MyCoRe User Guide

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

Release 0.9

27. April 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

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

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

ACHTUNG, diese Dokumentation basiert auf dem release 0.9 der MyCoRe-Software!

1.Voraussetzungen für eine MyCoRe Anwendung

1.1 Vorabbemerkungen

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

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

Tabelle 1: MyCoRe Komponentenübersicht für IBM CM

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

Tabelle 2: MyCoRe Komponentenübersicht für freie Lösung

1.2 Allgemein benötigte Software

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

- Java SDK – es ist mindestens ein SDK 1.3.1 erforderlich. Java wird normalerweise mit dem Betriebssystem ausgeliefert. Eine Installation ist in den entsprechenden Abschnitten beschrieben.
- BASH (erforderlich unter Unix-Systemen) – Alle zusätzlichen Unix-Shell-Scripts basieren auf der bash-Shell. Es ist also sehr sinnvoll, diese auf dem System mit zu installieren, sofern sie noch nicht vorhanden ist.
- CVS-Client (erforderlich) – Dieser dient dem Checkout des aktuellen MyCoRe-Codes und ist für den Einbau von Bug-Fixes unerlässlich. CVS ist in der SuSE-Linux-Distribution enthalten. Für andere

Systeme muss es von einem separaten Server[1] nachgeladen werden.

- ANT (erforderlich) – Unter Linux ist dieses Tool in den Distributionen enthalten, für die anderen Systeme muss es vom Server des Apache-Projektes[2] geholt werden. Es sollte mindestens Version 1.5.x zum Einsatz kommen.
- ImageMagick (optional) – Ein mächtiges Bildkonverter-Programm, welches bei der Bearbeitung großer Bildmengen sehr hilfreich ist. Auch dieses Produkt ist in Linux-Distributionen mit enthalten.

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

1.3 Hinweise zur Installation des IBM Content Manager 8.2

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

1.4 Hinweise zu vorbereitenden Installationen freier Datenbanken, XML:DB's, Text-Suchmaschinen und Web–Komponenten

1.4.1 Installation der Komponenten unter Linux

Die folgende Beschreibung erläutert die vorbereitenden Arbeiten unter SuSE Linux. Getestet wurde unter der Version 9.0. Es ist besonders aus Sicherheitsgründen besonders zu empfehlen, auch die vom Distributor angebotenen Updates für die nachfolgenden Komponenten nachzuinstallieren. Sollten für RedHat Abweichungen auftreten, so werden diese gesondert vermerkt.

1.4.1.1 Installation von Java

Als erster Schritt ist der Java-Compiler J2SE 1.4.1 oder höher zu installieren. JRE und SDK befinden sich unter in der SuSE in der Distribution. Zum Test unter 9.0 wurden Java 1.4.2 verwendet. Probleme mit höheren Versionen sind bisher nicht bekannt.

- **java2-jre-1.4.2-35**
- **java2-1.4.2-35**

Wenn allerdings SuSE von einem ftp Server installiert wird, so ist Java nicht standardmäßig enthalten. In diesem Fall gehen Sie bitte wie folgend vor:

1 http://cvshome.org/
2 http://ant.apache.org/

1. Holen sie bitte die Java Linux Version von der http://www.blackdown.org [3] Homepage.
2. Installieren Sie den Download kann beispielsweise nach */usr/local*.
3. `chmod +x j2sdk-1.4.2-<VERSION>-linux-<ARCH>-<GCC>.bin`
4. `../j2sdk-1.4.2-<VERSION>-linux-<ARCH>-<GCC>.bin`
5. Damit die Pfad Variablen alle richtig gesetzt werden eine Vorlage unter */etc/java* entsprechend editieren und mit `setDefaultJava` als root die Links richtig setzten.
6. Nun prüfen sie bitte bei Verwendung von JDK 1.4.x noch, mit welcher XALAN-Version sie arbeiten. Diese muss gleich der von MyCoRe mitgelieferten Version sein, damit Inkonstitenzen vermieden werden! **Ist dies nicht der Fall, beachten Sie bitte die Hinweise unter Punkt 2.2.** Der Test geschieht mit **java org.apache.xalan.xslt.EnvironmentCheck**
7. Zudem sollte man in der Datei */etc/profile.local*[4] noch die Variable *ANT_HOME*[5] gesetzt werden.

Die weiteren Beschreibungen gehen davon aus, dass die Applikation unter dem Nutzer **mcradmin** läuft. Richten sie diesen ggf. noch ein.

1.4.1.2 Die Installation von MySQL

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

1. Installieren Sie aus Ihrer Distribution die folgenden Pakete und danach ggf. noch vom Hersteller der Distribution per Netz angebotene Updates. Die angegebenen Versionsnummern sind nur exemplarisch.
   - **mysql-4.0.15-9**
   - **mysql-shared-4.0.15-9**
   - **mysql-client-4.0.15-9**
   - **mysql-devel-4.0.15-9**
   - **mysql-bench-4.0.15-9**
   - **mysqlcc-0.9.2-66**
2. Die Dokumentation steht nun unter */usr/share/doc/packages/mysql*.
3. Führen Sie als **root** das Kommando `rcmysql start` aus.
4. Führen Sie als **root** das Kommando `/usr/bin/mysqladmin -u root password new-password` aus.
5. Die folgende Sequenz sorgt dafür, dass der MyCoRe-User **mcradmin** alle Rechte auf der Datenbank hat. Dabei werden bei der Ausführung von Kommandos von **localhost** aus keine Passwörter abgefragt. Von anderen Hosts aus muss *ein-password* eingegeben werden.
   - `mysql -user=root -p`*`PASSWORD`* `mysql`
   - `GRANT ALL PRIVILEGES ON *.* TO mcradmin@localhost WITH GRANT OPTION;`

3 ftp://ftp.informatik.hu-berlin.de/pub/Java/Linux/
4 Falls diese noch nicht existiert einfach anlegen.
5 export ANT_HOME=/opt/jakarta/ant

   - `GRANT ALL PRIVILEGES ON *.* TO mcradmin@'%' IDENTIFIED BY 'ein-passwort' WITH GRANT OPTION;`
   - `quit`
6. Ist das Password einmal gesetzt, müssen Sie zusätzlich die Option -p verwenden.
7. Zum Verifizieren, ob der Server läuft nutzen Sie `mysqladmin version` und `mysqladmin variables`.
8. Jetzt können Sie die Datenbasis für MyCoRe mit nachstehendem Kommando anlegen.
   - `mysqladmin –u mcradmin create mycore`
9. Falls weitere Benutzer noch das Recht auf Selects von allen Hosts aus haben sollen, verwenden Sie die Kommandos
   - `mysql –user=mcradmin –pPASSWORD mycore`
   - `GRANT SELECT ON mycore.* TO mycorenutzer@'%';`
   - `quit`

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

1.4.1.3 Die Installation der freien XML:DB eXist

eXist ist eine frei verfügbare XML:DB, welche die entsprechenden Interfaces implementiert. Für MyCoRe wurde ein auf diesen Schnittstellen basierender Persitence-Layer implementiert. So ist die Nutzung von eXist direkt möglich.

Vorbedingungen

Für das folgende Szenario ist darauf zu achten, dass der Tomcat bzw. Websphere Server nicht auf den Port 8080 hört, da es ansonsten zu einem Konflikt mit der hier vorgestellten Installation kommt, weil der eXist servlet Container ebenfalls standardmäßig auf den port 8080 hört. Den Port für Tomact ändern sie in der *tomcatinstalldir/conf/server.xml*. Ebenfalls wird die in Abschnitt 1.4.1.2 beschriebene MySQL-Installation vorausgesetzt. Desweiteren sollte die aktuellste Version von MyCoRe und dem MyCoRe Sample auf ihrem Rechner in $MYCORE_HOME bzw. $MYCORE_SAMPLE_HOME liegen.

Installation

1. Download der aktuellen Version von eXist[6]. Achtung, da dieses Produkt noch stark im Wachsen ist, sollte hier ggf. die letzte Version geholt werden (CVS). Zum Test kam Version eXist-1.0 .
2. Entpacken Sie die Distribution in ein entsprechendes Verzeichnis, z. B. unter */home/mcradmin* (eXist-installdir).
3. Entfernen Sie zur Nutzung des Stand-Alone-Servers den Kommentar aus der Zeile `uri=xmldb:exist://localhost:8081` im File *client.properties*
4. Kommentieren Sie folgende Zeile aus.
   `uri=xmldb:exist://localhost:8080/exist/xmlrpc`
   Dies Einstellung wird zum Beispiel für die direkte Einbindung in Tomcat verwendet, wenn man über xmlrpc auf eXist zugreifen möchte.

6 http://exist-db.org

5. Starten Sie den Server mit `<eXist-installdir>/bin/startup.sh`
6. unter der URL `http://localhost:8080/exist/index.xml` sollte jetzt die eXist-Homepage erscheinen. Dies nur als Test.
7. Wenn alles okay ist, starten Sie `<eXist-installdir>/bin/server.sh`
8. Anschliessend können Sie auch den Client mit `<eXist-installdir>/bin/client.sh` starten. Hier sollten Sie dem Admin-User ein Password spendieren.

1.4.1.4 Die Nutzung der freien Text-Suchmaschine Lucene zur Volltextsuche

Um die frei verfügbare Text-Suchmaschine für die Volltextsuche Lucene des Apache Jakarta Projektes[7] zu benutzen, bedarf es keiner zusätzlichen Installationen. Derzeit ist die Version Lucene 1.3 Final Released aktuell. Die erforderliche ***lucene-1.3-final.jar*** Datei wird unter dem MyCoRe-Kern im Verzeichnis *~mycore/lib* mitgeliefert. Zur Nutzung ist lediglich die weiter hinten beschriebene Konfiguration in der MyCoRe-Anwendung nötig.

