



User Guide

RELEASE 2.0.2

7. Februar 2008

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 Helmbrecht-Schaar, Robert Stephan (Rostock)

Heiko Helmbrecht (München)

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. Im abschließenden Kapitel *Tipps und Problembeseitigung* wird 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-Webseiten 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.

Hinweis:

Diese Dokumentation basiert auf dem Release 2.0.1 der MyCoRe-Software. Um die genannten Features nutzen zu können, benutzen Sie bitte diesen Codestand inklusive der dazugehörigen Fixes!

Inhaltsverzeichnis

1	Allgemeines zu MyCoRe.....	1
1.1	Vorbemerkungen.....	1
1.1.1	Betriebssysteme.....	1
1.1.2	Relationale Datenbanken.....	1
1.1.3	Suchindex.....	1
1.1.4	Textextraktion.....	2
1.1.5	Streaming Server.....	2
1.1.6	Webanwendungs-Server.....	2
1.1.7	Web-Server.....	2
1.2	Die Beispielanwendung.....	3
1.3	Installationswege.....	3
2	Allgemeine Umgebungs- und Softwarevoraussetzungen.....	4
2.1	Erforderliche und zusätzliche Softwareprodukte.....	4
2.2	Setzen der Umgebungsvariablen.....	5
2.2.1	Für Windows.....	5
2.2.2	Für UNIX	5
2.3	Hinweise zur Installation zusätzlicher Komponenten.....	6
2.3.1	Installation von MySQL.....	6
2.3.2	Installation von Tomcat.....	7
2.3.3	Installation des Apache Webservers.....	7
3	Download und Installation des MyCoRe-Kerns.....	8
3.1	Download des MyCoRe-Kerns als Source-Paket.....	8
3.2	Download des MyCoRe-Kerns via Subversion.....	9
3.3	Konfiguration und Übersetzung des Kerns.....	9
4	Die MyCoRe-Beispielanwendung DocPortal.....	11
4.1	Grundlegender Aufbau und Ziel der Beispielanwendung.....	11
4.1.1	Allgemeines.....	11
4.1.2	Weiterführende Erläuterungen.....	11
4.2	Download der Beispielanwendung als Source-Paket.....	12
4.3	Download der Beispielanwendung via Subversion.....	12
4.4	Konfiguration und Installation von DocPortal.....	12
4.4.1	Grundlegendes zur Konfiguration.....	12
4.4.2	Initialisieren von DocPortal.....	13
4.4.3	Starten der MyCoRe-Kommandozeile.....	15
4.4.4	Erzeugen der Web-Anwendung.....	15
4.5	Laden der Beispieldaten.....	16
4.6	Sonstiges.....	16
5	Die Arbeit mit der Beispielanwendung.....	17
5.1	Benutzer.....	17
5.1.1	Benutzer der Beispielanwendung.....	17
5.1.2	Benutzer des WCMS.....	18
5.2	Zugriffsrechte auf Daten.....	18
5.3	Die Verwendung der Kommandozeilenschnittstelle	20
5.3.1	Basiskommandos des Kommandozeilensystems.....	20
5.3.2	Kommandos zur Arbeit mit der Benutzerverwaltung	21

5.3.3	Kommandos zur Administration des ACL-Systems.....	24
5.3.4	Kommandos zum Verwalten von Klassifikationen.....	24
5.3.5	Kommandos zur Verwaltung der Objekte.....	25
5.3.6	Kommandos zur Arbeit mit dem ImageViewer.....	26
5.3.7	Anfragen an das System per Kommandozeile.....	27
5.3.8	Sonstige Kommandos.....	27
5.4	Das SimpleWorkflow-System zur interaktiven Autorenenarbeit.....	27
5.4.1	Das MCRStartEditorServlet.....	29
5.4.2	Abläufe für neue Datenobjekte.....	33
5.4.3	Abläufe für Datenobjekte aus dem Server.....	33
5.4.4	Einbindung in das Access Control List - System.....	33
5.5	Der Klassifikationseditor.....	33
5.5.1	Start des Klassifikationseditors.....	34
5.5.2	Operationen des Klassifikationseditors.....	34
5.5.3	Klassifikationen.....	35
5.5.4	Kategorien.....	35
6	Möglichkeiten zur Nutzung optionaler Komponenten.....	38
6.1	Die Zusammenarbeit mit anderen DocPortal-Installationen.....	38
6.2	Die Integration des Bildbetrachter-Modules.....	40
6.3	Nutzung der OAI Schnittstelle.....	40
6.3.1	Grundlagen.....	40
6.3.2	Der OAI Data Provider.....	41
6.3.3	Installation.....	41
6.3.4	Der Deployment Descriptor.....	41
6.3.5	Die Einstellungen in mycore.properties.private.....	42
6.3.6	Instanzunabhängige Properties.....	42
6.3.7	Instanzabhängige Properties.....	43
6.3.8	Test.....	44
6.3.9	Zertifizierung und Registrierung.....	44
6.3.10	Formatierung der OAI-Ausgabe.....	44
6.4	Erzeugen einer Google Sitemap.....	44
6.5	Einbindung virtueller Host-Namen mit Hilfe des Apache-Web-Servers.....	45
6.5.1	Einbindung des Proxy-Modules.....	45
6.5.2	Die Verbindung von einer Servlet-Engine und Apache2.....	45
6.6	Sicherung der Daten.....	46
6.6.1	Backup.....	46
6.6.2	Recovery.....	47
6.7	Erzeugen einer Distribution.....	47
6.7.1	Installation von IzPack.....	47
6.7.2	Erstellen der Anwendungs-Distribution.....	47
6.7.3	Erstellen einer Beispieldaten-Distribution	48
7	Weiterführende Informationen zum Aufbau von MyCoRe-Anwendungen.....	49
7.1	XML-Syntax des Datenmodells.....	49
7.1.1	Die MCRObjektID.....	49
7.1.2	Das Klassifikationen-Datenmodell.....	50
7.1.3	Das Metadatenmodell.....	51
7.1.4	Das Speichermodell für die Multimediadaten (IFS).....	67
8	Tipps und Problembehebung.....	69

8.1	XSLT in ANT geht nicht.....	69
8.2	Java Console unter Linux Firefox.....	69
9	Anhang.....	70
9.1	Abbildungsverzeichnis.....	70
9.2	Tabellenverzeichnis.....	71

1 Allgemeines zu MyCoRe

1.1 Vorbemerkungen

MyCoRe bietet die Möglichkeit verschiedene Softwarekomponenten für die Speicherung und Verarbeitung der Daten (digitale Objekte und zugehörige Beschreibungsdaten) zu verwenden. Dabei spielt natürlich das verwendete Betriebssystem eine wesentliche Rolle. Jedes System hat Vor- und Nachteile, die an dieser Stelle kurz aufgezeigt werden sollen. Wir wollen es abschließend dem Anwender überlassen, in welchem System er für seine Anwendung die größten Vorteile sieht.

1.1.1 Betriebssysteme

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

1.1.2 Relationale Datenbanken

Im Bereich der SQL Datenbasis werden ab MyCoRe 1.3 alle Zugriffe über die Datenbank-Mapping-Sprache Hibernate¹ realisiert. Dieses garantiert eine einheitliche Programmierschnittstelle gegenüber der Anwendung und gestattet die Nutzung aller durch Hibernate unterstützten relationalen Datenbanksysteme wie HSQLDB², MySQL³, PostgreSQL⁴, IBM DB2⁵ und Oracle⁶. Über die Konfiguration von MyCoRe gibt der Administrator der Anwendung an, welche Datenbank konkret zum Einsatz kommt. Standardmäßig wird für DocPortal die HSQLDB verwendet. Dies ist ein sogenanntes leichtgewichtiges, Java-basiertes Datenbanksystem und erfordert keine zusätzlichen Installationen. Der erforderliche Code ist im MyCoRe-Paket bereits enthalten. Für produktive Anwendungsinstallationen sollte jedoch ein den Erfordernissen entsprechendes Datenbanksystem gewählt werden.

1.1.3 Suchindex

Für die Suche in den abgelegten Daten wurden verschiedene Ansätze realisiert. Mit Hilfe einer simplen JDOM-Suchmaschine über XPath-Ausdrücke kann auf einfache Art in den beschreibenden Daten der digitalen Objekte (Metadaten) gesucht werden. Dieser Ansatz ist für Entwicklungszwecke konzipiert. Standard ist der Einsatz einer Volltextsuchmaschine für Metadaten und Texte auf Basis des Apache-Projektes

¹siehe <http://www.hibernate.org/>

²siehe <http://hsqldb.org/>

³siehe <http://dev.mysql.com/>

⁴siehe <http://www.postgresql.org/>

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

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

Lucene⁷. Lucene ist ebenfalls in MyCoRe integriert und bedarf keiner zusätzlichen Installation. Zusätzlich kann die Volltextsuche des verwendeten relationalen Datenbanksystems eingesetzt werden. Bisher sind hinsichtlich der Leistungsfähigkeit von Lucene in einer Produktionsumgebung keine Einschränkungen bekannt.

1.1.4 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, entscheidet der jeweilige Administrator selbst. Über eigene Plugins lassen sich weitere Komponenten problemlos einbinden. Die jeweilige Software (OpenOffice, ps2ascii usw.) 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, odt und sxw.

1.1.5 Streaming Server

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.

