate, Dokumententyp, Herkunft für die formale Zuordnung von Dokumenten aber auch DDC und Pacs für die inhaltliche Klassifizierung werden von MyCoRe im Sample bereits mitgeliefert.

Umgekehrt bietet MyCoRe mit dem Klassifikationsbrowser die Möglichkeit über Klassifikationen zu navigieren und so zu den zugehörigen Dokumenten zu gelangen.

Syntax und Beschreibung von Klassifikationen wurden bereits in Kapitel 5.1.2 dargestellt.

<i>Rostock</i>		
[__14 Dok.]	Fakultäten und Institute [atlibri_class_00000003] <i>Liste und Struktur der Fakultäten und Institute der Universität Rostock</i>	 
[__12 Dok.]	Sprachen [atlibri_class_00000004] <i>Klassifikation über die Sprachen die in Suchbegriffen verwendet wurden</i>	 
[__17 Dok.]	Dokumententypen [atlibri_class_00000005] <i>Klassifikation der Dokumententypen</i>	 
[__14 Dok.]	Formate [atlibri_class_00000006] <i>Klassifikation der Formate</i>	 
[__0 Dok.]	Sachgruppen DNB [atlibri_class_00000007] <i>Sachgruppen der Deutschen Nationalbibliographie</i>	  
[__3 Dok.]	Archivierungsklassen [atlibri_class_00000008] <i>Klassifikation zu Evaluierung und Archivierung</i>	 

5.2 Start des Klassifikationseditors

Der Editor wird über den Klassifikationsbrowser mit dem Parameter `mode=edit` gestartet.

Bsp:

<http://localhost:8080/docportal/browse?mode=edit>

Erfüllen die Privilegien des Nutzers die Anforderungen, erhält man zunächst eine Liste aller im System installierten Klassifikationen, also auch die, die nicht über den Klassifikationsbrowser eingebunden sind. Die nicht eingebundenen Klassifikationen werden mit den Parametern aus dem Default-Block gesteuert!

5.3 Operationen des Klassifikationseditors

Achtung: Klassifikationen werden im Editiermodus grundsätzlich unsortiert angezeigt, da sonst die Verschiebeoperationen für Kategorien nicht sinnvoll nutzbar sind.

5.4 Klassifikationen

Neue Klassifikation erstellen

Über den Button [neue Klassifikation erstellen] kann eine Klassifikation angelegt werden. Dazu ist es notwendig mindestens ein Label für die Klassifikation anzugeben. Es können Label und zugehörige Beschreibungen in verschiedenen Sprachen angelegt werden, sowie eine URL. Die ID der Klassifikation wird vom System vergeben.

Klassifikation importieren

Über den Button [Klassifikation importieren] kann eine neue Klassifikation vom Dateisystem in MyCoRe importiert werden, oder eine bereits im System vorhandene Klassifikation aktualisiert werden. Ist der Import der Klassifikation nicht erfolgreich (z.B. XML ist nicht valide, oder die zu aktualisierende Klassifikation existiert nicht im System) erfolgt eine Fehlermeldung und die Klassifikation wird nicht importiert.

Bei Erfolg wird automatisch wieder auf die Liste aller im System vorhandenen Klassifikationen zurückgesprungen.

Export der Klassifikation



Mit dieser Funktion kann die ausgewählte Klassifikation in einem Extra-Browserfenster im XML-Format angezeigt werden oder auch über die rechte Maustaste auf dem lokalen Dateisystem gespeichert werden.

Editieren der Klassifikation



Die Beschreibungsdaten der ausgewählten Klassifikation können über ein Editorformular geändert werden. Es können weitere Label in anderen Sprachen hinzugefügt werden. Die ID kann nicht verändert werden. Ein Label muss für die Klassifikation mindestens belegt sein.