Im derzeitigen Entwicklungsstand können mit der MyCoRe Variante auf Basis freier Software nur PDF-Dateien textindiziert werden. Hierzu muss noch das Tool **xpdf** installiert werden. Es sollte in den gängigen Linux-Distributionen enthalten sein. Installieren Sie also

- **xpdf-2.02pl1-61**

1.4.1.5 Die Installation von Apache 2

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

- **apache2-2.0.48-9**
- **apache2-doc-2.0.48-9** (dieses Paket ist optional)

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

7 http://jakarta.apache.org/lucene/docs/index.html

1.4.1.6 Die Installation von Tomcat 4

Wie Apache, dessen Installation im vorigen Abschnitt beschrieben wurde, ist auch Tomcat als Servlet-Engine im Linux-Umfeld ein Quasi-Standard. Für die Arbeit der Web-Anwendung des MyCoRe-Projektes wird dieses Tool zwingend benötigt. Getestet wurde unter SuSE 9.0 mit folgenden Komponenten:

- **jakarta-tomcat-4.1.27-32**
- **apache2-jakarta-tomcat-connectors-4.1.27-32**

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

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

Weiterhin ist noch die richtige Gruppenzugehörigkeit für die Tomcat-Installation festzulegen.
- `chown wwwrun:nogroup -R /opt/jakarta/tomcat`

Mit `rctomcat [start; stop; restart]` als **root** kann der Server gestartet gestoppt bzw. neu gestartet werden und über `rctomcat status` wird der aktuelle Staus abgefragt. Nach erfolgreichem starten des Tomcat Servers sollte er unter der Standardadresse `http://localhost:8080/examples` erreichbar sein. Bitte beachten Sie aber, dass der standardmäßige Port 8080 auch von eXist (falls Sie dies verwenden) verwendet wird. In diesem Fall müssen Sie in der Konfigurationsdatei *server.xml* einen anderen Port auswählen. Die Dokumentation steht unter `http://localhost:8080/tomcat-docs`. Die Einbindung von Tomcat in virtuelle Hosts ist weiter unten beschrieben.

1.4.2 Installation der Komponenten unter MS Windows

1.4.2.1 Die Installation der freien XML:DB eXist

eXist ist eine frei verfügbare XML:DB, welche die entsprechenden Interfaces implementiert. Für MyCoRe wurde ein auf diesen Schnittstellen basierender Persistence-Layer implementiert. So sollte die Nutzung von eXist direkt möglich sein.

1. Download der aktuellen Version von eXist[8]. Achtung, da dieses Produkt noch stark im Wachsen ist, sollte hier die letzte Version geholt werden. Zum Test kam Version eXist-1.0 .
2. entpacken Sie die Distribution in ein entsprechendes Verzeichnis.
3. Entfernen Sie zur Nutzung des stand-alone-Servers den Kommentar aus der Zeile
   `uri=xmldb:exist://localhost:8081` in *<installdir>\eXist-1.0\client.properties*
4. Starten Sie *<installdir>\eXist-1.0\bin\server.bat*
5. Anschließend können Sie auch den Client mit
   `<installdir>\eXist-1.0\bin\client.bat`
   starten. Hier sollten Sie dem Admin-User ein Password spendieren.

8 http://exist-db.org/

2.Download und Installation des MyCoRe Kerns

2.1 Download des MyCoRe Kerns

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

cvs -d :pserver:anoncvs@server.mycore.de:/cvs checkout mycore

Nach dem erfolgreichen Checkout erhalten Sie unter dem Verzeichnis *~/mycore* folgende Dateistruktur. Es ist erforderlich, diese Pfadangabe in der Umgebungsvariablen MYCORE_HOME abzulegen.

Relativer Pfad	**Inhalt**
bin	Das Verzeichnis für die Shell-Scripts des Kerns.
bin/build.cmd	Startet den Build-Prozess unter einem Windows-System.
bin/build.sh	Startet den Build-Prozess unter einem Unix-System.
bin/build.properties	Konfigurationsdatei für den Kern, welche die Pfade zu den verwendeten Datenspeichern enthält.
build.xml	Die Konfigurationsdatei für den Build-Prozess.
documentation	Enthält alle MyCoRe-Dokumentationen.
documentation/StartingGuide	Eine kurze Einleitung in das Projekt.
documentation/UserGuide	Das Handbuch für die Standard-Installation von MyCoRe.
Documentation/QuickGuide	Einige Kurzbeschreibungen zu ganz bestimmten Spezialthemen rund um MyCoRe.
lib	Java-Archive von Komponenten, welche in MyCoRe genutzt werden.
license.txt	Die verbindliche Lizenz von MyCoRe.
schema	XML Schema Dateien des MyCoRe-Kerns.
sources/sources/org	Verzeichnis des Java-Quellcodes. Eine Beschreibung der Pakete steht im ProgrammerGuide.
stylesheets	XSLT Stylesheets des MyCoRe-Kerns.

Tabelle 3: Aufbau des MyCoRe-Kerns

2.2 Konfiguration und Übersetzten des Kerns

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

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

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

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

   `build.sh info` bzw. `build.cmd info`

   ausführen. Das Ant Build Tool zeigt Ihnen daraufhin die verwendeten JDK- und Ant-Software-Versionen und den generierten CLASSPATH und LIBPATH (für Unix Systeme) an.

5. Sollten Sie festgestellt haben, dass Ihr JDK ab 1.4.x eine andere Xalan-Version benutzt (ab 1.4.x ist Xalan im SDK), führen Sie bitte folgende Kommandos aus und prüfen Sie danach Ihr System erneut.
   - `cd $JAVA_HOME/jre/lib`
   - `mkdir endorsed`
   - `cd endorsed`
   - `cp $MYCORE_HOME/lib/xerces* .`
   - `cp $MYCORE_HOME/lib/xalan* .`
   - `cd $JAVA_HOME/lib`
   - `ln –s ../jre/lib/endorsed endorsed`

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

   `build.sh usage` bzw. **`build.cmd usage`**

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

   `build.sh jar` bzw. **`build.cmd jar`**

   Dabei entsteht, abhängig von dem von Ihnen gewählten Datenbank-System zur Speicherung der XML-Daten eine Jar-Datei *lib/mycore-for-[cm8—xm ldb].jar*.

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

   `build.sh javadocs` bzw. **`build.cmd javadocs`**

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

3.Die MyCore Beispielanwendung

3.1 Grundlegender Aufbau und Ziel der Beispielanwendung

3.1.1 Allgemeines

Um die Funktionsweise des MyCoRe-Kernes zu testen wurde eine Beispielanwendung basiered auf diesem Kern entwickelt. Sie soll dem Anwender eine voll funktionsfähige Applikation in die Hand geben, welche eine Vielzahl von Komponenten des MyCoRe-Projektes nutzt und deren Arbeitweise klar werden lässt. Um die Anwendung, im weiteren MyCoRe-Sample genannt, gleichzeitig sinnvoll einsetzten zu können, wurde als Beispielszenario ein Dokumentserver gewählt, wie er bei vielen potentiellen Nutzern zur Anwendung kommt. Auch soll das MyCore-Sample die Nachfolge des erfolgreichen MILESS-Dokumentservers sein und den Migrationspfad zu MyCoRe hin aufzeigen.
Analog zum MILESS werden auch hier die Metadaten in drei Gruppen aufgeteilt und von den eigentlichen multimedialen Objekten getrennt behandelt. Zur Modellierung des Datenmodells für die Multimedia-metadaten wurde das allgemein verbreitete Dublin Core Datenmodell herangezogen und umgesetzt. Zur Realisierung wurden folgende Metadaten-Objekte entwickelt:

Document - Dieses Objekt beinhaltet die eigentlichen Metadaten des Multimedi-Objektes, in unserem Fall im Dublin Core Format, sowie Verweise auf extern gehaltene Daten wie die drei im weiteren genannten. Das Datenmodell besteht aus 15 Feldern, zur Speicherung verschiedener Datentypen gibt es atomare Grundkomponenetn. Weiterhin beinhalten die Metaddaten noch Teile zu ihrer Struktur und Verwaltung.

Klassifikation - Hier können lineare oder hierarchich strukturierte Klassifikationen mit ihren Kategorien abgelegt werden. Jeder Verweis auf eine solche Kategorie aus einem Dokument heraus erfolgt mittels der ID der selben und natürlich der ID der Klassifikation als Identifiziererpaar. Für die Gestaltung der Klassifikationen wurde ein eigens Datenmmodel entwicklet.

LegalEntity - Auch diese Metadaten sind analog dem **Document** aus atomaren Komponenten zusammengesetzt und repräsentieren natürliche und juristische personen, wie Autoren oder Einrichtunge.

Derivate - Hinter diesem Begriff verbergen sich Darstellungsformen der eigentlichen multimedialen Objekte. Beispielsweise können das verschieden Dateiformate eines Images sein. Diese werden an das eigentliche Dokument gebunden.

Eine Übersicht zu den Beziehungen der Objektteile zueinander soll die folgende Grafik vermitteln. Wir haben uns aus den guten Erfahrungen von MILESS herraus entschlossen, diese Aufteilung beizubehalten, da so Einzelkomponeneten in Projekten schon geladen werden können, bevor andere Teile, z. B. das Scannen, abgeschlossen sind.