1.1.6 Webanwendungs-Server

MyCoRe benötigt für die interaktive Arbeit eine Servlet-Engine. Bekanntester Vertreter hier ist Tomcat¹⁰. Alternativ bietet die MyCoRe-Beispielanwendung DocPortal die Nutzung von Jetty¹¹ an. Besonders für kleinere Anwendungen stellt es eine interessante Alternative dar. Jetty ist eine javabasierte und leichtgewichtige Anwendung, die ebenfalls in der DocPortal-Distribution enthalten ist. Eine weitere Alternative ist der IBM Application Server¹². Für Produktionsumgebungen mit einer Apache-Proxy-Anbindung empfiehlt sich jedoch der Einsatz von Tomcat oder WebSphere.

1.1.7 Web-Server

Zur Arbeit mit MyCoRe in Produktionsumgebungen empfehlen wir die Nutzung des Apache-Web-Servers¹³. Damit ist es möglich der Anwendung eine URL auf Port 80 ohne weitere Portangaben zuzuweisen (virtuelle Hosts). Bei der Verwendung des IBM WebSphere Application Servers ist ein IBM Webserver (basierend auf Apache) bereits integriert.

⁷siehe <http://lucene.apache.org/>

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

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

¹⁰siehe <http://tomcat.apache.org/>

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

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

¹³siehe <http://httpd.apache.org/>

1.2 Die Beispielanwendung

Zur Demonstration des Leistungsumfangs von MyCoRe wird eine Beispielanwendung mitgeliefert. Diese soll anschaulich die Funktionalitäten von MyCoRe aufzeigen und gleichzeitig eine praxisnahe Anwendung darstellen. Als Anwendungsszenario haben die Entwicklern einen Dokumenten-Server mit Namen *DocPortal* gewählt. Die Installation und Nutzung von MyCoRe und der darauf aufbauenden Anwendung 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.

1.3 Installationswege

Für die Installation der Beispielanwendung DocPortal gibt es drei Wege. Um einen ersten Eindruck vom System zu erhalten, wird von der Entwicklergruppe eine Binär-Distribution auf den Webseiten¹⁴ bereitgestellt. Dabei handelt es sich um eine mit der Software IZPack generierte jar-Datei. Durch die Ausführung der Datei wird die Installation von DocPortal gestartet. Als Ergebnis erhält man eine leere Anwendung, deren Daten in der HSQLDB verwaltet werden. Als Webanwendungs-Server wird Jetty verwendet. Eine Installationsanleitung ist in dem Dokument **Quick Installation Guide** zu finden. Weiterhin sind Beispieldaten in Distributionen thematisch zusammengestellt und können analog zu DocPortal zu installiert werden.

Die folgende Dokumentation beschäftigt sich damit, den zweiten (Download eines fertigen Source-Zweiges) und den dritten (Download vom Subversion-System) Installationsweg zu beschreiben. Es wird erklärt, wie die Beispielanwendung DocPortal schrittweise zu installieren ist und wie der Administrator per Konfiguration diese Anwendung in seine Umgebung einpasst. Hierbei ist zu unterscheiden, ob das Zielsystem unter den Betriebssystemen UNIX (Linux) oder MS Windows arbeitet. Auf die jeweiligen Unterschiede wird in den einzelnen Installationsschritten ggf. näher eingegangen. DocPortal kann auch unter Mac OS installiert werden, ein Referenz- und Testsystem in der Community steht momentan jedoch nicht zur Verfügung.

Die Release-Stände sowie aktuelle Snapshots können von der Home Page des Projektes unter <http://www.mycore.de> als *.zip bzw *.tar Datei heruntergeladen werden. Eine weitere Möglichkeit besteht in der Nutzung der Subversion Quellen. Hier erfolgt die Auswahl der Release-Stände mittels Tags. Für den aktuellen Entwicklerstand ist der Zweig 'trunk' auszuwählen. Verwiesen sei in diesem Zusammenhang auch auf die Projektseite bei Sorceforge¹⁵.

¹⁴Siehe http://www.mycore.de/content/main/download/docportal_binary.xml

¹⁵Siehe <http://sourceforge.net/projects/mycore>

2 Allgemeine Umgebungs- und Softwarevoraussetzungen

In der nachfolgenden Beschreibung wird davon ausgegangen, dass alle Arbeiten unter UNIX/Mac OS durch einen anzulegenden Benutzer `mcradmin` ausgeführt werden. Für die Benutzung von MyCoRe/DocPortal muss der Benutzer die nachfolgend gelistete Software verwenden können. Die in MyCoRe enthaltenen UNIX-Skripte erwarten die `bash`-Shell unter `/bin/bash`. Diese muss entsprechend zur Verfügung stehen oder die Skripte müssen von Hand angepasst werden. Für den Benutzer definiert der Administrator weiterhin einige Umgebungsvariablen. Hinzu kommen eine Reihe von optionalen Anwendungen, die dem MyCoRe-Benutzer bei Bedarf ebenfalls zu Verfügung stehen sollten.

Unter MS Windows ist kein spezieller Benutzer vorgesehen. Trotzdem sind eine Reihe von Softwareprodukten erforderlich. Umgebungsvariablen sind ebenfalls in diesem Zusammenhang zu definieren. Für die Nutzung alternativer Komponenten gibt es in diesem Kapitel (wie zu allen anderen Punkten) Hinweise.

2.1 Erforderliche und zusätzliche Softwareprodukte

Produkt	UNIX/MacOS	MS Windows
Java 2 Plattform, SE, 1.6.x (J2SE) ¹⁶ oder höher	erforderlich	erforderlich
ANT 1.7.x ¹⁷ oder höher inklusive 'regular expression'- und 'trax'-Erweiterung	erforderlich	erforderlich
OpenOffice 2.0.x ¹⁸ oder höher	optional, wird zur Volltextindizierung benötigt	optional, wird zur Volltextindizierung benötigt, weiterhin kann auch die MS Office-Suite benutzt werden
XPDF ¹⁹	optional, wird zur Volltextindizierung benötigt	optional, wird zur Volltextindizierung benötigt
GhostScript ²⁰	optional, wird zur Volltextindizierung benötigt	optional, wird zur Volltextindizierung benötigt
Relationales Datenbanksystem (MySQL, PostgreSQL, DB2, Oracle)	optional, kann die mitgelieferte HSQLDB ersetzen	optional, kann die mitgelieferte HSQLDB ersetzen

¹⁶<http://java.sun.com/>

¹⁷<http://ant.apache.org/>

¹⁸<http://www.openoffice.org/>

¹⁹<http://www.foolabs.com/xpdf/download.html>

²⁰<http://www.cs.wisc.edu/~ghost/>

Produkt	UNIX/Mac OS	MS Windows
Tomcat 5.5.x	optional, kann das mitgelieferte Jetty ersetzen	optional, kann das mitgelieferte Jetty ersetzen
Apache 2.2.x	optional, wird zur Umsetzung virtueller Host-Namen verwendet	optional, wird zur Umsetzung virtueller Host-Namen verwendet
JAI ²¹ , 1.1.2 oder höher	optional, wenn Sie das MyCoRe-Imaging-Modul einsetzen wollen	optional, wenn Sie das MyCoRe-Imaging-Modul einsetzen wollen

Tabelle 2.1: Softwarevoraussetzungen

Unter Linux sind die meisten Programme in den gängigen Distributionen enthalten. Unter SuSE 10.3 fehlt nur JAI²². Für das Betriebssystem AIX stellt die Firma IBM verschiedene Software-Downloads unter <http://ftp.software.ibm.com/> bereit. Unter MS Windows können die Installationspakete von den angegebenen Webseiten geholt werden.

2.2 Setzen der Umgebungsvariablen

2.2.1 Für Windows

Folgende Umgebungsvariablen müssen für die Arbeit mit MyCoRe definiert sein:

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

Tabelle 2.2: Umgebungsvariablen unter Windows

2.2.2 Für UNIX

Folgende Umgebungsvariablen müssen für die Arbeit mit MyCoRe definiert sein:

Variable	Verweis auf ...
MYCORE_HOME	... das Basisverzeichnis des MyCoRe-Quellzweiges (z.B. ~/mycore)
DOCPORTAL_HOME	... das Basisverzeichnis des DocPortal-Quellzweiges (z.B. ~/docportal)

Tabelle 2.3: Umgebungsvariablen unter Unix

²¹<http://www.sun.com>

²²siehe <http://cmswiki.rrz.uni-hamburg.de/hummel/MyCoRe/Dokumentation/HowTo/SuSe10-3>

Für den Benutzer `mcradmin` setzen Sie diese Variablen, indem Sie folgende Zeilen in der `.bashrc` ergänzen (bei Verwendung einer anderen Shell als `bash` ggf. anpassen):

```
export MYCORE_HOME=/home/mcradmin/mycore
export DOCPORTAL_HOME=/home/mcradmin/docportal
```

2.3 Hinweise zur Installation zusätzlicher Komponenten

2.3.1 Installation von MySQL

MySQL kann als Alternative zu dem standardmäßig mitgelieferten Datenbanksystem HSQLDB verwendet werden. Diese Kapitel beschreibt die Installation und Grundkonfiguration von MySQL.

MySQL kann von <http://www.mysql.de> oder aus einer Linux-Distribution bezogen werden.