Löschen der Klassifikation



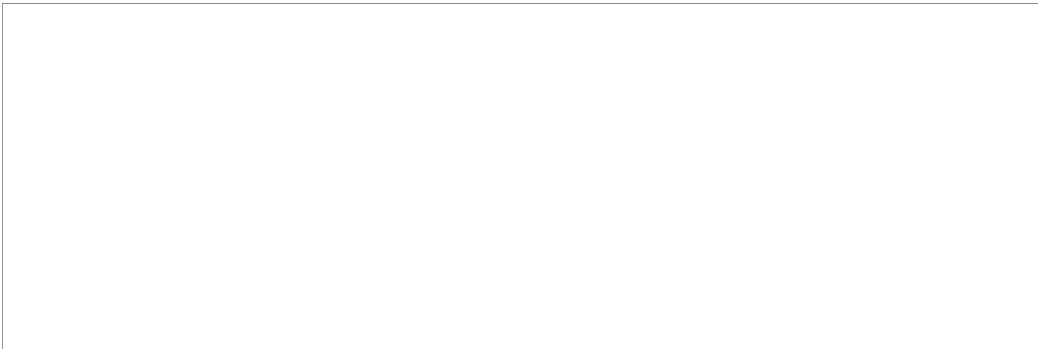
Die ausgewählte Klassifikation wird, inklusive der eventuell vorhandenen Kind-Kategorien gelöscht, nachdem geprüft wurde ob es keine Referenzen zu Objekten gibt. Im Browser ist das durch die Anzeige der Dokumentanzahl bereits sichtbar. Klassifikationen, bei denen eine Anzahl > 0 gezählt wurde besitzen daher keinen [Delete] Button.

Im Erfolgsfall wird wieder die aktuelle Klassifikationsliste ohne das gelöschte Klassifikation - Item angezeigt.

5.5 Kategorien

Beim Klick auf eine der aufgelisteten Klassifikationen wird die erste Ebene der Klassifikation mit den zugehörigen Kategorien geöffnet.

Die aktuell manipulierte Kategorie wird zur besseren Übersicht farblich hervorgehoben.



Für Kategorien gibt es folgende Aktionsmöglichkeiten:

Neue Kategorie anlegen



Unter der ausgewählten Kategorie wird eine neue Kategorie angelegt. Dazu wird ein Editorformular für die Angaben von ID, Label und Beschreibung geöffnet. Die Felder sind zunächst mit den Werten der Vorgängerkategorie (also der Ausgewählten unter der die neue Kategorie angelegt werden soll.) belegt.

Beim Abspeichern wird die Eindeutigkeit einer Kategorie - ID für die Klassifikation geprüft. Ist sie nicht eindeutig erfolgt eine Fehlermeldung. Im Erfolgsfall wird wieder die aktuelle Klassifikationsebene mit dem neuen Kategorie - Item angezeigt.

Editieren der Kategorie



Die ausgewählte Kategorie wird im das Editorformular geladen. Die Angaben von ID, Label und

Beschreibung können manipuliert werden.

Beim Abspeichern wird die Eindeutigkeit einer Kategorie - ID für die Klassifikation geprüft. Ist sie nicht eindeutig erfolgt eine Fehlermeldung. Im Erfolgsfall wird wieder die aktuelle Klassifikationsebene mit dem modifizierten Kategorie - Item angezeigt.

Die Kind-Knoten, die an der Kategorie möglicherweise hängen werden nicht geändert.

Löschen der Kategorie



Die ausgewählte Kategorie wird, inklusive der eventuell vorhandenen Kind-Kategorien gelöscht, nachdem geprüft wurde ob es keine Referenzen zu Objekten gibt. Im Browser ist das durch die Anzeige der Dokumentanzahl bereits sichtbar. Kategorien, bei denen eine Anzahl > 0 gezählt wurde besitzen daher keinen Delete - Button.

Im Erfolgsfall wird wieder die aktuelle Klassifikationsebene ohne das gelöschte Kategorie - Item angezeigt.

Verschieben der Kategorie



Eine Kategorie kann in der Hierarchie im Baum verschoben werden. Je nach Position im Baum sind die Verschiebemöglichkeiten Aufwärts, Abwärts, nach Links und Rechts möglich.

Auf- und Abwärts Verschieben verändert die Position der jeweils gewählten Kategorie um 1 Stelle in der aktuellen Ebene.

Die *Rechts*-Verschiebung bewirkt, dass die Kategorie zur Kind-Kategorie der darüber positionierten Kategorie wird. Dabei werden auch alle in der ausgewählten Kategorie eventuell vorhandenen Kindelemente mit verschoben. Die Kategorie wird quasi in die darüber positionierte Kategorie hinein geschoben und dort an das Ende dieser Ebene angehängt, also eine Ebene tiefer angelegt.