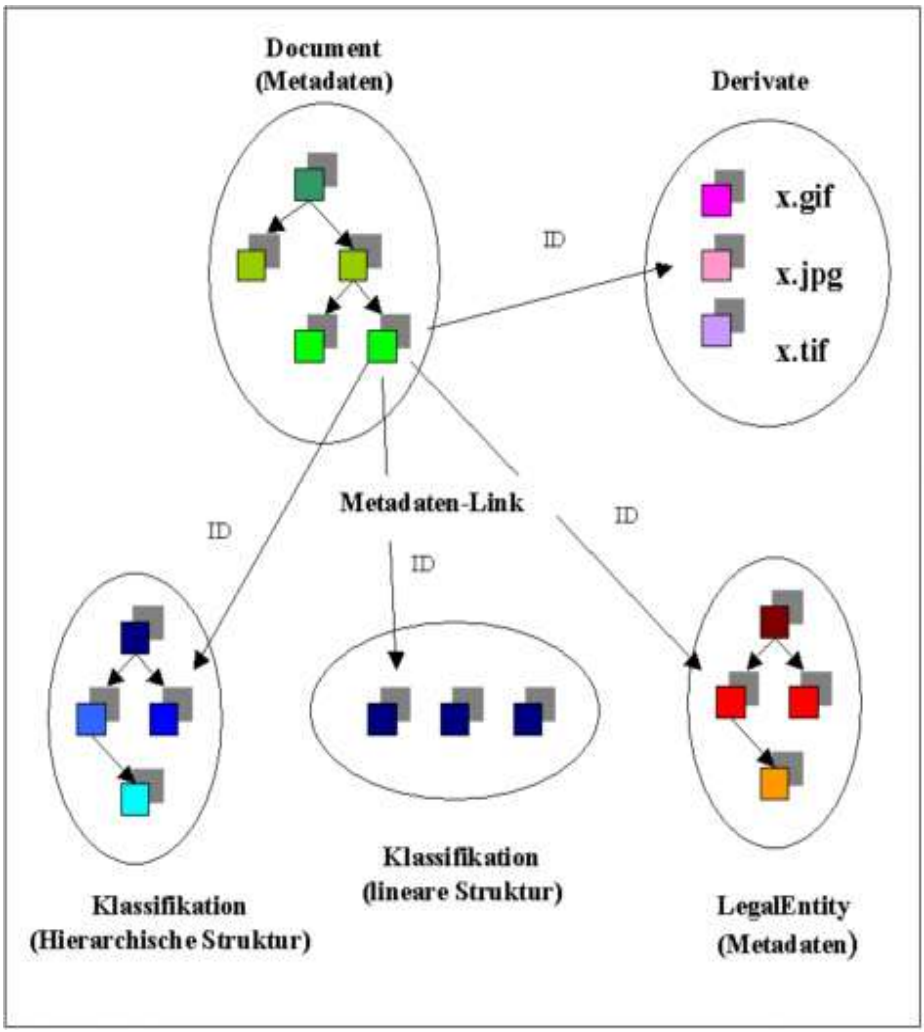

Abbildung 1: Übersicht der Objekte im MyCoRe-Sample

Wie man der Grafik entnehmen kann, ist es möglich, neben Kategorien in Klassifikationen auch Metadaten der Documents und LegalEntities hierarchisch ineinander zu verschachteln. Auf dieser Basis können dann Strukturen wie z. B. die eines Buches abgebildet werden. Alle Beziehungen zwischen den Metadaten sind über eine eindeutige MCRObjectID gelöst.

3.1.2 Die MCRObjectID

Dreh- und Angelpunkt aller Beziehungen von Metadaten und der daran angeknüpften Objekte ist ein eindeutiger Identifizierer. Im Falle des MyCoRe Projektes hat dieser den Namen **MCRObjectID** erhalten. Er hat für alle Matadaten einen einheitlichen Aufbau und wird ständig zur Identifizierung genutzt. Auch das API des Systems bietet die Möglichkeit, diese ID automatisch zu generieren.[9] Der Syntax der **MCRObjectID** ist wie folgend festgelegt:

MCRObjectID = *project_type_number*

project - Dieses Element der ID spezifiziert ein bestimmtes Projekt. Es dient dazu, Daten verschiedener Projekte, welche sich gemeinsam auf einem System befinden, zu unterscheiden.

9Siehe Programmers Guide.

Der Projektname sollte eindeutig sein und besteht aus einem Wort **ohne Sonderzeichen**. Achten Sie immer darauf, dass Sie sich bei Vorhaben mit mehreren MyCoRe-Partnern von Anfang an auf einen Namen pro Projektteilnehmer verständigen. Für unser Sample haben wir **MyCoReDemoDC** ausgewählt. **Dieser Teil der ID ist case sensitive!**

type - Jedem Metadatentyp des Datenmodells ist ein Typbezeichner zuzuordnen. Dieser stellt in MyCoRe die Verbindung von der Konfiguration des Metadatums bis hin zu dessen Suche und Präsentation dar. Der Typname der MCRObjectID wird immer in Kleinbuchstaben (lower case) innerhalb des Systems verwendet. Achten Sie von Anfang an darauf, dies umzusetzen. Im MyCoRe-Projekt gibt es reservierte Typnamen, diese sind

- **class** - für Klassifikationen
- **derivate** - für Derivate

Zur Umsetztung des Datenmodells wurden weiterhin die Typen **document** für Dokumente und **legalentity** für Personen verwendet. *number* - Dieser Teil dient der Identifizierung des Einzelobjektes. Es dürfen nur natürliche Zahlen angegeben werden. Planen Sie Ihre Vergabe der Nummern bei eigenen Objekten sorgfältig. Da die Nummern teilweise als Strings verarbeitet werden, ist es sinnvoll, sie mit einer ausreichenden Zahl von Vornullen zu versehen. Dies erledigt das MyCoRe-System automatisch unter Zuhilfenahme des Konfogurationswertes `MCR.metadata_objectid_number_pattern` im File *mycore.properties.private* .

3.1.3 Das User- und Rechtemodell

3.1.4 Das Sample-Datenmodell

Das Datenmodell des MyCoRe-Samples soll exemplarisch einen kleinen Dokument-Server abbilden, wie er dann für große Lösungen übernommen werden kann. Dabei sollen Multimediale Objekte verschiedenster Art und Herkunft gemeinsam verwaltet werden. Als Grundgerüst für die Dokumente diente uns das Dublin Core Datenmodell, welches in einigen Punkten inhaltlich etwas nach unseren Auffassungen angepasst wurde. Das Personen-Datenmodell ist hingegen derzeit propitär. Ggf. kommen hier in einer späteren Version von MyCoRe Standard-Datenmodelle zum Einsatz. Die Klassifikationen sind hierarchisch angelegt, d. h. Kategorien können wieder Unterkategorien beherbergen. Im folgenden werden die einzelnen Felder des Metadatenmodelles beschrieben und es wird aufgezeigt, welche Suchen vorgesehen sind. Dabei bedeutet PS - parametrische Suche, TS - Volltextsuche. Eine genaue Beschreibung der Suchmechanismen finden Sie im Programmers Guide.

3.1.4.1 Die Klassifikationen (Classifications)

Für das Datenmodell sind die folgenden Klassifikationen vorgesehen. Alle Label und Descriptions der einzelnen Kategorien können dabei in mehreren Sprachen angegeben werden.

- **Format** - enthält eine einfache Liste der möglichen Formate des Dokumentes.
- **Typen** - enthält eine einfache Liste der möglichen Typen des Dokumente.
- **Eigner** - enthält eine hierarchiche Liste aller möglichen Eignergruppen eines Dokumentes.

3.1.4.2 Das Dokument (Document)

Bezeichner	Typ	Beschreibung
Title	MCRMetaLangText	Titel des Werkes, alternativer Titel usw. in mehreren Sprachen.
Creator	MCRMetaLinkID	Enthält einen oder mehrere Verweise auf den/die Autor(en) des Werkes.
Subject	MCRMetaClassification	Enthält Verweise auf Klassifikationen , die das Werk in Kategorien einstufen.
Description	MCRMetaLangText	Enthält eine Kurzbeschreibung des Inhaltes, ggf. in mehreren Sprachen.
Publisher	MCRMetaLinkID	Enthält einen oder mehrere Verweise auf den/die an der Publikation beteiligten des Werkes.
Contributor	MCRMetaLinkID	Enthält einen oder mehrere Verweise auf den/die weiteren Beteiligten des Werkes.
Date	MCRMetaDate	Enthält das Erscheinungsdatum des Werkes.
Type	MCRMetaClassification	Enthält einen Verweis auf Klassifikation, die das Werk in eine Type-Kategorien einstuft.
Format	MCRMetaClassification	Enthält einen Verweis auf Klassifikation, die das Werk in eine Format-Kategorien einstuft.
Identifier	MCRMetaLangText	Enthält eine Identifikationsmerkmale wie Inventarnummern usw., ggf. in mehreren Sprachen.
Source	MCRMetaLangText	Enthält eine Quellenangabe, ggf. in mehreren Sprachen.
Keywords	MCRMetaLangText	Enthält eine Schlüsselwortes, ggf. in mehreren Sprachen.
Coverage	MCRMetaLangText	Enthält eine Erstreckung, ggf. in mehreren Sprachen.
Rights	MCRMetaLangText	Enthält eine Rechtebeschreibung, ggf. in mehreren Sprachen.

Tabelle 4: MyCoRe Document Datenmodell

Relativer XPath	parametric search	text search
ID	•	-
label	•	-
structure/derobjects/derobject	•	-
structure/parents/parent	-	-
structure/children/child	-	-
metadata/titles/title	•	•
metadata/creators/creator	•	-
metadata/subjects/subject	•	-
metadata/descriptions/description	•	•
metadata/publishers/publisher	•	-
metadata/contributors/contributor	•	-
metadata/dates/date	•	-
metadata/types/type	•	-
metadata/formats/format	•	-
metadata/identifiers/idetifier	•	-
metadata/sources/source	-	-

Relativer XPath	parametric search	text search
metadata/keywords/keyword	•	•
metadata/coverages/coverage	•	•
metadata/rights/right	•	-
service/servflags/servflag	•	-
services/servdates/servdate	•	-

Tabelle 5: MyCoRe Document Suchschema

3.1.4.3 Die Personendaten (LegalEntities)

Bezeichner	Typ	Beschreibung
Person	MCRMetaPerson	Name der Person mit akademischen Titel, Adelstitel usw.
Corporation	MCRMetaCorporation	Name einer Institution oder Einrichtung mit Abteilungsbezeichnung usw.
Address	MCRMetaAdress	Die Adresse mit Land, Staat, ort, PLZ, Straße und Nummer.
Date	MCRMetaDate	Datumsangaben wie Geburtstag.
Phone	MCRMetaLangText	Telefonnummern
WEB	MCRMetaLangText	Eine oder mehrere URS' s.
eMail	MCRMetaLangText	Eine oder mehrere eMail Adressen.