1. `mysql-5.0.x` oder höher inklusive JDBC-Connector laut mitgelieferter Dokumentation installieren
2. MySQL starten
3. Passwort des root-Nutzers vom MySQL setzen

```
mysql -u root mysql
set password for root@'%'=password('...');
flush privileges;
```

4. 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 -p<rootpassword> mysql
GRANT ALL PRIVILEGES ON *.* TO mcradmin@localhost WITH
GRANT OPTION;
quit
```

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

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

6. 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
```

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

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

8. Falls weitere Benutzer 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
```

2.3.2 Installation von Tomcat

Tomcat kann als Alternative zum in DocPortal mitgelieferten Jetty eingesetzt werden. Tomcat ist ebenfalls ein Webanwendungs-Server, der besonders bei größeren Installationen Verwendung findet.

Bezugsquelle ist entweder direkt das Apache-Projekt <http://tomcat.apache.org/> oder die entsprechende Linux-Distribution. Hinweise zur Installation entnehmen Sie der jeweiligen Dokumentation. Hingewiesen werden soll lediglich auf den Umstand, dass für einen korrekten Lauf von MyCoRe die `CATALINA_OPTS`-Variable um die Optionen `'-server -Xms256m -Xmx512m -Xincgc'` mindestens erweitert werden sollte.

MyCoRe arbeitet beim Encoding der Zeichen standardmäßig mit UTF-8. Für eine korrekte Darstellung passen Sie die Datei `%CATALINA_HOME%/conf/server.xml` an, indem Sie das Tag für den Connector um das Attribut `URIEncoding='UTF-8'` ergänzen.

2.3.3 Installation des Apache Webservers

Die Installation des Apache Web Servers ist für das MyCoRe/DocPortal-Beispiel nicht erforderlich. Sollten Sie aus diesem Beispiel jedoch eine eigene Produktionsinstallation ableiten wollen, so benötigen Sie den Server zur Gestaltung virtueller Hosts. Für die Interaktion zwischen Servlet-Engine (Tomcat empfohlen) und Apache wird das Proxy-Modul verwendet. Die genaue Konfiguration entnehmen Sie der entsprechenden Literatur.

3 Download und Installation des MyCoRe-Kerns

3.1 Download des MyCoRe-Kerns als Source-Paket

Für eine Step-by-Step Installation der DocPortal-Anwendung benötigen Sie zuerst das MyCoRe-Basissystem (den MyCoRe Kern). DocPortal wird sich im weiteren Verlauf der Installation darauf beziehen und die erforderlichen Komponenten daraus verwenden. Der Kern muss in dem Verzeichnis zu finden sein, das unter der Umgebungsvariablen `MYCORE_HOME` angegebene wurde bzw. muss der Wert von `MYCORE_HOME` dem Installationsverzeichnis des Kerns entsprechen.

Auf der Web-Seite²³ des MyCoRe-Kerns wird ein Download des Source-Paketes angeboten. Da sich die folgende Anleitung auf das Release 2.0.x bezieht, sollten Sie dieses oder einen weiterentwickelten Snapshot auswählen. Packen Sie das erhaltene *.zip bzw. *.tar Verzeichnis auf einem Plattenbereich aus und setzen Sie die `MYCORE_HOME` Umgebungsvariable wie oben angegeben.

Nach dem erfolgreichem Auspacken des Source-Codes bzw. erfolgreichem Checkout (siehe unten) erhalten Sie unter dem Verzeichnis `~/mycore` folgende Dateistruktur.

Relativer Pfad	Inhalt
biuld.xml	Das Steuer-Script für ANT.
changelog.txt	Enthält Hinweise zu Produktänderungen.
config	Einige wenige Konfigurationsdateien für den Kern
documentation	Enthält alle MyCoRe-Dokumentationen.
install.xml	Das ANT-Steuer-Script für die Komplettinstallation.
lib	Java-Archive von Komponenten, welche in MyCoRe genutzt werden.
license.txt	Die verbindliche Lizenz von MyCoRe.
migration	Tools zur Migration von vorigen Releases.
modules	Zusätzliche modulare Programmkomponenten.
patches	Hinweise zu Updates
schema	XML Schema Dateien des MyCoRe-Kerns.
sources	Verzeichnis des Java-Quellcodes.
stylesheets	XSLT Stylesheets des MyCoRe-Kerns.
tests	Spezielle Klassen für JUnit-Test

Tabelle 3.1: Inhalt des MyCoRe-Kerns

²³<http://www.mycore.de/content/main/download.xml>

Das Einspielen von Updates erfolgt dann durch überschreiben der Quellen mit dem neuen Release / Snapshot.

3.2 Download des MyCoRe-Kerns via Subversion

Alternativ können Sie auch direkt aus dem Subversion-Repository die Programmdateien auschecken. Lesender Zugriff erfolgt über einen Subversion Client und den Apache+WebDAV Zugang. Es werden drei Zweige angeboten:

- tags – hier finden Sie Snapshots
- trunk – dies ist der aktuelle Entwicklungszweig (HEAD)
- braches – sowie Releases inklusiver ergänzender Fixes

```
svn checkout http://server.mycore.de/svn/mycore/trunk
```

Bugfixes und Korrekturen können bei Bedarf über ein Update direkt eingespielt werden.

3.3 Konfiguration und Übersetzung des Kerns



1. MyCoRe verwendet Apache Ant, 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 Skript ignoriert den lokal gesetzten `CLASSPATH` und generiert stattdessen einen eigenen `CLASSPATH` entsprechend Ihrer Konfiguration. Somit kann sichergestellt werden, dass nur die erforderlichen Pakete und Klassen in der richtigen Version verwendet werden. Die Konfiguration der Systemumgebung verwendeter Systeme für die XML-Speicherung (Lucene, SQL oder JDOM) 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. Sollten Sie das Imaging-Modul mit nutzen wollen, so muss JAI mit installiert sein und Sie müssen die für das Property `MCR.Modules.MyCoRe` die entsprechende Konfigurationszeile in der Datei `build.properties` auswählen.
4. Für SuSE-Linux müssen Sie noch die beiden Properties `MCR.System.SharedJarsDir` und `MCR.System.Jars` in der Datei `build.properties` aktivieren, da alle systemweiten `*.jar`-Dateien in einem gemeinsamen Verzeichnis `/usr/share/java` ablegt werden.

5. Sie sollten zunächst prüfen, ob ihre Systemumgebung korrekt eingerichtet ist, indem Sie

```
ant info
```

ausführen. Ant 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

4.1 Grundlegender Aufbau und Ziel der Beispielanwendung

4.1.1 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.

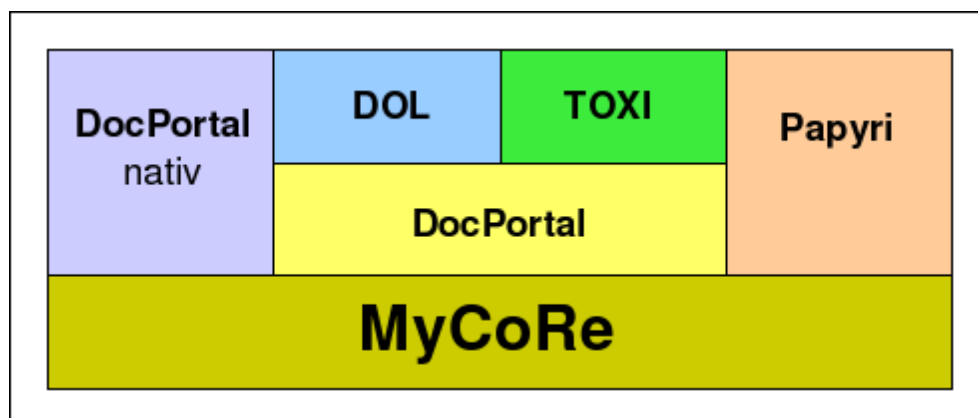


Abbildung 4.1: MyCoRe-Schichtenmodell

4.1.2 Weiterführende Erläuterungen

Die weiterführenden Erläuterungen zum Datenmodell und der Struktur der Beispielanwendung finden Sie in einem gesonderten Kapitel.

Auf der offiziellen Web-Seite des MyCoRe-Projektes²⁴ finden sie die in den nachfolgenden Abschnitten zu installierende Anwendung bereits lauffähig vor. Nach Abschluss der Arbeiten sollte auch Ihr System so funktionieren.

4.2 Download der Beispielanwendung als Source-Paket

Nachdem Sie den MyCoRe-Kern erfolgreich installiert haben, ist nun die Installation der mitgelieferten Beispielanwendung sinnvoll. Hier können Sie ein erstes Gefühl dafür gewinnen, wie eine eigene Anwendung gestaltet sein könnte.

Die entsprechenden *.zip bzw. *.tar Dateien finden Sie analog zum MyCoRe-Kern auf der Web-Seite des Projektes. Packen Sie diese einfach aus und weisen Sie die Umgebungsvariable `DOCPORTAL_HOME` entsprechend zu. Für Updates dieser Komponenten verfahren Sie wie in Abschnitt 3.1 beschrieben.

4.3 Download der Beispielanwendung via Subversion

Der Download von DocPortal mittels Subversion erfolgt analog zu dem des MyCoRe-Kerns.