Die *Links*-Verschiebung bewirkt das Gegenteil der Rechts-Verschiebung, dass die Kategorie aus der Ebene in die darüber liegende Ebene geschoben wird. Dabei werden auch alle in der ausgewählten Kategorie eventuell vorhandenen Kindelemente mit verschoben. Die Kategorie wird quasi in die darüber liegende Ebene geschoben und dort an das Ende der Ebene angehängt.

6 Möglichkeiten zur Nutzung optionaler Komponenten

MyCoRe bzw. DocPortal bietet neben den in der Standardinstallation verwendeten Komponenten noch eine Reihe von optionalen Zusätzen. Auch der Einsatz von anderer externer Software wie Tomcat / Apache / Speicher-Backends u. v. m. soll in diesem Kapitel besprochen werden.

Die Zusammenarbeit mit anderen DocPortal-Installationen

Das MyCoRe-System ist so konzipiert, dass hinsichtlich der Metadaten gleichartige Installationen miteinander arbeiten und von einer gemeinsamen Oberfläche (Frontend) abgefragt werden können. Hierzu müssen die Remote-Instanzen definiert werden. Auch die eigene Installation kann über diesen Weg abgefragt werden. Das ist in der Beispielinstallation auch schon vorgesehen. Die Remote-Suche nutzt dabei die WebServices Komponenten von MyCoRe. Das installierte DocPortal-Sample gestattet grundsätzlich auch eine Suche über den webservice-Dienst gegen den **localhost**.

Die Konfiguration ist denkbar einfach, es sind alle Hosts, für die eine Remote-Abfrage angeboten werden soll, in die Datei `DOCPORTAL_HOME/config/hosts.xml` einzutragen. Nach einem `ant webapps` sind die Hosts dann mittels der Suchmasken durchsuchbar.

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

<hosts xmlns="http://www.mycore.org/">
  <host alias="remote" url="http://localhost:8291/services/MCRWebService">
    <label xml:lang="de">Lokal via Webservice</label>
    <label xml:lang="en">Local via Webservice</label>
  </host>
  <host alias="leipzig" url="http://mycoresamplelinux.dl.uni-leipzig.de/services/MCRWebService">
    <label xml:lang="de">Universität Leipzig</label>
    <label xml:lang="en">University of Leipzig</label>
  </host>
</hosts>
```

Von den Entwicklern des MyCoRe-Projektes wird exemplarisch eine DocPortal-Installation bereitgehalten. Diese ist im Konfigurationsfile `hosts.xml` notiert und sollten in der Regel verfügbar sein.

Alias	URL	Port	Standort
remote	http://localhost:8291/	8291	lokale Installation
leipzig	http://mycoresamplelinux.dl.uni-leipzig.de/	8291	Uni Leipzig

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

Nutzung der OAI Schnittstelle

6.1 Grundlagen

Die Open Archives Initiative²² 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 werden²³. 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.

6.2 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 **Sets** 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.

6.3 Installation

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

6.4 Der Deployment Descriptor

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

²² <http://www.openarchives.org/>

²³ <http://www.openarchives.org/OAI/openarchivesprotocol.html>

```

<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 3: Einbindung des OAI-Data-Providers in web.xml

6.5 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.

6.6 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 Miles ist dies `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, hierbei muß aber auch jeweils ein Namensraum und ein Schema angegeben werden, z. B.

MCR.oai.metadata.transformer.xmetadiss=MyCoReOAI-mycore2xmetadiss.xsl

MCR.oai.metadata.namespace.xmetadiss=http://www.ddb.de/standards/xMetaDiss/

MCR.oai.metadata.schema.xmetadiss=http://www.ddb.de/standards/xmetadiss/xmetadiss.xsd

Diese Stylesheets benutzen als Eingabe das Ergebnis des ersten Stylesheets.

6.7 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 Identifikator des OAI-Repositories (wird vom Harvester abgefragt).
- MCR.oai.setscheme.OAIDataProvider=DocPortal_class_00000006
DocPortal_class_00000005 **(notwendig)** Die MyCoRe-Klassifikation, die zur Bildung der Struktur des OAI-Repositories verwendet wird.
- MCR.oai.dini-mapping.doc-type.text=FORMAT0001 **(optional)** Mapping der Klassifikations-CategoryIDs auf international übliche Bezeichnungen in jeweils einer eigenen Zeile. Hier erscheint im OAI-Result „doc-type:text“ statt „FORMAT0001“. Beachte hierzu die DINI-Empfehlungen zu OAI:
<http://www.dini.de/documents/OAI-Empfehlungen-Okt2003-de.pdf>
- 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.