Tabelle 6: MyCoRe LegalEntity Datenmodell

Relativer XPath	parametric search	text search
ID	•	-
label	•	-
structure/derobjects/derobject	•	-
structure/parents/parent	-	-
structure/children/child	-	-
metadata/persons/person	•	•
metadata/corporations/corporation	•	•
metadata/adresses/address	•	-
metadata/dates/date	•	-
metadata/phones/phone	-	-
metadata/webs/web	-	-
metadata/emails/email	-	-
service/servflags/servflag	•	-
services/servdates/servdate	•	-

Tabelle 7: MyCoRe LegalEntity Suchschema

3.2 Download der Beispielanwendung

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. Das MyCoRe Sample wird für alle unterstützten

Systeme über das CVS Repository ausgeliefert. Das Holen der aktuellen Version erfolgt mit dem Kommando

```
cvs -d :pserver:anoncvs@server.mycore.de:/cvs checkout
     mycore-sample-application
```

Nach dem erfolgreichen Checkout erhalten Sie im Verzeichnis *~/mycore-sample-application* folgende Dateistruktur. Nun muss die Umgebungsvariable MYCORE_SAMPLE_HOME auf die Wurzel des Beispieles gesetzt werden.

Relativer Pfad	**Inhalt**
bin	Das Verzeichnis für die Shell-Scripts des Beispiels.
bin/build.cmd	Startet den Build-Prozess unter einem Windows-System.
bin/build.sh	Startet den Build-Prozess unter einem Unix-System.
build.xml	Die Konfigurationsdatei für den Build-Prozess.
buildaddon.xml	Eine weitere Konfigurationsdatei für den Build-Prozess.
buildtamino.xml	Eine weitere Konfigurationsdatei für den Build-Prozess.
config	Das Verzeichnis mit allen erforderlichen Konfigurationsdateien.
content	Hier sind einige Beispieldaten für das MyCoRe-Sample abgelegt.
content/classification	Die Beispielklassifikationen.
content/documents	Die Beispieldokumente.
content/derivates	Die Beispiel-Derivate.
content/legalentities	Die Beispielpersonen.
content/user	Die Beispielnutzer.
license.txt	Die verbindliche Lizenz von MyCoRe.
schema	XML Schema Dateien des MyCoRe-Beispiels.
sources/sources/org	Verzeichnis des Java-Quellcodes für zusätzliche Klassen. Eine Beschreibung der Pakete steht im ProgrammerGuide.
stylesheets	XSLT Stylesheets des MyCoRe-Beispiels.
unixtools	Weitere hilfreiche Unix-Tools für die Arbeit mit dem Beispiel.
webpages	Statische Web-Seiten des Beispiels.

Tabelle 8: Aufbau des MyCoRe-Beispiels

3.3 Konfiguration zur Arbeit mit den Beispieldaten

3.3.1 Grundlegende Konfigurationen

Dieser Abschnitt beschäftigt sich mit der Konfiguration der Beispielanwendung in allgemeinen Bereichen wie JDBC, Logger, usw. Die MyCoRe Konfigurationen für das Beispiel finden Sie im Verzeichnis *mycore-sample-application/config*. In diesem Verzeichnis finden Sie eine Datei *mycore.properties.priv_template* . Diese müssen Sie nach *mycore.properties.private* kopieren. Sie enthält alle Konfigurationsparameter, die für eine erste Installation und eine direkte Übernahme des Beispiels in eine Produktionsumgebung angepasst werden müssen. Alle Beschreibungen in diesem Punkt beziehen sich auf dieses File.

3.3.1.1 Pfade und Systemumgebung anpassen

Auch für die Zusammenstellung und Installation der Beispiel-Anwendung verwendet MyCoRe das Apache Ant Build-Tool. Entsprechend der Installationsanleitung des Ant-Paketes sollten Sie zunächst die Umgebungsvariablen `JAVA_HOME` und `ANT_HOME` gesetzt haben. Sollten diese Variablen auf Ihrem System noch nicht gesetzt sein, können Sie dies in der Datei `build.sh` (Unix) bzw. `build.cmd` (Windows) nachholen und korrigieren.

Die MyCoRe Beispiel-Anwendung verwendet die Dateien aus dem MyCore Kern, insbesondere die erzeugte Datei *mycore-for[cm8—xmldb].jar* und die Konfigurationsdatei für den Build-Prozess *build.properties*.

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

`build.sh info` bzw. **`build.cmd info`**

ausführen. Das Ant Build Tool zeigt Ihnen daraufhin die verwendeten JDK- und Ant-Software-Versionen und den generierten CLASSPATH und LIBPATH (für Unix Systeme) an. Eine Übersicht über alle wesentlichen Build-Ziele erhalten Sie mit

`build.sh usage` bzw. **`build.cmd usage`**

3.3.1.2 JDBC-Treiber konfigurieren

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

In der Konfigurationsdatei *mycore.properties.private* legen Sie im Parameter **MCR.persistence_sql_driver** fest, welcher JDBC-Treiber verwendet werden soll. Weiterhin müssen Sie die Variable **MCR.persistence_sql_database_url** anpassen, die die JDBC URL für Verbindungen zu Ihrer Datenbank festlegt. Der DB2 Library-Name **LIB** muss durch den aktuellen (z. B. **ICMNLSDB**) ersetzt werden. Analog dazu muss der User **ODBC** bei MySQL durch den entsprechenden Nutzer (z. B. **mcradmin**) ersetzt werden. Beachten Sie dabei insbesondere, dass meist Gross/Kleinschreibung relevant ist! Weiterhin können Sie die minimale und maximale Anzahl der gleichzeitigen Verbindungen zur Datenbank festlegen.

```
# JDBC parameters for connecting to DB2
#MCR.persistence_sql_database_url=jdbc:db2:LIB
#MCR.persistence_sql_driver=COM.ibm.db2.jdbc.app.DB2Driver

# JDBC parameters for connecting to MySQL
MCR.persistence_sql_database_url=jdbc:mysql://localhost/mycore?user=ODBC
MCR.persistence_sql_driver=org.gjt.mm.mysql.Driver

MCR.persistence_sql_init_connections=1
MCR.persistence_sql_max_connections=5
```

3.3.1.3 Debug konfigurieren

Innerhalb des MyCoRe-Projektes wird zum Erzeugen aller Print-Ausgaben für das Commandline-Tool und/oder die Stdout-Logs das externe Paket **log4j**des Apache-Jakarta-Projektes benutzt[10] . Dieses ist mittlerweile ein Quasistandard und ermöglicht eine gezielte Steuerung der Informationen, welche man erhalten möchte. In der Grundkonfiguration in mycore.properties.private ist der Output-Level INFO eingestellt. Eine zweite Standardvorgabe ist DEBUG, diese ist auskommentiert und kann alternativ bei Problemen genommen werden. **log4j** bietet jedoch darüber hinaus noch viele weitere Möglichkeiten, die Sie bitte der Dokumentation zu diesem Produkt entnehmen. Ergänzend sei auch auf das MyCoRe-Konfigurationsfile mycore.properties.logger hingewiesen.

```
# Set the log level and appender for the general logger
MCR.log4j.rootLogger=INFO, stdout
#MCR.log4j.rootLogger=DEBUG, stdout
```

3.3.1.4 Metadaten-Store konfigurieren

Den Typ des zur Laufzeit des Systems zu nutzenden Metadaten-Store konfigurieren haben Sie bereits in der Datei `$MYCORE_HOME/bin/build.properties` über den Parameter `MCR.XMLStore.Type` festgelegt. Dabei sind die Werte **`cm8`** für IBM Content Manager 8, oder **`xmldb`** für eine XML:DB kompatible XML-Datenbank wie eXist oder Tamino (Software AG) möglich[11]. Die MyCoRe Beispiel-Anwendung verwendet automatisch diese Konfiguration, sie müssen nun nur noch die Parameter der einzelnen XML-Stores konfigurieren.