```
svn checkout http://server.mycore.de/docportal/trunk
```

4.4 Konfiguration und Installation von DocPortal

4.4.1 Grundlegendes zur Konfiguration

Die Konfiguration der Beispielanwendung ist im Verzeichnis `$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 für die Nutzung der HSQLDB funktionieren. Kopieren Sie **als erstes** die Datei `mycore.properties.private.template` nach `mycore.properties.private`. Diese Datei 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.

Hinweis:

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:

²⁴<http://mycoresamplelinux.dl.uni-leipzig.de/>

- `MCR.Modules.Application` – Diesen Wert sollten Sie ändern, wenn das Iview-Module mit eingesetzt werden soll. Hierfür muss auch das Imaging-Module im Kern aktiviert worden sein.
- `MCR.basedir` – Pfad zur DocPortal-Root bzw. entspricht `DOCPORTAL_HOME`
- `MCR.Mail.Address` – Mail-Adresse, an die alle Nachrichten gehen sollen
- `MCR.Mail.Server` – SMTP-Server für ausgehende Mail
- `MCR.WebService.Admin` – Kennung des Axis-Admins zum deploy des Webservice
- `MCR.WebService.AdminPasswd` – und das dafür erforderliche Passwort

mycore.properties.wcms

DocPortal enthält ein WCMS-Modul (Web Content Management System) für die webbasierte Pflege der statischen Webseiten.

Die Konfiguration des WCSM wird in folgender Datei vorgenommen:

```
docportal/modules/module-wcms/aif/config/mycore.properties.wcms *
```

- Die Datei wird ggf. automatisch durch „ant webapps“ angelegt.

hibernate.cfg.xml

DocPortal verwendet Hibernate als Schnittstelle zwischen der Java-Anwendung und der verwendeten relationalen Datenbank. Kopieren Sie zunächst die Datei

```
docportal/config/hibernate/hibernate.cfg.xml.template
```

nach

```
docportal/config/hibernate/hibernate.cfg.xml.
```

Unter www.hibernate.org finden Sie ggf. weitere Hinweise, wenn Sie zu einem späteren Zeitpunkt die Hibernate-Anbindung für Ihre Datenbank optimieren wollen, oder besonderen Funktionen wie Statistiken, Caches und Logging nutzen wollen.

document.xsl

Um den Image-Viewer nun in der DocPortal-Anwendung zu sehen, muss dieser noch in der Dokumenten-Darstellung integriert werden. Dazu sind in der Datei `$DOCPORTAL_HOME/modules/iview/xsl/document.xsl` nur die Kommentarzeichen zu entfernen.

4.4.2 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. Es werden standardmäßig drei Verzeichnisse angelegt, welche generierte Daten enthalten.

Verzeichnis	Beschreibung
build	In diesem Verzeichnis werden alle für die Kommandozeilenarbeit und die Webanwendung erforderlichen Daten abgelegt.
data	In diesem Verzeichnis sind standardmäßig alle Daten, Inizes und Objektdaten abgelegt.
save	In diesem Verzeichnis befinden sich alle Sicherungen. Es wird durch Anwendung des Kommandos Save... bzw. ant saveWebContent gefüllt.

Tabelle 4.1: Verzeichnisse des Build-Prozesses

Führen Sie die nachfolgenden Kommandos nun schrittweise aus.

1. Prüfen Sie die Systemumgebung in DocPortal mit
`ant info`
2. Erzeugen aller erforderlichen Anwendungsverzeichnisse
`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 und kopieren Sie die notwendigen Java Bibliotheken mit dem Kommando
`ant jar`
5. Erzeugen Sie die CommandLineTools mit Hilfe des Kommandos
`ant create.scripts`
6. Starten der HSQLDB (wenn kein anderes Datenbanksystem verwendet wird)
`build/bin/hsqldbstart.sh` bzw. `build/bin/hsqldbstart.cmd`
7. Laden Sie die Standard-ACL's, Gruppen und Benutzer für die Beispielanwendung mit den Kommandos
`ant create.users` und `ant create.default-rules`
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.

4.4.3 Starten der MyCoRe-Kommandozeile

Starten Sie die MyCoRe-Kommandozeile, auch `CommandLineInterface` genannt, durch Aufruf von `build/bin/mycore.sh` (UNIX/MacOS) bzw. `build\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 7.1.2 auf Seite 50.

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.

4.4.4 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-Skript 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 Skript, welches die Servlet-Engine Jetty aktiviert.

`build/bin/jettystart.sh` bzw. `build\bin\jettystart`

Starten Sie nun einen Web-Browser der URL <http://localhost:8291/>

Damit Ihre Anwendung auch über die WebServices erreichbar ist (entfernte Anfragen), müssen Sie diese Funktionalität noch gesondert aktivieren. Nachdem Sie mit den Properties `MCR.WebServices.Admin` und `MCR.WebServices.AdminPasswd` Kennung und Passwort für den Administrator des Webservice gesetzt haben, verwenden Sie hierzu bei laufender Servlet Engine das Kommando

1. `ant webservice.deploy`

Die nun verfügbaren WebService-Dienste sehen Sie bei Eingabe der URL <http://localhost:8291/servlets/AxisServlet> in Ihrem Web-Browser.

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

```
1. ant webservice.undeploy
```

GRATULATION,
SIE HABEN NUN ERFOLGREICH DocPortal INSTALLIERT!

4.5 Laden der Beispieldaten

MyCoRe stellt eine umfangreiche Sammlung von Beispieldaten bereit. Diese wurde aus Performance-Gründen in ein extra Paket gepackt. Sie können diese als *.jar-Dateien von MyCoRe-Webserver 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
audiosample	Einige größere Audiodateien
defaultsample	Ein kleineres 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

Tabelle 4.2: Beispieldaten im CVS

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

```
java -jar -Xmx1024m -Xms1024m ...-installable.jar
```

4.6 Sonstiges

Mit Hilfe des Build-Prozesses können Sie auch gezielt die erzeugte Anwendung und/oder die Daten entfernen. Während das Kommando `ant clean` bzw. `ant clean.system` alle generierten Anwendungsdaten (standardmäßig im `build-`Verzeichnis) entfernt, löscht `ant clean.data` das Verzeichnis der geladenen Daten. Bitte gehen Sie mit diesen Kommandos sorgfältig um.

Das Kommando `ant docs` sammelt alle Dokumentationen und JavaDocs von MyCoRe und DocPortal ein und speichert sie im Verzeichnis `build/documentation` ab.

5 Die Arbeit mit der Beispielanwendung

5.1 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.

5.1.1 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 als auch für eine ganze Gruppe erfolgen. Im Kapitel 5.2 wird dieses Zugriffssystem (ACL-System) näher erläutert.

Gruppe	Beschreibung
root	Die Gruppe der Superuser
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 Gruppe der Gäste

Tabelle 5.1: Beispielgruppen in DocPortal

Benutzer	Gruppe	Passwort
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

Tabelle 5.2: Beispielbenutzer in DocPortal

5.1.2 Benutzer des WCMS

Aktuell ist nur der Benutzer admin mit dem Passwort wcms eingerichtet.

Wenn sie weitere Nutzer anlegen wollen, führen sie bitte folgende Schritte aus:

- mycore.jar in den CLASSPATH aufnehmen
- Aufruf von

```
java wcms.util.HashCipher neuesKennwort
```
- den dann ausgegebenen Hashwert in Datei

```
docportal/modules/module-wcms/aif/config/WCMSUserDB.xml
```

eintragen

5.2 Zugriffsrechte auf Daten

Alle in MyCoRe gespeicherten Objekte besitzen einen Zugriffsschutz über Access Control Lists (ACL). Dieser Schutzmechanismus ordnet jedem Objekt eine Reihe von Attributen zu, welche als Bedingung erfüllt sein müssen, damit der Zugriff auf das Objekt gewährt wird. Der Vergleich der in den ACL's gespeicherten Voraussetzungen gegen den aktuellen Zustand des Zugriffs erfolgt mittels der Sitzung (Session). In ihr werden alle Umgebungsinformationen wie zugreifende IP, Benutzer, usw. gespeichert. Das für MyCoRe implementierte ACL-System kennt drei Attribute für die Auswertung.

- Benutzer
- Gruppen

- Netzadressen in Form von IP's bzw. IP-Blöcken

Bei der interaktiven Eingabe neuer Daten werden standardmäßig im Projekt festgelegte Zugriffsregeln vorgegeben. Diese trennen sich in die Bereiche Lesen und Bearbeiten. Im Bereich des Bearbeitens wird wiederum in die Arbeit im Workflow und im Server-System unterschieden. Hier können jeweils die Arbeitsgänge Editieren und Löschen getrennt voneinander festgelegt werden. Für DocPortal sind folgende Berechtigungen (Permissions) definiert:

- read – gestattet den lesenden Zugriff
- writewf – gestattet das Schreiben bzw. Ändern, solange das Objekt im Simple Workflow (SWF) ist
- deletewf – gestattet das Löschen, solange das Objekt im Simple Workflow (SWF) ist
- writedb – gestattet das Schreiben bzw. Ändern, wenn das Objekt im Server ist
- deletedb – gestattet das Löschen, wenn das Objekt im Server ist

Die Permissions werden in der Editor-Verarbeitung des Neuanlegens eines Datensatzes aus einem Standard-ACL-Datensatz erzeugt. Hierzu kommt die Berechtigung des aktuellen Autors zu allen Bearbeitungsfunktionen. Die Leserechte sind für alle Benutzer zugelassen. Diese Einstellung gilt konkret für die Docportal-Anwendung. Für eigene Projekte kann dies ganz anders definiert werden. Die Permissions werden im Workflow-Prozess direkt im Datensatz gespeichert. Die Notation der Zugriffsregeln im Datenabschnitt **services** ist als Datentyp **MCRMetaAccessRule** im Kapitel 7 näher beschrieben. Beim Einstellen (Hochladen) der Objekte in den Server werden die Regeln **writewf** und **deletewf** vollständig entfernt, da sie nicht mehr benötigt werden. Alle anderen Zugriffsregeln werden im MyCoRe-ACL-System in die Datenbank eingetragen und anschließend aus dem Originaldatensatz gelöscht. Damit ist das Objekt gesichert, ein Zugriff darauf benötigt nun immer die Zustimmung der ACL-Komponente. Eine Änderung der Permissions kann nun nur noch bei einer Schreibberechtigung erfolgen. Die interaktiven Komponenten von DocPortal bieten hier ein Aktionsfeld mit einem Editor-Formular an. **Nutzer der Gruppe admingroup sind immer für den Zugriff berechtigt, auch wenn dies nicht explizit angegeben ist.**

Um den einzelnen digitalen Objekten ggf. eine höhere Sicherheit zu geben, können ergänzende Permissions in den Derivate-XML-Daten angegeben werden. Bei einem Zugriff wird zuerst geprüft, ob Zugriffsrechte für den Metadatenatz bestehen, anschließend wird geprüft ob es zusätzliche Einschränkungen aus dem Derivate gibt. Standardmäßig ist im Derivate immer die Permission **'true'** gesetzt, was keinen weiteren Einschränkungen entspricht.

Das ACL-System gestattet prinzipiell eine beliebige Sicherung mit anwendungsspezifischen Kriterien. Im Programmer Guide finden Sie detaillierte Informationen zu Weiterentwicklung Ihrer eigenen Applikationen.

5.3 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, usw.) 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 den MyCoRe Programmer Guide.

5.3.1 Basiskommandos des Kommandozeilensystems

Das Kommandozeilensystem enthält eine kleine Zahl an Basiskommandos. Diese sollen im folgenden Abschnitt beschrieben werden. Sie werden ergänzt durch weitere 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:

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

5.3.2 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 einmalig bei der Installation und Konfiguration des MyCoRe-Systems 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 geändert 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 übernommen 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 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 Plausibilitätsprüfung 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.