6.8 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.

6.9 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.

Einbindung virtueller Host-Namen mit Hilfe des Apache-Web-Servers

Dieses Kapitel bezieht sich auf die SuSE 9.2 Distribution. Für andere Linux-Systeme sind ggf. kleine Änderungen erforderlich.

Standardmäßig ist der Apache2 ohne Einbindung der Proxy-Module in den Installations-CD's enthalten. Soll die Proxy-Funktionalität genutzt werden, dann ist die Neucompilierung der Quellen von Apache2 erforderlich. Der Quellcode des Apache2 liegt auf <http://httpd.apache.org> für ein Download bereit. Die aktuelle Version ist **httpd-2.0.54**.

Für die Übersetzung des Apachen2 sind noch die *apr* und *apr-util* Komponenten erforderlich. Diese sind nicht Standardmäßig in den Installations-CD's von SuSE enthalten. Die Versionen *apr-1.1.1* und *apr-util-1.1.2* stehen unter <http://httpd.apache.org> als Tar/Zip-Files zur Verfügung. Im Quellverzeichnis von **httpd-2.0.54** sind die *apr* und *apr-util* Quellen der Version 0.9.6 enthalten.

6.1 Installation von httpd-2.0.54 mit Einbindung von mod_proxy

Entpacken des `httpd-2.0.54.tar.gz` in ein Arbeitsverzeichnis.

- `tar -xf httpd-2.0.54.tar`
- `cd httpd-2.0.54`
- `./configure --enable-proxy --enable-proxy-connect --enable-proxy-ftp --enable-proxy-http`
- `make`
- `make install` (installiert neuen Apache2 standardmäßig unter `/usr/local/apache2`)

Soll nun diese neue Version des Apache2 immer bei einem Neustart aktiviert wird, muss das Skript `/usr/local/apache2/bin/apachectl` nach `/etc/init.d/apache2` kopiert werden, oder es ist ein entsprechender Link zu setzen.

Zur Kontrolle der Übersetzung können Sie mittels des Kommandos

`/usr/local/apache2/bin/httpd -l` die Einbindung der Proxy-Module testen. Die Auflistung muss die beiden Module *apr* und *apr-util* anzeigen.

6.2 Die Verbindung von Tomcat5 und Apache2

Die Verbindung zwischen dem Apache2 und Tomcat5 wird in den Konfigurationsfiles `/usr/local/apache2/httpd.conf` und der `server.xml` von der Tomcat-Anwendung konfiguriert. Es wird ein virtueller Host in der `httpd.conf` definiert.

```
<VirtualHost mycoresample.dl.uni-leipzig.de:80>

    ProxyPass / http://mycoresample.dl.uni-leipzig.de:8291/
    ProxyPassReverse / http://mycoresample.dl.uni-leipzig.de:8291/

    ...

</VirtualHost>
```

Abbildung 4: Ausschnitt der `httpd.conf`

Die folgenden Änderungen basieren auf den im Laufe der Installation benutzten Tomcat5 Konfiguration, wie Sie in Kapitel 3 beschrieben ist.

```
<Service name="MYCORESAMPLE-Standalone">

    <!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
    <Connector port="8291"
        maxThreads="200" minSpareThreads="25" maxSpareThreads="150"
        enableLookups="false" redirectPort="8292" acceptCount="800"
        debug="0" connectionTimeout="2000000"
        bufferSize="67440000" socketBuffer="-1"
        URIEncoding="UTF-8"
        proxyName="mycoresample.dl.uni-leipzig.de" proxyPort="80"
        disableUploadTimeout="true" />

    ...
</Service>
```

Abbildung 5: Änderungen in der server.xml

Nach dem Neustart von Tomcat5 und Apache2 sollte das System nun über die virtuelle Adresse ansprechbar sein.