3.3.1.5 Konfiguration von IBM Content Manager 8.2

Falls Sie `MCR.XMLStore.Type=cm8` verwenden, passen Sie in der Datei `mycore.properties.private` die Variablen `MCR.persistence_cm8_*` an. Die Einträge sind eigentlich selbsterklärend, so dass an dieser Stelle auf weitere Erläuterungen verzichtet werden kann.

```
# Special values for the persistence layer
MCR.persistence_cm8_max_connections=2
MCR.persistence_cm8_library_server=ICMNLSDB
MCR.persistence_cm8_user_id=icmadmin
MCR.persistence_cm8_password=????????

# Special values for the text search engine
MCR.persistence_cm8_textsearch_ccsid=819
MCR.persistence_cm8_textsearch_lang=DE_DE
MCR.persistence_cm8_textsearch_indexdir=/home/db2inst1/sqllib/db2ext/indexes
MCR.persistence_cm8_textsearch_workingdir=/home/db2inst1/sqllib/db2ext/indexes
```

Da der Persitence Layer CM8 auf einem Mapping der XML-Daten nach DB2 besteht, müssen die CM8 ItemTypes vor dem Laden der Daten separat angelegt werden. Dies geschiet mittels

`build.sh create.metastore` bzw. `build.cmd create.metastore`

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

11 Zur Nutzung von Tamino konsultieren Sie bitte die gesonderte Dokumentation.

3.3.1.6 Die Nutzung von eXist als XML:DB Backend

Falls Sie `MCR.XMLStore.Type=xmldb` verwenden, passen Sie in der Datei `mycore.properties.private` die Variablen `MCR.persistence_xmldb_driver` und `MCR.persistence_xmldb_database_url` an. Die folgenden Zeilen sind für Nutzer der freien Software eXist gedacht. Sollten Sie eine andere XML:DB benutzen, passen sie die Daten bitte entsprechend den Erfordernissen an.

```
MCR.persistence_xmldb_driver=org.exist.xmldb.DatabaseImpl
MCR.persistence_xmldb_database_url=xmldb:exist://localhost:8081/db/mycore
MCR.persistence_xmldb_database=exist
```

Die folgenden Zeilen sind für Nutzer der freien Software eXist gedacht. Sollten Sie eine andere XML:DB benutzen, passen sie die Daten bitte entsprechend den Erfordernissen an. Starten Sie nun den eXist-Client (*<eXist-installdir>/bin/client.sh* bzw. *cliend.cmd*) und führe Sie folgende Kommandos zum anlegen der Stores unter eXist aus:

```
mkcol mycore
chown guest guest mycore
cd mycore
mkcol legalentity
chown guest guest legalentity
mkcol document
chown guest guest document
mkcol derivate
chown guest guest derivate
quit
```

3.3.1.7 Speicherung von Dokumenten konfigurieren

Neben den Metadaten sind im Sample auch eine Reihe von Dokumenten und Bildern zum Laden in die Beispielanwendung abgelegt.

3.3.1.7.1 File System Store

In der Grundkonfiguration verwendet die Beispiel-Applikation zur Speicherung der Datei-Inhalte der Derivate das lokale Dateisystem. Passen Sie in der Datei `mycore.properties.private` die Variable `MCR.IFS.ContentStore.FS.BaseDirectory` an und erzeugen Sie ein neues, leeres Verzeichnis am angegebenen Ort.

3.3.1.7.2 Lucene Store

Sollten sie in der freien Variante noch die Textindizierung für *.txt und *.pdf mittels Lucene wünschen, so sind noch folgende Dinge zu tun:

1. `build.sh jar`
2. Setzen Sie die nachfolgenden Variablen in `mycore.properties.private` auf die richtigen Werten. Dabei kann das Lucene.BaseDirectory auf das selbe Verzeichnis verweisen wie der File System Store.
   ```
   MCR.PluginDirectory
   MCR.IFS.ContentStore.Lucene.IndexDirectory
   MCR.IFS.ContentStore.Lucene.BaseDirectory
   ```

3. Erzeugen Sie nun noch die eingetragenen Verzeichnisse.
4. Als letztes muss noch in der Datei `ContentStoreSelectionRules.xml` die Sektion für den Lucene-Store aktiviert werden.

3.3.1.7.3 CM8 Store

Die Speicherung der Dokumente im IBM Content Manager bietet die Möglichkeit einer automatischen Textindizierung. Dazu müssen aber alle Konfigurationen für die Nutzung des TextSearch Extenders gesetzt werden. Genaue Hinweise zur Installation finden Sie in einem gesonderten Dokument der Rubrik **Quick Guide**.

3.3.2 Laden der Beispiel-Inhalte

1. Laden Sie die Benutzer, Gruppen und Privilegien für die Beispielanwendung mit
   `build.sh load.users` bzw. **`build.cmd load.users`**
2. Laden Sie die Beispiel-Klassifikationen mit
   `build.sh load.classifications` bzw.
   `build.cmd load.classifications`
3. Laden Sie die Personen-, Autoren- und Kontaktdaten der Beispiel-Anwendung mit
   `build.sh load.legalentities` bzw.
   `build.cmd load.legalentities`
4. Laden Sie die Dokumenten-Metadaten der Beispiel-Anwendung mit
   `build.sh load.documents` bzw.
   `build.cmd load.documents`
5. Laden Sie die Dokumenten-Derivate mit den dazugehörigen Dateien mit
   `build.sh load.derivates bzw. build.cmd load.derivates`

3.4 Arbeiten mit dem MyCoRe Command Line Interface

3.4.1 Erzeugen der Skripte mycore.sh / mycore.cmd

Neben dem MyCoRe-Web-Interface kann für administrative Zwecke das MyCoRe Command Line Interface (CLI) genutzt werden. Zum Aufruf des CLI müssen Sie zunächst die erforderlichen Shell-Skripte erzeugen:

`build.sh scripts` bzw. **`build.cmd scripts`**

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

3.4.2 Aufruf des CommandLineInterface

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

3.4.3 Tests auf der Basis des CommandLineInterface

Nachdem Sie nun die Testdaten geladen haben, besteht schon einmal die Möglichkeit, zu testen, ob die geladenen Daten sich anfassen lassen. Hierzu können sie das ConnamdLineInterface benutzen. Starten Sie `bin/mycore.sh` (Unix) bzw. `bin/mycore.cmd` (Windows) und versuchen Sie zum Beispiel folgende Kommandos:

```
login gandalf
alleswirdgut
list all users
list all groups
list all privileges

query local class /mycoreclass[@ID="MyCoReDemoDC_class_0002"]
query local document /mycoreobject[@ID="MyCoReDemoDC_document_0001"]

save derivate MyCoReDemoDC_derivate_0002 to derivate_2
```

3.5 Die Zusammenarbeit mit anderen MyCoRe-Sample-Installationen

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

3.5.1 Die eigene Installation

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

```
# Configuration for the own host with remote access
MCR.remoteaccess_remote_query_class=org.mycore.backend.remote.MCRServletCommunic
ation
MCR.remoteaccess_remote_host=pcclu02.rz.uni-leipzig.de
MCR.remoteaccess_remote_protocol=http
MCR.remoteaccess_remote_port=8080
MCR.remoteaccess_remote_query_servlet=/mycoresample/servlets/MCRQueryServlet
MCR.remoteaccess_remote_ifs_servlet=/mycoresample/servlets/MCRFileNodeServlet
```

3.5.2 Standard-Server des MyCoRe-Projektes

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

Alias	URL	Port	Standort
mcrLpzHHttp	http://ibmdlh.rz.uni-leipzig.de/mycoresample	8080	Uni Leipzig
mcrGreiHttp	http://mycoresample.rz.uni-greifswald.de/mycoresample	80	Uni Greifswald

Tabelle 9: Feste Test-Instanzen für das MyCoRe-Beispiel

3.6 Erzeugen und Konfigurieren der Web-Anwendung

3.6.1 Erzeugen der Web-Anwendung

Durch Eingabe von

`build.sh webapp` bzw. **`build.cmd webapp`**

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

`build.sh war` bzw. **`build.cmd war`**

aufrufen.

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

3.6.2 Konfiguration des Web Application Server

3.6.2.1 Tomcat

Die grundlegende Installation von Tomcat wurde bereits beschrieben. Nun soll auf dieser Basis das die WEB-Anwendung des MyCoRe-Samples installiert werden. Dabei ist an dieser Stelle nur ein einfaches Szenario auf der Basis der Tomcat-Grundinstallation beschrieben. Für die Konfiguration komplexerer Modelle, z. B. mehrere Applikationen nebeneinander, gibt es weiter hinten in diesem Dokument eine ausführliche Anleitung. **Achtung, auch Tomcat bringt eigene Xeces/Xalan Archive mit sich. Sollten Sie diese im JDK bereits ersetzt haben, so müssen Sie die mit MyCoRe gelieferte Version auch unter TOMCAT_HOME/common/endorsed stellen und die alten Dateien löschen.**

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

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

3.6.2.2 Websphere

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

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

Öffnen Sie nun eine Web-Browser mit der URL `http://<hostname>:9090/admin` und melden Sie sich an. Nun sind folgende Schritte durchzuführen:

1. (linke Seite) **Server Application Server**
2. (rechte Seite) **NEW**
3. (rechte Seite) Server Name **mycoresample NEXT**
4. (rechte Seite) **FINISH**
5. (linke Seite) **Applications Install New Applications**
6. (rechte Seite) **Server Path** Pfad zum File mycoresample.war eintagen **/mycoresample** im Feld Context Root eintragen **NEXT**
7. (rechte Seite) Preparing for application installation **NEXT**
8. (rechte Seite) Step 1 **NEXT**
9. (rechte Seite) Step 2 **NEXT**
10. (rechte Seite) Step 3 Auswählen **mycoresample** in der Checkbox dann auswählen der Zeile mit **server=mycoresample APPLY**
11. (rechte Seite) Step 3 **NEXT**
12. (rechte Seite) Step 4 **FINISH**
13. (linke Seite) **Server Application Server**
14. (rechte Seite) **mycoresample Process Definition Process Execution**
15. (rechte Seite) User auf **mcradmin** setzen
16. (rechte Seite) Group auf **mcr** setzen
17. (rechte Seite) **APPLY OK**
18. (rechte Seite) **Java Virtual Machine Classpath**
19. (rechte Seite) einfügen der Pfade
    /home/db2inst1/sqllib/java12/db2java.zip:
    /usr/lpp/cmb/lib/cmbsdk81.jar:
    /usr/lpp/cmb/cmgmt
20. (rechte Seite) **APPLY OK**
21. (rechte Seite) oben auf den Text **save** klicken
22. (rechte Seite) **SAVE**
23. (linke Seite) **Environment Update Web Server Plugin OK**
24. **Logout**

Nun muss der Application Server gestartet werden:

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

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

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

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

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

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

4.Vom Sample zum eigenen Dokumenten-Server

4.1 Allgemeines

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

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

- Kopieren des MyCoRe-Samples in einen gesonderten Verzeichniszweig (z. B. *~/docserv*. Es ist von Vorteil diesen Pfad auch in einer Environment Variable abzulegen (z. B. export DOCSERV_HOME=~/docserv).
- Legen Sie für Ihre Anwendung die Namen für die Datenbank-Tabellen, XML- und IFS-Stores fest und ändern Sie die Konfigurations-Dateien entsprechend. Erzeugen Sie nun die Data Stores und XML-Schema Files.
- Modifizieren Sie die User-Daten entsprechend ihren Anforderungen und laden Sie das User-System.
- Erstellen Sie sich alle benötigten Klassifikationen (z. B. eine aller Ihrer Einrichtungen) und laden Sie diese.
- Laden Sie nun erste, per Hand oder Script erstellte Metadaten-Test-Files und laden Sie auch diese. Nun können Sie auf Commandline-Ebene schon mittels mycore.sh bzw. mycore.cmd Anfragen an das System stellen und erste Test durchführen.
- Legen Sie die Stores für die Multimedia-Objekte fest, konfigureieren Sie diese und laden Sie einige Beispiele zum Test.
- Legen Sie nun eine URL für ihren Server fest und setzen Sie einen Web-Server (WebSphere oder Apache/Tomcat) auf.
- Installiern Sie Ihre Anwendung im Web-Server und modifizieren Sie schrittweise die Präsentation nach Ihren Bedürfnissen.
- Testen und integrieren Sie die in diesem Kapitel beschriebenen weiterfführenden Funktionalitäten, welche nicht im MyCoRe-Sample enthalten sind.