```
export all users to file {0}
```

```
export all groups to file {0}
```

```
export user {0} to file {1}
```

export 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 entstandenen 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) `export 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:> export user <superuser> to file <superuser.xml>
MyCoRe:> export user <guest> to file <guest.xml>
MyCoRe:> export group <superuser-group> to file <superuser-group.xml>
MyCoRe:> export group <guest-group> to file <guest-group.xml>
MyCoRe:> export all users to file <all-users.xml>
MyCoRe:> export 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
```

5.3.3 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.

export 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} -

5.3.4 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, da 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 geladenen Klassifikationen. Achtung, lassen Sie keine Kategorien einer bestehenden Klassifikation weg, da sonst Ihr System ggf. in einen inkonsistenten Zustand kommen kann!

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

export classification {0} to {1} with {2} – es wird eine Klassifikation mit der im Parameter {0} angegebenen MCRObjectID in eine Datei mit dem im Parameter {1} angegebenen Namen gespeichert. Es wird zur Transformation der zu speichernden Daten das Stylesheet {2}-classification.xsl unter \$DOCPORTAL_HOME/stylesheets verwendet.

export all classifications to directory {0} with {1} – es werden alle im System befindlichen Klassifikationen in Dateien des Verzeichnisses mit dem im Parameter {0} angegebenen Namen gespeichert. Es wird zur Transformation der zu speichernden Daten das Stylesheet {1}-classification.xsl unter \$DOCPORTAL_HOME/stylesheets verwendet.

5.3.5 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 MCRObjectIDs anzugeben. Es werden zur Transformation der zu speichernden Daten die Stylesheets {3}-object.xsl und {3}-derivate.xsl unter \$DOCPORTAL_HOME/stylesheets verwendet. Standard ist save-object.xsl bzw. save-derivate.xsl.

`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 MCRObjectID'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 MCRObjectID {0} in das Verzeichnis {1}.

5.3.6 Kommandos zur Arbeit mit dem ImageViewer

`clear image cache` – das Kommando löscht den Image Cache.

`create image cache for file {0}` - das Kommando erzeugt den Image Cache für die Datei {0}.

`remove image cache for file {0}` - das Kommando löscht den Image Cache für die Datei {0}.

`create image cache for derivate {0}` - das Kommando erzeugt den Image Cache für das gesamte Derivate {0}.

`remove image cache for derivate {0}` - das Kommando löscht den Image Cache für das gesamte Derivate {0}.

Hinweis: Beachten Sie, dass diese Kommandos nur funktionieren, wenn Sie die notwendigen Schritte aus Kapitel 6.2.2 -> „Aktivieren des Cache“ durchgeführt haben.

5.3.7 Anfragen an das System per Kommandozeile

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

run local query {0} – das Kommando startet eine Anfrage gegen das lokale System mit einer Query, welche im String {0} steht.

run distributed query {0} – das Kommando startet eine Anfrage gegen alle bekannten Systeme mit einer Query, welche im String {0} steht.

5.3.8 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 ID for base {0}

get next 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 MCRObjectID für eine vorgegebene MCRObjectID-Basis zurück. Die Basis besteht aus Projekt- und Type-String in der Form `project_type`.²⁵

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!

5.4 Das SimpleWorkflow-System zur interaktiven Autorenenarbeit

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 ein 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:

²⁵Siehe Abschnitt zur MCRObjectID.

- 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 verändern.
- 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.

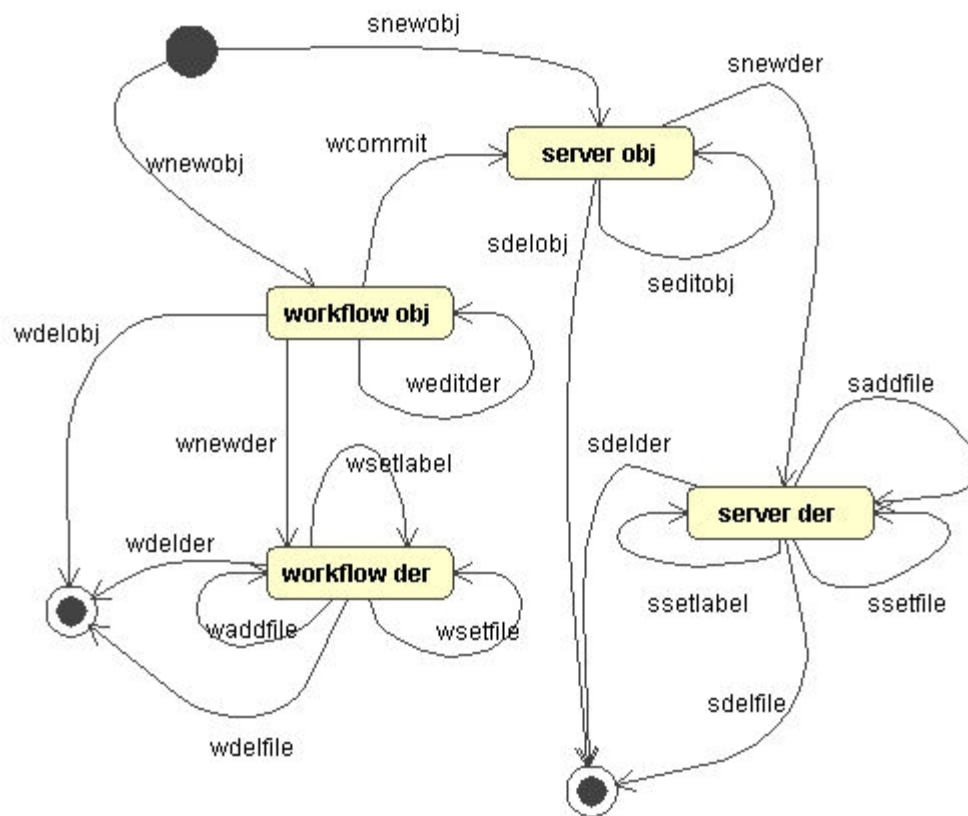


Abbildung 5.1: Funktionsschema des SimpleWorkflow

5.4.1 Das MCRStartEditorServlet

Dieses Servlet ist der Einstiegspunkt für die Nutzung des SimpleWorkflow-Systems. Von ihm aus werden alle Verarbeitungsprozesse angestoßen. 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 5.3: Parameter des MCRStartEditorServlets

Die nächsten Tabellen sollen eine Übersicht der möglichen Aktionen geben. Jede Aktion ist dabei an eine entsprechende Berechtigung gebunden, welche der aktuelle Benutzer gerade haben muss. Hat er sie nicht, so wird seine Aktion abgewiesen und nicht ausgeführt. 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 alle Aktionen mit s einen Zugriff und Änderungen im Server-System.

Aktion	todo	ID	Permission	ruft
Anlegen neuer Metadaten	wnewobj	tf_mcrId	create-type	editor_form_type_[layout_]author.xml
Anlegen eines Neuen Derivates	wnewder	se_mcrId	create-type	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_type_[layout_]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_type_editor.xml
Löschen eines Derivates	wdelder	se_mcrId	deletewf	editor_type_editor.xml
Löschen einer Datei aus einem Derivate	wdelfile	se_mcrId	writewf	editor_type_editor.xml
Setzen der Hauptdatei in einem Derivate	wsetfile	se_mcrId	writewf	editor_type_editor.xml
Hochladen eines Datensatzes vom Plattenbereich zum Server	wcommit	se_mcrId	writedb	MCRQueryServlet mit der ID mit mode=ObjectMetadata

Tabelle 5.4: Mögliche Aktionen mit dem MCRStartEditorServlet auf dem Plattenbereich

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

Aktion	todo	ID	Permission	ruft
einem Derivate				

Tabelle 5.5: Mögliche Aktionen mit dem MCRStartEditorServlet im Server

5.4.2 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.