7 Weiterführende Informationen zum Aufbau von MyCoRe-Anwendungen

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 Programmier Guide. In den folgenden Abschnitten wird lediglich auf die XML-Syntax der Daten eingegangen.

7.1 Die MCRObjektID

Die Identifikation eines jeden MyCoRe Objektes erfolgt grundsätzlich über eine eindeutige ID. Die ID kann per Hand vergeben oder auch automatisch via API generiert werden. Diese hat für alle Objekte einen einheitlichen Aufbau, dessen Inhalt für jedes Projekt und jeden Datentyp festzulegen ist:

ID = "projektkürzel_type_nummer"

<i>projektkürzel</i>	Dieses Element ist für ein Projekt und/oder eine Einrichtung / Datengruppe festzulegen, zum Beispiel <i>UBLPapyri</i> oder <i>MyCoReDocument</i> . In MyCoRe wird es teilweise auch zur Identifikation einzelner Konfigurationsdaten mit genutzt.
<i>type</i>	Das Element beschreibt den Datenmodelltyp, d. h. der <i>type</i> verweist auf die zugehörige Datenmodell-Konfiguration, zum Beispiel <i>datamodel-author</i> oder <i>datamodel-document</i> . In MyCoRe wird es oft zur Identifikation einzelner Konfigurationsdaten im Zusammenhang mit der Verarbeitung dieses Datenmodells genutzt.
<i>nummer</i>	Ist eine frei wählbare positive Integerzahl. Diese Zahl kann in Verantwortung des Projektmanagers per Hand oder automatisch vergeben werden. Bei der Projektdefinition wird die Größe des Zahlenbereiches festgelegt. Es hat sich als sinnvoll erwiesen, nicht weniger als 8 Ziffern einzuplanen.

Tabelle 6: Aufbau der MCRObjektID

Im MyCoRe-Projekt sind zwei MCRObjektID-Typnamen reserviert und dürfen nicht für anderweitige Objekte genutzt werden. Der Typ **class** steht für Klassifikationen, der Typ **derivate** wird für Multimediaobjekte verwendet.

Es sei noch einmal ausdrücklich darauf hingewiesen, dass die MCRObjektID eine zentrale Rolle im ganzen MyCoRe-Projekt spielt. Über sie werden alle Daten identifiziert und referenziert. Es sind daher die vorgegebenen Regeln streng einzuhalten. Da es derzeit für den Datentyp zum anhängen nur eine *type*-Bezeichnung gibt, kann es beim Design eines Projektes hilfreich sein, sich für eine Gruppe von Projektkürzeln zu entscheiden, z. B. *DOLAuthor_author_...*, *DOLDocument_document_...* usw. So kann jedem Datenmodell eine dedizierte Derivate-Gruppe zugeordnet werden z. B. *DOLAuthor_derivate_...* oder *DOLDocument_derivate_...*. Diese Trennung ist nicht zwingend, hat sich aber bei der Verwaltung großer Datenmengen als günstig erwiesen. Manchmal ist es sogar sinnvoll, hierzu noch mehrere Projektkürzel für ein Datenmodell zu verwenden, je nach Umfang des Datenbestandes und der Sicherungs- und Reparatur-Strategien des Projektes.

7.2 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 DocPortal 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-IDs mit verwenden, so planen Sie entsprechende führende Nullen ein, andernfalls wird das Suchergebnis fehlerhaft! Weiterhin ist es sehr zu empfehlen, dieser Zahlenfolge einen Buchstaben voran zusetzen, 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.

```

<?xml version="1.0" encoding="UTF-8" ?>
<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>
</categories>
</mycoreclass>

```

Abbildung 6: XML-Syntax eines Klassifikations-Objektes

7.3 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 der MyCoRe-Beispielanwendung 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.