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

4.2 XML-Syntax des Datenmodells

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

4.2.1 Das Klassifikationen-Datenmodell

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

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

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

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

Abbildung 2: XML-Syntax eines Klassifikations-Objektes

4.2.2 Das Metadatenmodell

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

Um die Metadaten besser auf unterschiedlichen Datenspeichern ablegen zu können, wurde ein System von XML-Strukturen entwickelt, das es gestattet, neben den eigentlichen Daten wie Titel,

Autor usw. auch Struktur- und Service-Informationen mit abzulegen. Die eigentlichen Nutzerdaten sind wiederum typisiert, was deren speicherunabhängige Aufzeichnung erheblich vereinfacht. Es steht dem Entwickler einer Anwendung jedoch frei, hier bei Bedarf weitere hinzuzufügen. Im Folgenden soll nun der Aufbau der Metadatenobjekte im Detail beschrieben werden. Zum Verständnis des Samples sei hier auch auf den vorigen Abschnitt verwiesen. Die Metadaten werden komplett in XML erfasst und verarbeitet. Für die Grundstrukturen und Standardmetadatentypen werden seitens MyCoRe bereits XMLSchema-Dateien mitgeliefert.

4.2.2.1 XML-Syntax eines Metadatenobjektes

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

Abbildung 3: XML-Syntax eines Metadaten-Objektes

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

4.2.2.2 XML-Synatx des XML-Knotens structure

Im XML-Knoten **structure** sind alle Informationen über die Beziehung des Metadatenobjektes zu anderen Objekten abgelegt. Es werden derzeit die folgenden XML-Daten unter diesem Knoten abgelegt. Die Tag-Namen parents/parent, children/child und derobjects/derobject sind reserviert und **dürfen NICHT anderweitig verwendet werden!** Alle Sub-Knoten haben einen Aufbau wie für MCRMetaLinkID beschieben.

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

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

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

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

Abbildung 4: XML-Syntax eines structure XML-Knotens

4.2.2.3 XML-Syntax des XML-Knotens metadata

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

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

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

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

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

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

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

Abbildung 5: XML-Syntax eines metadata XML-Knotens

4.2.2.4 MyCoRe Metadaten-Basistypen

In MyCoRe gibt es eine Reihe von vordefinierten XML-Datenstrukturen zur Abbildung bestimmter mehr

oder minder komplexer Daten. Diese Strukturen bilden die MyCoRe-Datentypen, welche von der Dateneingabe bis hin zur Validierung und Datenpräsentation für einen einheitlichen Umgang mit den Daten sorgen. Dabei ist zwischen einfachen, recht atomaren Typen und anwendungsspezifischen komplexen Typen zu unterscheiden. Eine Auflistung finden Sie in nachfolgender Tabelle.

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

Tabelle 10: MyCoRe-Datentypen

4.2.2.4.1 XML-Syntax des Metadaten-Basistyps MCRMetaAddress

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

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

Abbildung 6: XML-Syntax des Metadaten-Basistyps MCRMetaAddress

Abbildung 7: Beispiel des Metadaten-Basistyps MCRMetaAddress

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

4.2.2.4.2 XML-Syntax des Metadaten-Basistyps MCRMetaBoolean

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

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

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

Abbildung 8: XML-Syntax des Metadaten-Basistyps MCRMetaBoolean

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

Abbildung 9: Beispiel des Metadaten-Basistyps MCRMetaBoolean

4.2.2.4.3 XML-Syntax des Metadaten-Basistyps MCRMetaClassification

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

12siehe voriges Kapitel

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

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

Abbildung 10: XML-Syntax des Metadaten-Basistyps MCRMetaClassification

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

Abbildung 11: Beispiel des Metadaten-Basistyps MCRMetaClassification

4.2.2.4.4 XML-Syntax des Metadatentyps MCRMetaCorporation

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

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

Abbildung 12: XML-Syntax des Metadaten-Basistyps MCRMetaCorporation

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

Abbildung 13: Beispiel des Metadaten-Basistyps MCRMetaCorporation

4.2.2.4.5 XML-Syntax des Metadaten-Basistyps MCRMetaDate

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

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

Abbildung 14: XML-Syntax des Metadaten-Basistyps MCRMetaDate

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

Abbildung 15: Beispiel des Metadaten-Basistyps MCRMetaDate

4.2.2.4.6 XML-Syntax des Metadatentyps MCRMetaLangText

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

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

Abbildung 16: XML-Syntax des Metadaten-Basistyps MCRMetaLangText

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

Abbildung 17: Beispiel des Metadaten-Basistyps MCRMetaLangText

4.2.2.4.7 XML-Syntax der Metadatentypen MCRMetaLink und MCRMetaLinkID

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

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

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

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

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

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

Abbildung 18: XML-Syntax des Metadaten-Basistyps MCRMetaLink

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

Abbildung 19: Beispiel des Metadaten-Basistyps MCRMetaLink

4.2.2.4.8 XML-Syntax des Metadatentyps MCRMetaNumber

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

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

Abbildung 20: XML-Syntax des Metadaten-Basistyps MCRMetaNumber

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

Abbildung 21: Beispiel des Metadaten-Basistyps MCRMetaNumber

4.2.2.4.9 XML-Syntax des Metadatentyps MCRMetaPerson

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

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

Abbildung 22: XML-Syntax des Metadaten-Basistyps MCRMetaPerson

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

Abbildung 23: Beispiel des Metadaten-Basistyps MCRMetaPerson

4.2.2.4.10 XML-Syntax des Metadatentyps MCRMetaXML

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

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

Abbildung 24: XML-Syntax des Metadaten-Basistyps MCRMetaXML

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

Abbildung 25: Beispiel des Metadaten-Basistyps MCRMetaXML

4.2.2.5 XML-Syntax des XML-Knotens service

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

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

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

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

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

Abbildung 26: XML-Syntax des service XML-Konotens

4.2.3 Das Speichermodell für die Multimediadaten (IFS)

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

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

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

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

Abbildung 27: XML-Syntax des Derivate-Datenmodells

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

4.3 Anpassungen des MyCoRe-Samples

4.3.1 Grundlegendes

Um einen Dokument-Server oder andere MyCoRe-Anwendungen problemlos als eigenständige Anwendung zu betreiben, ist es notwendig hierfür auch separate Speicherbereiche für die Daten und Objekte bereitzustellen. Hierzu muss die Konfiguration an verschiedenen Stellen geändert werden. Oft reicht schon die Änderung der Tabellennamen aus. Für diese Beispiel wird der Prefix **DOC** verwendet.

Im nachfolgenden Abschnitt wird davon ausgegangen, dass Sie den MyCoRe-Sample Baum in ein gesondertes Anwendungsverzeichnis kopiert haben, z. B. /docserv welches durch die Environment Variable z. B.

```
export DOCSERV_HOME=/docserv
```

beschrieben wird.

An dieser Stelle sollen Sie auch noch einmal darauf hingewiesen werden, dass es auch unveränderliche Einstellungen gibt, welche Sie beim compilieren des MyCoRe-Kernes einmal festgelegt haben. Die betrifft vor allem die benutzte SQL-Datenbank (DB2, MySQL, ...) und den verwendeten Metadaten-Store (CM8, XML:DB,...). Diesen benutzen alle Ihre Applikationen gemeinsam!

Kopieren Sie nun wieder die private Konfigurationsdatei und bearbeiten Sie diese nach folgenden Gesichtspunkten:

```
cd ~/mycore-sample-application/config
cp mycore.properties.priv_template mycore.properties.private
```

- Wenn Sie den Content Manager 8.2 einsetzen überprüfen Sie den Namen des Library-Servers und Tragen Sie das Zugangspasswort ein. Ändern Sie die Namen der ItemTypes für die Objekte und das IFS und deren Prefixe (zwei andere, noch nicht verwendete Zeichen). Weiterhin sollten Sie andere Index-Verzeichnisse für die Textsuche angeben. Siehe Konfiguration von IBM Content Manager 8.2.
- Weinn Sie mit eXist arbeiten wollen, ändern sie die Collection von mycore z. B. auf docserv. Diese Collection müssen Sie nun mittels eXist-Client analog zum MyCore-Sample anlegen. Siehe Die Nutzung von eXist als XML:DB Backend.
- Die Einstellungen für die JDBC-Treiber können Sie aus Ihrer beispiel-Konfiguration übernehmen, hier sind keine Anpassungen erforderlich, da Sie andere Tabellennamen verwenden.
- Nun folgt der wohl umfangreichste Bereich, in denen Sie Änderungen vornehmen sollten. Es ist sehr zu empfehlen, für jedes Projekt eigene Tabellen zu verwenden. Hierzu müssen z. B. in den Tabellennamen **MCR** durch eine beliebige andere Zeichenfolge ersetzt werden. Für den Dokumenten-Server bietet sich hier DOC an, also MCRXMLDER --> DOCXMLDER usw.
- Im Bereich des User Management sollten Sie die User- und Group-Namen sowie die angegebnen Passworte des Beispiels durch eigene Werte ersetzen.