5.4.3 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 Programmier Guide.

5.4.4 Einbindung in das Access Control List - 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.

5.5 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>	 

Abbildung 5.2: Auswahl der zu bearbeitenden Klassifikation

5.5.1 Start des Klassifikationseditors

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

Beispiel:

`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!

5.5.2 Operationen des Klassifikationseditors

Hinweis:

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

5.5.3 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.

5.5.4 Kategorien

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

Klassifikationen editieren



Abbildung 5.3: Einzelne Kategorien einer Klassifikation bearbeiten

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 angehängen, 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 angehängen.

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.

6.1 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/"
    class="org.mycore.services.fieldquery.MCRQueryClientWebService"
    access="webservice"
    servicepath="services/MCRWebService"
    staticpath="receive/">
    <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/"
    class="org.mycore.services.fieldquery.MCRQueryClientWebService"
    access="webservice"
    servicepath="services/MCRWebService"
    staticpath="receive/">
    <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. Die Attribute für einen Host bedeuten im Einzelnen:

- `alias` – der im MyCoRe-System zu verwendende Alias für einen externen Host
- `url` – die Basis-URL eines externen Hosts
- `class` – die MyCoRe-Java-Klasse, die der externen Zugriff realisiert²⁶
- `access` – einen Typ-String für die Zugriffsform
- `servicepath` – die URL-Erweiterung als Pfad auf dem externen Systembibliotheken
- `staticpath` – die URL-Erweiterung für die Darstellung der statischen URL des Objektes auf dem entfernten Rechner

²⁶Das Beispiel zeigt der standardmäßigen Zugriff über WebServices.

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

Tabelle 6.1: Feste Test-Instanzen für das MyCoRe-Beispiel

6.2 Die Integration des Bildbetrachter-Modules

DocPortal bietet einen komfortablen Bildbetrachter, um Dateien verschiedener Formate anzuzeigen. Dazu sind die beiden Module „Imaging“ aus dem MyCoRe-Kern und das Modul „IView“ aus DocPortal zu installieren. Dieser Abschnitt beschreibt die notwendigen Arbeiten, wenn Sie die Module nicht bei der Erstinstallation mit integriert haben.

Hinweis: Aus lizenzrechtlichen Gründen können die Module nicht direkt in die Distribution eingebunden werden, weil sie auf Sun's Java-Bibliothek JAI(JDK) beruhen. Der Download der Software muss vom jeweiligen Nutzer selbst durchgeführt werden. Eine direkte Integration der *.jar Files in MyCoRe ist leider nicht gestattet.

Die Einbindung des Bildbetrachters erfolgt in zwei Schritten:

1. Integration des „Module-Imaging“ im MyCoRe-Kern
2. Integration des „Module-IView“ in der DocPortal-Anwendung

Beide Phasen wurde bereits ausführlich in den Abschnitten des Kapitels 4 beschrieben.

Hinweis: Es wird empfohlen MCR-IView mit dem eingebauten Cache zu benutzen. So kann sichergestellt werden, dass hochaufgelöste Bilder ohne Verzögerungen betrachtet werden können.

In der Datei `$DocPortal_HOME/modules/iview/config/mycore.iview.properties` sind die entsprechenden Einstellungen zur Nutzung des Cache per Standard schon vorgenommen. Änderungen können bei Bedarf durchgeführt werden (Detaillierte Informationen zur Cache-Konfiguration können im ProgrammersGuide nachgelesen werden).

6.3 Nutzung der OAI Schnittstelle

6.3.1 Grundlagen

Die Open Archives Initiative²⁷ 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.

²⁷<http://www.openarchives.org/>

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 werden²⁸. Ein Repository wird von einem Data Provider betrieben, um Harvestern den Zugriff auf Metadaten zu ermöglichen.

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

6.3.2 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.

6.3.3 Installation

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

6.3.4 Der Deployment Descriptor

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

²⁸<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>

```

6.3.5 Die Einstellungen in mycore.properties.private

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.

6.3.6 Instanzunabhängige Properties

MCR.oai.adminemail=admin@uni-irgendwo.de (notwendig) Der Administrator der OAI-Repositories.

MCR.oai.resumptiontoken.dir=%MCR.basedir%/oai (notwendig) Ein Verzeichnis, in welches der OAI Data Provider Informationen über Resumption Token ablegt. Standardmäßig ist dies ein Unterverzeichnis von der Anwendungswurzel.

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.MCROAIQueryImpl (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 Metadaten-Schema in das für OAI benutzte OAI Dublin Core Metadaten-

Schema durchführt. Wenn sich das im Archiv benutzte Metadaten-Schema ändert, muss dieses Stylesheet angepasst werden. Optional können weitere Stylesheets angegeben werden, die einen Harvester mit anderen Metadaten-Formaten 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. Weitere Transformer sind bereits im Projekt vorhanden.

```
MCR.oai.resumptiontoken.store=org.mycore.backend.hibernate.MCRHIBResum
ptionTokenStore (notwendig) Der Eintrag beschreibt die Speicherklasse, die
zur Ablage der Resumption Token verwendet wird.
```

6.3.7 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.searchfields.OAIDataProvider=format,type, subject
(notwendig) Die Suchindex-Felder, die zur Bildung der Struktur des OAI-
Repositories verwendet wird.
```

```
MCR.oai.setscheme.classids.OAIDataProvider.format=DocPortal_class_0000
0006 (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.queryRestriction.OAIProvider=<condition field="objecttype"
value="document" operator="=" /> (optional) Die MyCoRe-Query-Ausdruck
in XML, der 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.
```

6.3.8 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.

6.3.9 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.

6.3.10 Formatierung der OAI-Ausgabe

Für die Ausgabe von OAI-Requests in Browsern wurde ein Stylesheet integriert. Damit kann der Browser die Rückgabe des OAI Providers in XHTML konvertieren. Es werden außerdem Links für weitere OAI-Anfragen generiert. Die Originaldatei steht auf <http://software.eprints.org/xslt.php> unter GPL zur Verfügung.

6.4 Erzeugen einer Google Sitemap

In MyCoRe gibt es im Indexing-Module ein kleines Servlet, welches eine Datei erzeugt, die den Konventionen des Sitemap-Protokolls²⁹ von Google entspricht.. Der Zugriff vom Internet aus erfolgt mit der URL http://myhost/sitemap_google.xml . Für die Konfiguration dieser XML-Ausgabe stehen in den Property-Dateien zwei Werte zur Verfügung.

- `MCR.googlesitemap.types` – Liste von mit Komma getrennten MyCoRe-Objektypen – Standard ist document
- `MCR.googlesitemap.freq` – Zugriffsfrequenz von Google – Standard ist monthly – weiter möglich sind: always / hourly / daily / weekly / monthly / yearly / never

Um den Zugriff von Google gezielt anzufordern, sollten folgende Schritte durchgeführt werden:

- Auf <http://www.google.de/sitemaps> gehen.
- Ein Google – Konto einrichten.
- Die URL der Anwendung eintragen.
- Die URL der zugehörigen *sitemap* eintragen.

Nun sollte Google die geladenen Datenobjekte gezielt indizieren und so eine gute Verfügbarkeit in den Suchmaschinen erreichen.

²⁹<https://www.google.com/webmasters/sitemaps/docs/de/protocol.html>

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

Standardmäßig ist der Apache2 in den Installations-CD's aller gängigen Linux-Distributionen und in MacOS enthalten. Für Windows muss ein gesonderter Download erfolgen.³⁰ Der Quellcode des Apache2 liegt auf <http://httpd.apache.org> für ein Download bereit. Die folgende Beschreibung bezieht sich auf die Apache-version 2.0.x. Die weitere Beschreibung bezieht sich hinsichtlich der Pfade auf ein UNIX/MacOS-System, für Windows sind die dazu korrespondierenden Pfade zu nutzen.

6.5.1 Einbindung des Proxy-Modules

Die Einbindung des Proxy-modules ist relativ einfach zu bewerkstelligen. In der Datei `/etc/sysconfig/apache2` sind in der Zeile der Variable **APACHE_MODULES** die Module **proxy,proxy_http,proxy_connect** hinzuzufügen. Nach der Änderung ist der Neustart des Apache-Servers erforderlich.

6.5.2 Die Verbindung von einer Servlet-Engine und Apache2

Die Verbindung zwischen dem Apache2 und der Servlet-Engine wird in den Konfigurationsfiles `/etc/apache2/httpd.conf` und `/etc/apache2/http-vhosts.conf` konfiguriert.

In der Datei `/etc/apache2/httpd.conf` ist die Include-Anweisung für das Lesen der Zusatzkonfiguration `http-vhosts.conf` zu aktivieren. Anschließend wird der eigentliche virtuelle Host in dieser Datei definiert. Dabei sind natürlich die Pfade zu den einzelnen Verzeichnissen entsprechend den aktuellen Gegebenheiten anzupassen.