XML-Syntax eines Metadatenobjektes

```
<?xml version="1.0" encoding="UTF-8" ?>
<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 7: XML-Syntax eines Metadaten-Objektes

- Für **xsi:noNamespaceSchemaLocation** ist das entsprechende XMLSchema-File des Metadatentyps anzugeben (document.xsd)
- Die **ID** ist die eindeutige MCRObjektID.
- 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!**

XML-Syntax 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 beschrieben.

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 Vererbungs-bäumen und kann durch den Anwender 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 8: XML-Syntax eines structure XML-Knotens

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 `parasearch` für die Einbindung in die parametrische Suche und `textsearch` für die Volltext-Suche über das gesamte Metadatenobjekt angegeben werden. Es sind jeweils die booleschen 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 eingeschlossenen 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.

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

Abbildung 9: XML-Syntax eines metadata XML-Knotens

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

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	MCRMetaAccessRule
MCRMetaClassification	MCRMetaAddress
MCRMetaISBN	MCRMetaHistoryDate
MCRMetaISO8601Date	MCRMetaInstitutionName
MCRMetsLangText	MCRMetaPersonName
MCRMetaLink	MCRMetaIFS
MCRMetaLinkID	MCRMetaXML
MCRMetaNBN	
MCRMetaNumber	

Tabelle 7: MyCoRe-Basisdatentypen

XML-Syntax des Metadatentyps MCRMetaAccessRule

Der Basistyp MCRMetaAccessRule ist für den Import/Export von Access Control Lists (ACL's)

gedacht. Der Metadatentyp stellt einen Container für die entsprechenden Access-Conditions dar und wird nur im MCRService-teil verwendet.

```
<servacl class="MCRMetaAccessRule">
  <servacl permission="...">
    <condition format="xml">
...
    </condition>
  </servacl>
</servacl>
```

```
<servacl class="MCRMetaAccessRule">
  <servacl permission="read">
    <condition format="xml">
      <boolean operator="and">
        <!-- USER OR GROUP -->
        <boolean operator="or" />
        <!-- DATE -->
        <boolean operator="and" />
        <!-- IP -->
        <boolean operator="or" />
        <!-- -->
      </boolean>
    </condition>
  </servacl>
</servacl>
```

XML-Syntax des Metadatentyps 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.

```

<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>

```

Abbildung 11: Beispiel des Metadatentyps MCRMetaAddress

XML-Syntax des Metadatentyps 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="...">
    ...
  </subtag>
  ...
</tag>

```

Abbildung 12: XML-Syntax des Metadatentyps 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 13: Beispiel des Metadatentyps MCRMetaBoolean

XML-Syntax des Metadaten-Basistyps MCRMetaClassification

Der Basistyp MCRMetaClassification dient der Einbindung von Klassifikationen²⁴ und deren 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. **Bitte beachten Sie die Hinweise zur Gestaltung der Kategorie-IDs im vorigen Kapitel!**

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

Abbildung 14: XML-Syntax des Metadaten-Basistyps MCRMetaClassification

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

Abbildung 15: Beispiel des Metadaten-Basistyps MCRMetaClassification

XML-Syntax des Metadaten-Basistyps MCRMetaHistoryDate

Der Basistyp MCRMetaHistoryDate ist speziell kreiert, um Datumsangaben für historische Projekte speichern und suchen zu können. dabei wird sowohl ein verbaler Text wie eine konkrete Datumskonvertierung gespeichert. Das Datum wird im gregorianischen Kalender abgelegt, auch für die Zeit vor Einführung des selben. Somit erreicht man eine eindeutige Datumsangabe, die auch vor Christi Geburt funktioniert. Diese Datumsangabe wird intern in Integerwerte ab 4000 v. Chr. umgerechnet. Die Formel der Integer-Werte ist

$$\text{ivon/ibis} = 4000 + 10000 * \text{Jahr} + 100 * \text{Monat} + \text{Tag}$$

Somit ist eine scharfe Datumssuche mit Hilfe der Integer-Daten möglich. Die Eingabe der Daten erfolgt nach den Regeln:

- Im **text**-Feld steht ein beliebiger String gemäß den Projektvorgaben
- Die Felder **von** und **bis** enthalten gregorianische Datumsangaben.
- Ist für **von** und/oder **bis** nichts angegeben, werden Standardwerte genommen. Das sind 1.1.3000 v. Chr. und 31.12.3000 n. Chr. Die Werte können auch per Konfiguration eingestellt werden:

²⁴siehe voriges Kapitel

MCR.history_date_min und **MCR.history_date_max**.