- Der nächste Abschnitt gibt den Remote-Zugriff auf Ihr System an. Ändern Sie die Daten wie Host, Port und Servlet-Pathes auf die von Ihnen belegten Werte ab.
- Für die IFS-Stores in das lokale File-System und nach Lucene sind eigene Verzeichnisse anzugeben und **auch anzulegen**!
- Um mehr von den inneren Abläufen des MyCore-Projektes zu verstehen, kann es hilfreich sein, temporär den DEBUG-Mode im Logging-System einzuschalten.
- Auch die Durchsicht der anderen Konfigurationsdateien verbessert das Verständnis von MyCore.

4.3.2 Füllen des eigenen Dokumenten-Servers

Nachdem Sie alle Konfigurationen eingetragen und alle zugehörigen Verzeichnisse usw. erzeugt haben, können Sie auf Basis des Commandline-Tools nun erste Daten einlesen. Modifizieren Sie dazu ggf. die mitgelieferten Unix-Tools.

Klassifikationen

Erstellen Sie sich entsprechend der Hinweise unter Das Klassifikationen-Datenmodell eigene Klassifikationen und/oder modifizieren Sie die im MyCoRe-Projekt mitgelieferten.

4.4 Gestaltung der Web-Server

4.4.1 Nutzung von Tomcat und Apache2

4.4.1.1 Allgemeines

Ziel dieses Szenarios ist, dass unter mcradmin (oder einem anderen User) mehrere Applikationen mit verschiedenen URL Aliasen parallel laufen und sich dabei nicht ins Gehege kommen. Konkret bedeutet dies, dass verschiedene Tomcat Instanzen, die über einen Connector in den Apache Server integriert sind, parallel nebeneinander laufen. Zudem können die einzelenen Applikationen über eigene URLs angesprochen werden was mit Apache über sogenannte virtual hosts realisiert wird. Das folgende Beispiel ist etwas SuSE spezifisch sollte sich jedoch ohne weiteres auch auf eine andere Distribution übertragen lassen . Eine sehr gute Anleitung findet sich jedoch auch in der Dokumentation von Tomcat.

4.4.1.2 Installation von Tomcat

Es wurden folgende Versionen von Tomcat unter SuSE installiert:

- **jakarta-tomcat-4.1.27-37**
- **apache2-jakarta-tomcat-connectors-4.1.27-37**

Die Binaries können von der Tomcat-Hompage heruntergeladen werden.

4.4.1.3 Arbeiten mit mehreren Tomcat-Instanzen

Für das Aktivieren mehrerer Tomcat-Instanzen sind folgende Environment-Varibalen notwendig,

$CATALINA_BASE und **$CATALINA_HOME.** Dabei verweist **$CATALINA_HOME** auf das installierte Tomat-Verzeichnis, also Bsp. */opt/jakarta/tomcat*.

Die Varibale **$CATALINA_BASE** wird auf das Verzeichnis der zu installierenden Tomcat-Instanz gesetzt, Bsp. */work/mcradmin/papyri-tomcat* . In dem angelegten Verzeichnis der Tomcat-Instanz müssen dann noch folgende Verzeichnisse angelegt werden, *work*, *conf*, *temp*, *webapps*, *logs* und *bin*.

Das Verzeichnis *conf* wird von */opt/jakarta/tomcat/conf* übernommen.

```
cp /opt/jakarta/tomcat/conf/*  $CATALINA_BASE/conf.
```

Im Verzeichnis **$CATALINA_BASE/conf** ist dann das File *server.xml* zu bearbeiten. Es müssen in dieser Datei die entsprechenden Ports eingetragen werden. Man sollte 5 aufeinanderfolgende Ports verwenden (Bsp. 8190-8194). Die Ports müssen in der */etc/services* noch nicht belegt sein. Im File *server.xml* sind folgende Zeilen anzupassen[14]:

```
<Server port="8190" shutdown="SHUTDOWN" debug="0">
  <!-- Define the Tomcat Stand-Alone Service -->
  <Service name="DocServ-Standalone">
    <!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
    <Connector
     className="org.apache.coyote.tomcat4.CoyoteConnector"
               port="8191" minProcessors="5" maxProcessors="75"
               enableLookups="true" redirectPort="8192"
               acceptCount="100" debug="0"
               connectionTimeout="20000"
               useURIValidationHack="false"
               disableUploadTimeout="true" />
  </Service>
```

14Rot markierter Text ist entsprechend anzupassen.

```
  <Service name="DocServ-Instance">
    <!-- Define a Coyote/JK2 AJP 1.3 Connector on port 8019 -->
    <Connector
      className="org.apache.coyote.tomcat4.CoyoteConnector"
       port="8194" minProcessors="5" maxProcessors="75"
       enableLookups="false"
       acceptCount="10" debug="0" connectionTimeout="0"
       useURIValidationHack="false"
       protocolHandlerClassName="org.apache.jk.server.JkCoyoteHandler"/>
    <Engine name="Urzlib" defaultHost="urzlib.dl.uni-leipzig.de" debug="0"
jvmRoute="tomcat_02">
      <Logger className="org.apache.catalina.logger.FileLogger"
        prefix="apache_log." suffix=".txt" timestamp="true"/>
      <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
        debug="0" resourceName="UserDatabase"/>
      <!-- Access log processes all requests for this virtual host. -->
      <Valve className="org.apache.catalina.valves.AccessLogValve"
        directory="logs"  prefix="urzlib_access_log."
        suffix=".txt" pattern="common" resolveHosts="false"/>
      <Host name="urzlib.dl.uni-leipzig.de" debug="10"
        appBase="/work/mcradmin/urzlib-tomcat/webapps"
        <!-- unpackWARs="false" autoDeploy="true" -->
        >
        <Alias>urzlib</Alias>
        <Context path="/urzlib" docBase="ROOT" debug="10" />
        <Valve className="org.apache.catalina.valves.AccessLogValve"
          directory="logs"  prefix="home_access_log." suffix=".txt"
          pattern="common" resolveHosts="false"/>
        <Context path="/manager" debug="0" privileged="true"
          docBase="/opt/jakarta/tomcat/server/webapps/manager/" />
        <Context path="/admin" debug="0" privileged="true"
          docBase="/opt/jakarta/tomcat/server/webapps/admin/" />
      </Host>
    </Engine>
  </Service>
</Server>
```

Abbildung 28: Das Tomcat Konfigurationsfile server.xml

Unter $**CATALINA_BASE/webapps** ist noch ein Link zu setzen. Dieser zeigt auf das Verzeichnis, in welchem die Applikation liegt.

```
ln -s /home/mcradmin/mycore-sample-tomcat/webapps ROOT
```

Nach dem bearbeiten des Files server.xml ist das Startscript für diese Tomcat-Instanz unter

$**CATALINA_BASE/**bin zu erstellen. Das Startscript, Bsp. urzlib.sh hat folgenden Inhalt[15]:

```
#!/bin/bash
# set the environment
CATALINA_BASE=/home/mcradmin/docserv-tomcat
export CATALINA_BASE
if [ $1 = 'start' ]
  then
  rm -r $CATALINA_BASE/logs/*
  fi
exec /opt/jakarta/tomcat/bin/catalina.sh "$@"
```

Abbildung 29: Das Listing des Start-Scriptes urzlib.sh

Nun kann diese Tomcat-Instanz gestartet werden.

4.4.1.4 Apache2 Installation

Wir haben unter SuSE den Apachen2 der Version apache2-2.0.48-9 installiert.

Es ist das File **workers2.properties** unter /etc/apache2 anzulegen, damit eine Verbindung vom Apachen zur Tomcat-Instanz hergestellt werden kann. Dieses File hat in unserem Beispiel folgenden Inhalt und ist entsprechend zu bearbeiten.

```
logger]
level=DEBUG
# Alternate file logger
[logger.file:0]
level=DEBUG
file=/var/log/jk2.log

[config:]
file=/etc/apache2/workers2.properties
debug=10
debugEnv=10

[uriMap:]
info=Maps the requests. Options: debug
debug=1
```

15Rot markierter Text ist entsprechend anzupassen.

```
[shm:]
info=Scoreboard. Required for reconfiguration and status with multiprocess
server
file=/var/log/jk2.shm
size=1000000
debug=1
disabled=1

[workerEnv:]
info=Global server options
timing=1
debug=1
logger=logger.file:0

[lb:lb]
#info=Default load balancer.
debug=1
[channel.socket:docserv.mydoamin.de:8194]
info=Ajp13 forwarding over socket
debug=0
tomcatId=docserv.mydomain.de:8194
disabled=1
keepalive=On
timeout=10
group=lb1

[channel.apr:docserv.mydomain.de:8194]
info=Ajp13 forwarding over socket
debug=0
tomcatId=docserv.mydomain.de:8194
disabled=0
keepalive=On
timeout=20
group=lb1

[status:status]
disabled=1
## Define mappings
[uri:docserv.mydomain.de/*.jsp]
#ver=0
disabled=2
#group=lb
group=ajp13:docserv.mydomain.de:8194
```

```
[uri:docserv.mydomain.de/servlets/*]
#ver=0
disabled=2
#group=lb
group=ajp13:docserv.mydomain.de:8194

[uri:/jkstatus/*]
#disabled=0
info=Display status information and checks the config file for changes.
group=status:status
```

Abbildung 30: Die Datei workers2.properties

4.4.1.5 Einrichten virtueller Hosts

Das Einrichten virtueller Hosts erfolgt durch das Anlegen des Files *httpd.conf.local* unter dem Verzeichnis /etc/apache2.