³⁰Siehe Kapitel 2

```
<VirtualHost mycoresample.dl.uni-leipzig.de:80>
    ProxyPass / http://mycoresamplelinux.dl.uni-leipzig.de:8291/
    ProxyPassReverse / http://mycoresamplelinux.dl.uni-leipzig.de:8291/

    ServerAdmin mcradmin@mycoresamplelinux.dl.uni-leipzig.de
    DocumentRoot /home/mcradmin/docportal/webapps
    ServerName mycoresamplelinux.dl.uni-leipzig.de
    ErrorLog /var/log/apache2/mycoresample-error_log
    CustomLog /var/log/apache2/mycoresample-access_log common
    Alias /mycoresamplelinux "/home/mcradmin/docportal/webapps"

<Directory "/home/mcradmin/docportal/webapps/" >
    Options Indexes FollowSymLinks
    DirectoryIndex
    AddHandler jakarta-servlet2 .jsp
    Order deny,allow
    Deny from all
    Allow from all
</Directory>

<Directory "/mycoresamplelinux" >
    Options Indexes FollowSymLinks
    DirectoryIndex
    AddHandler jakarta-servlet2 .jsp
    Order deny,allow
    Deny from all
    Allow from all
</Directory>
</VirtualHost>
```

Nach dieser Änderung ist zuerst die Servlet-Engine zu starten. Anschließend kann der Apache-Server mit `/usr/sbin/rcapache2` neu gestartet werden.

6.6 Sicherung der Daten

6.6.1 Backup

Natürlich muss ein Dokumenten-Server im Produktionseinsatz auch ein Datensicherungskonzept haben. Je nach Einsatz der gewählten Datenbank ist der erste Schritt natürlich eine Sicherung der selben nach Backup-Vorgaben des Herstellers. Dazu kommt auch eine regelmäßige Backup-Strategie für das Server-System.

Ein weiterer Schritt ist das komplette Auslesen des Datenbestandes und die Speicherung auf einem externen (ggf. Netzwerk-) Filesystem. Mit dieser Methode

lassen sich auch Migrationen durchführen. Die Distribution enthält im Verzeichnis `bin` Scripts für Windows und Linux, die diese Aufgabe realisieren. Gestartet wird jeweils das `Save.cmd` bzw. `Save.sh` Script. Diese Kommandodateien rufen dann weitere Scripts, welche mit `Save...` beginnen auf. Im Ergebnis des Kommandos werden alle Daten in das `$DOCPORTAL_HOME/save` -Verzeichnis exportiert.

6.6.2 Recovery

Die mittels des Save-Kommandos exportierten Daten können im Bedarfsfall wieder in ein ggf. neu erstelltes System eingespielt werden. Sollte es nötig sein, einzelne Benutzer mit Ihren gesicherten Passwörtern neu zu laden, so ist vorher das im `bin`-Verzeichnis befindliche Script `SelectUserFromSave.sh` zu starten. Nach erfolgreichem Lauf befinden sich für jeden Benutzer / jede Gruppe Dateien im Verzeichnis `$DOCPORTAL_HOME/config/user`. Diese Daten können dann im MyCoRe-Kommandozeilen-Tool mit den Kommandos `'delete user ...'` und `'import user from file ...'` geladen werden.

Sollte nur der Metadaten-Index (z. B. Lucene) defekt sein, so hilft das Script `RepairFromXMLStore.sh` diesen Index neu zu erstellen.

Sind die XML-Daten bzw. die Derivate defekt bzw. weg so könne die gesicherten Daten aus dem `save`-Verzeichnis in die `Workflow-Directories` kopiert werden. Mit dem Kommandozeile-Tool bzw. speziellen Scripts unter `unixtools` können diese Objekte nun wieder in das System eingebracht werden.

6.7 Erzeugen einer Distribution

6.7.1 Installation von IzPack

IzPack³¹ ist eine Third-Party-Software, welche das generieren von Distributionen inklusive einer Vorort-Installationsroutine ermöglicht. Die Software ist für Open Source Projekte frei verfügbar. Die folgende Beschreibung bezieht sich auf die Arbeit unter Linux, sollte aber unter der anderen Betriebssystemen analog genauso funktionieren.

Die Software ist am günstigsten unter `/usr/local` als Verzeichnis IzPack zu installieren. Anschließend wurde noch die Umgebungsvariable `IZPACK_HOME` in die Datei `~/.bashrc` eingetragen. Nun kann die Software in den eigenen Build-Prozess eingebunden werden. Im Falle von MyCoRe ist das die Datei `DOCPORTAL_HOME/build.xml`³².

```
export IZPACK_HOME=/usr/local/IzPack
```

6.7.2 Erstellen der Anwendungs-Distribution

Um eine eigene Distribution zu erstellen, ist es sinnvoll, in einer Sandbox ein leeres Docportal-System aufzusetzen. Dazu gehen Sie entsprechend der Anleitungen in den

³¹siehe <http://www.izforge.com/izpack/>

³²siehe Target izpack

Kapiteln 2-4 vor. Wichtig ist, dass die Beispieldaten NICHT mit installiert werden, da sonst die Distribution sehr schnell extrem groß wird. Zum anderen will ggf. der Nutzer lieber ein leeres System um gleich eigene Dateien einstellen zu können. Auf jeden Fall sollten aber alle Beispielbenutzer und -klassifikationen mit geladen sein. Auch die Web-Anwendung sollte aufgebaut sein und funktionieren.

IzPack geht von einer Variable `INSTALL_PATH` aus, welche die Installationswurzel seitens der späteren Nutzer definiert. Es müssen daher erst alle `MYCORE_HOME`, `DOCPORTAL_HOME`-Variablen sowie die `%basedir%`-Variablen dagegen ausgetauscht werden. Dies geschieht mit Hilfe eines ANT-Targets, welches in der Datei *docportal/build.xml* enthalten ist. Das Target setzt nach dem Bau der Distribution auch alle ersetzten Werte wieder auf ihren Ursprung zurück. Das betrifft jedoch nur das System auf dem Sie konkret die Distribution erstellen. Soll eine Distribution für mehrere Systeme erstellt werden, müssen die Scripts unter `.../bin` auf einem gesondert aufzusetzenden Zielsystem generiert werden. Die offizielle Distribution enthält als Zielsystem **Windows und Linux**.

Wenn die Installation in der Sandbox komplett funktioniert, muss noch die Datei `mycore.properties.private` in `mycore.properties.private.izpack` kopiert werden. Darin ist dann für den Eintrag `MCR.basedir` die Variable `$INSTALL_PATH` zu setzen. Sind alle Vorbereitungen abgeschlossen, so kann die Distribution mit folgendem Kommando gebaut werden:

```
ant izpack
```

Installiert wird diese Distribution mit `java -jar docportal-installable.jar`. Eine genaue Anleitung ist im **Quick Installation Guide** zu finden.

6.7.3 Erstellen einer Beispieldaten-Distribution

Auch die Beispieldaten sind in selbstinstallierenden Distributionen erhältlich. Um selbst eigene Daten in dieser Form verteilen zu können, müssen Sie nur ein existierendes Beispiel adaptieren und die dort verwendeten Steuerdateien entsprechend anpassen. Auch hier erzeugt ein `ant izpack` die gewünschte Distribution. Die Scripts unter `bin` sorgen für das Laden der Daten zum Zeitpunkt der Installation.

7 Weiterführende Informationen zum Aufbau von MyCoRe-Anwendungen

7.1 XML-Syntax des Datenmodells

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

7.1.1 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"

projektkürzel	Dieses Element ist für ein Projekt und/oder eine Einrichtung / Datengruppe festzulegen, zum Beispiel UBLPapyri oder MyCoReDocument. In MyCoRe wird es teilweise auch zur Identifikation einzelner Konfigurationsdaten mit genutzt.
type	Das Element beschreibt den Datenmodelltyp, d. h. der type verweist auf die zugehörige Datenmodell-Konfiguration, zum Beispiel datamodel-author oder datamodel-document. In MyCoRe wird es oft zur Identifikation einzelner Konfigurationsdaten im Zusammenhang mit der Verarbeitung dieses Datenmodells genutzt.
nummer	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 7.1: 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.

7.1.2 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 sein, 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.*'.

Hinweis:

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>
```

7.1.3 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 Standardmetadatatypen 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>
```

Die oben gezeigte Syntax stellt den Rahmen eines jeden Metadaten-Objektes dar. Diese Struktur ist immer gleich und muss so eingehalten werden.

- 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 Vererbungsbäumen 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>
```

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. 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="...">
    ...
  </...>
  ...
</metadata>
```

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 7.2: 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.

XML-Syntax des Metadatentyps MCRMetaAddress:

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

Beispiel des Metadatentyps MCRMetaAddress:

```
<addresses class="MCRMetaAddress" >
  <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>
```

XML-Syntax des Metadatentyps MCRMetaBoolean

Der Basistyp MCRMetaBoolean beinhaltet eine Liste von Wahrheitswerten mit zugehörigen type Attributen. Folgende Werte sind zulässig:

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

XML-Syntax des Metadatentyps MCRMetaBoolean:

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

Beispiel des Metadatentyps MCRMetaBoolean:

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

XML-Syntax des Metadaten-Basistyps MCRMetaClassification

Der Basistyp MCRMetaClassification dient der Einbindung von Klassifikationen³³ 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. Bitte beachten Sie die Hinweise zur Gestaltung der Kategorie-IDs im vorigen Kapitel!