- Mögliche Notationen für die Datumsangaben sind 01.01.1999 / -01.12.200 / 1035 / -133 .
- Die Felder **ivon** und **ibis** werden automatisch gesetzt.

```
<tag    class="MCRMetaHistoryDate"    heritable="..."    parasearch="..."
textsearch="...">
  <subtag type="..." xml:lang="...">
    <text>...</text>
    <von>...</von>
    <ivon>...</ivon>
    <bis>...</bis>
    <ibis>...</ibis>
    <von>...</von>
  </subtag>
  ...
</tag>
```

Abbildung 16: Syntax des Metadaten-Basistyps MCRMetaHistoryDate**XML-Syntax des Metadatentyps MCRMetaInstitutionName**

Der Basistyp MCRMetaInstitutionName beinhaltet eine Liste der Namen einer Firma oder Einrichtung oder eines Bereiches der selben. Dabei soll berücksichtigt werden, dass die Name in verschiedenen Sprachen und in international gängigen Formen gespeichert werden sollen. Über das Attribut **type** ist eine zusätzliche Differenzierung der verschiedenen Namen möglich.

- **name** beinhaltet den vollständigen Namen (Pflicht)
- **nickname** das Pseudonym (z. B. UBL) (optional)
- **property** den rechtlichen Stand, GmbH (optional)

Das optionale Attribut textsearch bei diesem Typ bewirkt nur die Speicherung des Namens als Metadaten-Volltext.

```
<tag class="MCRMetaInstitutionName" heritable="..." parasearch="..."
textsearch="...">
  <subtag xml:lang="...">
    <fullname>...</fullname>
    <nickname>...</nickname>
    <property>...</property>
  </subtag>
  ...
</tag>
```

Abbildung 17: Syntax des Metadaten-Basistyps MCRMetaInstitutionName

XML-Syntax des Metadatentyps MCRMetaISBN

```
<tag class="MCRMetaISBN" heritable="..." parasearch="...">
  <subtag>ISBN</subtag>
</tag>
```

Abbildung 18: Syntax des Metadaten-Basistyps MCRMetaISBN

Dieser Metadatentyp ist ganz speziell zur Speicherung einer ISBN gedacht. Er gestattet nur eine Kardinalität.

XML-Syntax des Metadatentyps MCRMetaISO8601Date

Dieser Metadatentyp ist wie MCRMetaDate für das Speichern von Zeitangaben gedacht. Er bietet jedoch eine höhere zeitliche Auflösung, bis in den Millisekundenbereich. Unterstützt werden alle Formate der Informationsseite des W3C (<http://www.w3.org/TR/NOTE-datetime>). Sie enthält nähere Informationen zu den Formaten und zur ISO-Norm: ISO 8601 : 1998 (E)

Wie MCRMetaDate unterstützt MCRMetaISO8601Date die Verwendung des type-Attributs, auf Grund seiner Sprachunabhängigkeit in der Formatierung der Datumsangabe fehlt die Unterstützung für das lang-Attribut. Das Verwenden von MCRMetaISO8601Date ermöglicht eine Syntaxprüfung der Datumsangabe bereits auf XMLSchema-Ebene, durch den dort definierten Datentyp xsd:duration, auf dem der MyCoRe-Datentyp abgebildet wird.

Optional kann ein format-Attribut verwendet werden. Dies erzwingt für das Datum das angegebene Format. So ist bei der Formatangabe „YYYY“ das Datum „2006-01“ ungültig. Ohne die Formatangabe hingegen ist das gleiche Datum gültig, weil es dem unterstützten Format „YYYY-MM“ entspricht.

```
<tag class="MCRMetaISO8601Date" heritable="..." parasearch="...">
  <subtag type="..." format="...">YYYY-MM-DDThh:mm:ss.sTZD</subtag>
</tag>
```

Abbildung 19: Syntax des Metadaten-Basistyps MCRMetaISO8601Date

```
<dates class="MCRMetalSO8601Date" heritable="false" parasearch="true">
  <date type="sample">2006-01-16T13:20:30.85+01:00</date>
</dates>
```

Abbildung 20: Beispiel des Metadaten-Basistyps MCRMetalSO8601Date

XML-Syntax des Metadatentyps MCRMetalLangText

Der Basistyp MCRMetalLangText 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 spezifiziert werden, in welcher Form der Text geschrieben ist.