XML-Syntax des Metadaten-Basistyps MCRMetaClassification:

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

Beispiel des Metadaten-Basistyps MCRMetaClassification:

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

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 mit dem dazugehörigen Kalender gespeichert. Das Datum wird im Format des angegebenen Kalenders abgelegt, auch für die Zeit vor Einführung des selben. Zur Implementierung des Datentyps wurde die frei verfügbare ICU-Library der Firma IBM genutzt. Sie bietet eine Reihe von Kalendern an, die so für diesen Datentyp nun verfügbar sind. Alle Datumsangaben werden zur internen Verarbeitung in MyCoRe in eine Julian Day Number, also eine fortlaufende Tageszahl, umgerechnet. Diese wird neben einer lesbaren Form in dem Datentyp MCRMetaHistoryDate gespeichert.

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.

³³siehe voriges Kapitel

- Ist für von und/oder bis nichts angegeben, werden Standardwerte genommen. Das sind 1..1.4713 BC und 31.12.3999 AD.
- Die Felder ivon bzw. ibis enthalten die korrespondierenden Werte zu von bzw. bis.
- Das calendar-Feld kann die Werte **gregorian** oder **islamic** enthalten.
- Mögliche Notationen für die Datumsangaben sind 01.01.1999 / -01.12.200 / 1035 / 133 BC.

Syntax des Metadaten-Basistyps MCRMetaHistoryDate:

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

Beispiel des Metadaten-Basistyps MCRMetaHistoryDate:

```

<date class="MCRMetaHistoryDate" heritable="...">
  <dates type="written" xml:lang="de">
    <text>4. Jh. v. Chr.</text>
    <von>BC01.01.399</von>
    <ivon>1575694</ivon>
    <bis>BC31.12.300</bis>
    <ibis>1830997</ibis>
    <calendar>gregorian</calendar>
  </dates>
</date>

```

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)

Syntax des Metadaten-Basistyps MCRMetaInstitutionName:

```

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

```

Beispiel des Metadaten-Basistyps MCRMetaInstitutionName:

```
<insts class="MCRMetaInstitutionName" heritable="...">
  <inst xml:lang="de">
    <fullname>Obelix & Co. KG</fullname>
    <nickname>O. & Co.</nickname>
    <property>KG</property>
  </inst>
</insts>
```

XML-Syntax des Metadatentyps MCRMetaISBN

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

Syntax des Metadaten-Basistyps MCRMetaISBN:

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

Syntax des Metadaten-Basistyps MCRMetaISBN:

```
<isbns class="MCRMetaISBN" heritable="...">
  <isbn>1-234-56567-4</isbn>
</isbns>
```

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 unterstützen Format „YYYY-MM“ entspricht.

Syntax des Metadaten-Basistyps MCRMetaISO8601Date:

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

Beispiel des Metadaten-Basistyps MCRMetaISO8601Date:

```
<dates class="MCRMetaISO8601Date" heritable="false">
  <date type="sample">2006-01-16T13:20:30.85+01:00</date>
</dates>
```

XML-Syntax des Metadatentyps MCRMetaLangText

Der Basistyp MCRMetaLangText dient der Speicherung einer Liste von Textabschnitten mit zugehöriger Sprachangabe. Über das form Attribut kann noch spezifiziert werden, in welcher Form der Text geschrieben ist.

XML-Syntax des Metadaten-Basistyps MCRMetaLangText:

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

Beispiel des Metadaten-Basistyps MCRMetaLangText:

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

XML-Syntax der Metadatentypen MCRMetaLink und MCRMetaLinkID

Der Basistyp MCRMetaLink wurde geschaffen, um eine Verknüpfung verschiedener MyCoRe-Objekte untereinander zu realisieren. Außerdem können hier genauso Verweise auf beliebige externe Referenzen abgelegt werden. Der Typ MCRMetaLink ist eine Implementation des W3C XLink Standards³⁴. Auf dieser Basis enthält der MyCoRe-Metadatentyp zwei Arten von Links - eine Referenz und einen bidirektionalen Link. Bei beiden werden jedoch in MCRMetaLink nicht alle Möglichkeiten der XLink Spezifikation ausgeschöpft, da dies für die in MyCoRe benötigten Funktionalitäten nicht erforderlich ist.

Im Referenztyp ist das Attribut xlink:type='locator' immer anzugeben. Die eigentliche Referenz wird im xlink:href Attribut notiert. Dabei kann die Referenz eine URL oder eine MCRObjektID sein. Daneben können noch weitere Informationen im

³⁴siehe 'XLM Linking Language (XLink) Version 1.0'

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.

XML-Syntax des Metadaten-Basistyps MCRMetaLink:

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

Beispiel des Metadaten-Basistyps MCRMetaLink:

```
<urls class="MCRMetaLink" heritable="false">
  <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"\>
      </url>
  </url>
</urls>
```

XML-Syntax des Metadatentyps MCRMetaNBN

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

Syntax des Metadaten-Basistyps MCRMetaNBN:

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

Beispiel des Metadaten-Basistyps MCRMetaNBN:

```
<nbns class="MCRMetaNBN" heritable="...">
  <nbns>urn:nbns:1-2334</nbns>
</nbns>
```

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.

XML-Syntax des Metadaten-Basistyps MCRMetaNumber:

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

Beispiel des Metadaten-Basistyps MCRMetaNumber:

```
<masse class="MCRMetaNumber" heritable="false">
  <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>
```

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.

XML-Syntax des Metadaten-Basistyps MCRMetaPersonName:

```

<tag class="MCRMetaPersonName" heritable="...">
  <subtag type="..." xml:lang="..">
    <firstname>...</firstname>
    <callname>...</callname>
    <surname>...</surname>
    <fullname>...</fullname>
    <academic>...</academic>
    <peerage>...</peerage>
    <prefix>...</prefix>
  </subtag>
  ...
</tag>

```

Beispiel des Metadaten-Basistyps MCRMetaPersonName:

```

<tag class="MCRMetaPersonName" heritable="true">
  <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>

```

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.

XML-Syntax des Metadaten-Basistyps MCRMetaXML:

```

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

```

Beispiel für die Definition dieses Datentyps in der Datamodel-Datei:

```

<!-- beliebiges XML-Objekt -->
<element name="teixmls" minOccurs="0" maxOccurs="1">
  <mcrmetaxml name="teixml" class="MCRMetaXML"
    minOccurs="1" maxOccurs="1"/>
</element>

```

und ein Beispiel mit Metadaten zum Metadaten-Basistyp MCRMetaXML:

```

<teixmls class="MCRMetaXML">
  <teixml inherited="0">
    <TEI>
      <teiHeader>
        <title>Text Encoding Initiative, ein Dokumentenformat
          zur Kodierung und den Austausch von Texten
        </title>
      </teiHeader>
    </TEI>
  </teixml>
</teixmls>

```

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 `servacls` 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.

XML-Syntax des `service` XML-Knotens:

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

7.1.4 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-Syntax des *Derivate*-Datenmodells:

```
<?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>
```

- 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 Tipps und Problembehebung

8.1 XSLT in ANT geht nicht

Beim Einsatz von SuSE Linux 10.1 kann es vorkommen, dass die Erstellung der MyCoRe-Schema-Dateien aus ANT heraus scheitert. In diesem Falle sollte ant-trax-1.6.5.x nachinstalliert werden. Es ist darauf zu achten, dass trax ANT in der Datei */etc/ant.conf* bekannt gemacht wird.

```
OPT_JAR_LIST="$OPT_JAR_LIST regexp ant/ant-apache-regexp" # RPM package  
regexp
```

```
OPT_JAR_LIST="$OPT_JAR_LIST jaxp_transform_impl ant/ant-trax" # RPM package  
trax
```

8.2 Java Console unter Linux Firefox

Unter der Linux-Version des Firefox wird die Java Console nicht direkt angeboten. Diese ist aber zur Fehlersuche im Applet recht hilfreich. Sie kann gesondert gestartet werden mit:

```
/usr/lib/jvm/java-1.5.0-sun-1.5.0_07/bin/ControlPanel
```

9 Anhang

9.1 Abbildungsverzeichnis

Abbildung 4.1: MyCoRe-Schichtenmodell.....	11
Abbildung 5.1: Funktionsschema des SimpleWorkflow.....	29
Abbildung 5.2: Auswahl der zu bearbeitenden Klassifikation.....	34
Abbildung 5.3: Einzelne Kategorien einer Klassifikation bearbeiten.....	36

9.2 Tabellenverzeichnis

Tabelle 2.1: Softwarevoraussetzungen.....	5
Tabelle 2.2: Umgebungsvariablen unter Windows.....	5
Tabelle 2.3: Umgebungsvariablen unter Unix.....	5
Tabelle 3.1: Inhalt des MyCoRe-Kerns.....	8
Tabelle 4.1: Verzeichnisse des Build-Prozesses.....	14
Tabelle 4.2: Beispieldaten im CVS.....	16
Tabelle 5.1: Beispielgruppen in DocPortal.....	17
Tabelle 5.2: Beispielbenutzer in DocPortal.....	18
Tabelle 5.3: Parameter des MCRStartEditorServlets.....	30
Tabelle 5.4: Mögliche Aktionen mit dem MCRStartEditorServlet auf dem Plattenbereich.	31
Tabelle 5.5: Mögliche Aktionen mit dem MCRStartEditorServlet im Server.....	33
Tabelle 6.1: Feste Test-Instanzen für das MyCoRe-Beispiel.....	40
Tabelle 7.1: Aufbau der MCRObjektID.....	49
Tabelle 7.2: MyCoRe-Basisdatentypen.....	55