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

Abbildung 21: XML-Syntax des Metadaten-Basistyps MCRMetalLangText

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

Abbildung 22: Beispiel des Metadaten-Basistyps MCRMetalLangText

XML-Syntax der Metadatentypen MCRMetalLink und MCRMetalLinkID

Der Basistyp MCRMetalLink wurde geschaffen, um eine Verknüpfung verschiedener MyCoRe-Objekte untereinander zu realisieren. Außerdem können hier genauso Verweise auf beliebige externe Referenzen abgelegt werden. Der Typ MCRMetalLink ist eine Implementation des W3C XLink Standards²⁵. Auf dieser Basis enthält der MyCoRe-Metadatentyp zwei Arten von Links - eine Referenz und einen bidirektionalen Link. Bei beiden werden jedoch in MCRMetalLink 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

²⁵ siehe 'XLM Linking Language (XLink) Version 1.0'

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 DocPortal 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 in der Beispielanwendung nichts über den Verweis.

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 MCRObjectIDs 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 23: 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 24: Beispiel des Metadaten-Basistyps MCRMetaLink

XML-Syntax des Metadatentyps MCRMetaNBN

Diese Metadatentyp ist ganz speziell zur Speicherung einer NBN gedacht. Er gestattet nur eine Kardinalität.

```
<tag class="MCRMetaNBN" heritable="..." parasearch="...">
  <subtag>NBN</subtag>
</tag>
```

Abbildung 25: Syntax des Metadaten-Basistyps MCRMetaNBN

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="..." measurement="...">
    ...
  </subtag>
  ...
</tag>
```

Abbildung 26: XML-Syntax des Metadaten-Basistyps MCRMetaNumber

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

Abbildung 27: Beispiel des Metadaten-Basistyps MCRMetaNumber

XML-Syntax des Metadatentyps MCRMetaPersonName

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, Kosenamen usw. **firstname** repräsentiert den/die Vornamen, **callname** den Rufnamen, **surname** den Familiennamen, **academic** den akademischen Titel und **peerage** den Adelstitel und **prefix** Namenszusätze wie 'von', 'de' usw. **fullname** enthält nochmal den automatisch zusammengesetzten Namen. 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="MCRMetaPersonName" heritable="..." parasearch="...">
  <subtag type="..." xml:lang="..">
    <firstname>...</firstname>
    <callname>...</callname>
    <surname>...</surname>
    <fullname>...</fullname>
    <academic>...</academic>
    <peerage>...</peerage>
    <prefix>...</prefix>
  </subtag>
  ...
</tag>

```

```

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

```

Abbildung 29: Beispiel des Metadaten-Basistyps MCRMetaPersonName

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 viel 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 30: XML-Syntax des Metadaten-Basistyps MCRMetaXML

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

Abbildung 31: Beispiel des Metadaten-Basistyps MCRMetaXML

XML-Syntax des XML-Knotens service

Für die Einrichtung eines Workflow und um die Wartung großer Datenmengen zu vereinfachen sowie für den Import / Export der ACL's wurde der XML-Abschnitt service in das Metadatenobjekt integriert. Hier sind Informationen wie Datumsangaben, ACL's 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 MCRMetaISO8601Date. 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.

Die servacIs enthalten Access Control System Conditions für die genutzten Permissions wie **read**, **writedb** oder **deletedb**. Eine genaue Beschreibung ist dem Kapitel über ACL's des Programmer Guide zu entnehmen.

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

theoretisch unbegrenzt.

```
<service>
  <servdates class="MCRMetaISO8601Date">
    <servdate type="...">...</servdate>
    ...
  </servdates>
  <servacls class="MCRMetaAccessRule">
    <servacl permission="...">
      ...
    </servacl>
  </servacls>
  <servflags class="MCRMetaLangText">
    <servflag>...</servflag>
    ...
  </servflag>
</service>
```

Abbildung 32: XML-Syntax des service XML-Knotens

7.4 Das Speichermodell für die Multimediataten (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" encoding="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 33: XML-Syntax des Derivate-Datenmodells

- Für **xsi:noNamespaceSchemaLocation** ist die entsprechende XML Schema-Datei anzugeben (Derivate.xsd)
- Die **ID** ist die eindeutige MCRObjektID.
- 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 MCRObjektID 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.

8 Hints & Tips / Troubleshooting

Bekannte Probleme

Diese Abschnitt wird erst nach Freigabe des endgültigen Release MyCoRe 1.3 gefüllt.