In unserem Beispiel hat dieses File dann folgenden Inhalt[16]:

```
LoadModule jk2_module /usr/lib/apache2/mod_jk2.so

<IfModule mod_jk2.c>
  JkSet config:file /etc/apache2/workers2.properties
</IfModule>

<Directory "/*/WEB-INF/*">
        AllowOverride None
        Deny from all
    </Directory>

NameVirtualHost 139.18.108.9:80
```

16Rot markierter Text ist wieder entsprechend Ihrer Anwendung anzupassen.

```
<VirtualHost docserv.mydomain.de:80>
#<VirtualHost 139.18.108.9>
#<VirtualHost *>
     ServerAdmin mcradmin@docserv.mydomain.de
     DocumentRoot /work/mcradmin/docserv-tomcat/webapps/ROOT
     ServerName docserv.mydomain.de
     ServerAlias docserv
     ErrorLog /var/log/apache2/docserv-errors
     TransferLog /var/log/apache2/docserv-access
     ServerPath /work/mcradmin/docserv-tomcat/webapps/ROOT
    Alias /urzlib "/work/mcradmin/docserv-tomcat/webapps/ROOT"

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

Abbildung 31: Die Datei http.local.conf

Dem Apachen muss nun noch mitgeteilt werden, dass es virtuelle Hosts gibt und das dieses File verarbeitet werden soll. Dies geschieht durch das Hinzufügen folgender Zeile in dem Konfigurationsfile **/etc/sysconfig/apache2**

```
# This allows you to add e.g. VirtualHost statements without
# touching
# /etc/httpd/httpd.conf itself, which makes upgrading easier.
#
APACHE_CONF_INCLUDE_FILES="/etc/apache2/httpd.conf.local"
```

Abbildung 32: Ein Ausschnitt aus der Datei /etc/sysconfig/apache2

Wenn nun alles richtig eingestellt ist, kann der Apache2 durch rcapache2 start aktiviert werden.

4.4.2 Arbeiten mit IBM Websphere

Im Kapitel 3 wurde bereits der Einsatz von IBM WebSphere in einer simplen Variante besprochen. Nun soll die Anwendung jedoch unter einer virtuellen Host-Adresse erreichbar sein. Dazu sind

folgende Schritte erforderlich:

- Beschaffen Sie sich einen Alias für Ihren Host z. B. docserv.mydomain.de
- Starten Sie den WebSphere-Admin-Client mit
  `/usr/WebSphere/AppServer/bin/startServer.sh server1`
- Tragen Sie den Virtuell Host ein und vergeben Sie dem Eintag einen Namen z. B. dochost
  - Environment – Virtual Hosts
  - New
  - Host docserv.mydomain.de mit Alias dochost eintargen
- Installieren Sie die Web-Anwendung in folgenden Schritten
  - ServerPath : .../docserv/docserv.war
  - ContextRoot: /
  - Next
  - Default virtual Host: dochost
  - Next
  - Next
  - DocServApplication: dochost
  - Next
  - Verbindung mit Server herstellen – dochost auswählen – Apply
  - Next
  - Finish
- Nun muss noch ein Update von WebSphere erfolgen
  - Environment – Update Web Server Plugin
  - OK
- Stoppen Sie den WebSphere-Admin-Client mit
  `/usr/WebSphere/AppServer/bin/stopServer.sh server1`
- Starten Sie nun Ihre Anwendung mit
  `/usr/WebSphere/AppServer/bin/startServer.sh docserv`

4.5 Ausbau des Dokumenten-Servers mit zusätzlichen MyCoRe-Komponenten

4.5.1 Anbindung anderer Dokumenten-Server an das System

MyCoRe bietet die Möglichkeit, mehrere Server-Instanzen mit gleichen Datenmodellen zu einem gemeinsamen virtuellen Server zusammenzuschalten. Dies ist relativ leicht zu konfigurieren.

Wichtig ist dabei jedoch, dass die Datenmodelle in ihren Strukturen und Tag-Bezeichnern identisch sind, da sonst die Suche nach Dokumenten oder Objekten erfolglos verläuft.

Die Anzahl der zusammengeschalteten Server ist theoretisch nicht begrenzt, in der Praxis sind mehr als 5-10 bestimmt wenig sinnvoll. Das System ist so gestaltet, dass beim Ausfall einzelner Instanzen die Arbeit mit den anderen nicht beeinträchtigt wird. Es fehlen lediglich die Ergebnisse des ausgefallenen Servers. Bevor Sie mit der Konfiguration beginnen, sollten Sie den Abschnitt 'Die Zusammenarbeit mit anderen MyCoRe-Sample-Installationen' lesen.

Um weitere Systeme zu ergänzen oder deren Einträge zu änderen, gehen Sie wie folgend vor:
Im Konfigurations-File *mycore.properties.private* unter $DOCSERV_HOME/config ist die Zeile mit **MCR.remoteaccess_hostaliases** anzupassen und im File *mycore.properties.remote* ist der Zugriff auf die neue Instanz zu definieren.

```
# List of the remote server aliases
MCR.remoteaccess_hostaliases=remote,NewRemoteSystem

# Configuration for the new host with remote access
MCR.remoteaccess_NewRemoteSystem_query_class=org.mycore.backend.remote.MCRServ
letCommunication
MCR.remoteaccess_NewRemoteSystem_host=the.new.server.name
MCR.remoteaccess_NewRemoteSystem_protocol=http
MCR.remoteaccess_NewRemoteSystem_port=80
MCR.remoteaccess_NewRemoteSystem_query_servlet=/mycoresample/servlets/MCRQuery
Servlet
MCR.remoteaccess_NewRemoteSystem_ifs_servlet=/mycoresample/servlets/MCRFileNod
eServlet
```

Abbildung 33: Konfigurationseintrag für zusätzlich abzufragende Instanzen

Ergänzen Sie nun noch die Suchmasken unter $DOCSERV_HOME/config um die weiteren Search-Instanzen und Testen Sie Ihre Konfiguration. **Achtung, wenn Sie diesen Dienst anderen anbieten, muss Ihr Web-Server aktiv sein!**

4.5.2 Einbindung weiterer Content Stores

4.5.3 Nutzung der OAI Schnittstelle

4.5.3.1 Grundlagen

Die Open Archives Initiative[17] hat 2001 ein offenes Protokoll für das Sammeln (Harvesting) von Metadaten vorgestellt. Dies geschah vor dem Hintergrund, dass gängige Suchmaschinen im WWW für die wissenschaftliche Nutzung wegen der i.d.R. unüberschaubaren Treffermenge und der fehlenden Qualität der angebotenen Treffer kaum nutzbar sind. Das **Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)** liegt mittlerweile in der Version 2.0 vor.

Das OAI-PMH dient zur Kommunikation zwischen **Data Providern** und **Service Providern**. Unter einem Data Provider versteht man hierbei ein Archivierungssystem, dessen Metadaten von einem

17 http://www.openarchives.org/

(oder mehreren) Service Provider(n) abgerufen werden, der diese als Basis zur Bildung von Mehrwertdiensten benutzt (z.B. der Suche über viele Archive gleichzeitig).

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

- Ein **Harvester** ist ein Client, der OAI-PMH Anfragen stellt. Ein Harvester wird von einem Service Provider betrieben, um Metadaten aus Repositories zu sammeln.
- Ein **Repository** ist ein über das Netzwerk zugänglicher Server, der OAI-PMH Anfragen verarbeiten kann, wie sie im Open Archives Initiative Protocol for Metadata Harvesting 2.0 vom 2002-06-14 beschrieben werden[18]. Ein Repository wird von einem Data Provider betrieben, um Harvestern den Zugriff auf Metadaten zu ermöglichen.

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

4.5.3.2 Der OAI Data Provider

MyCoRe bietet ein extrem flexibles Klassifikations-/Kategoriensystem. Ein OAI-Repository kann hiervon nur eine Klassifikation zur Strukturierung der Metadaten abbilden, d.h. einer MyCoRe-Klassifikation wird zu einem OAI-Repository. Es werden nur Metadaten zu Archivaten an den Harvester ausgeliefert, die in genau dieser MyCoRe-Klassifikation erfasst sind. Zur weiteren Einschränkung kann eine weitere Klassifikation angegeben werden, die für den OAI Data Provider aber nicht strukturbildend ist.

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

4.5.3.3 Installation

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

4.5.3.4 Der Deployment Descriptor

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

18 http://www.openarchives.org/OAI/openarchivesprotocol.html

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

Abbildung 34: Einbindung des OAI-Data-Providers in web.xml

4.5.3.5 Die mycore.properties

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

4.5.3.6 Instanzunabhängige Properties

- oaiadminemail=admin@uni-irgendwo.de **(notwendig)** Der Administrator der OAI-Repositories.
- oairesumptiontoken.dir=/mycore/temp **(notwendig)** Ein Verzeichnis, in welches der OAI Data Provider Informationen über Resumption Token ablegt.
- oairesumptiontoken.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.
- oaimaxreturns=50 **(notwendig)** Die maximale Länge von Ergebnislisten, die an einen Harvester zurückgegeben werden. Überschreitet eine Ergebnisliste diesen Wert, so wird ein Resumption Token angelegt.
- oaiqueryservice=org.mycore.services.oai.MCROAIQueryService **(notwendig)** Die Klasse, die für das Archiv das Query-Interface implementiert. Für Miless ist dies oaiOAIService .
- oaimetadata.transformer.oai\s\do6(d)c=MyCoReOAI-mycore2dc.xsl **(notwendig)** Das Stylesheet, das die Transformation aus dem im Archiv benutzten Metadatenschema in das für OAI benutzte OAI Dublin Core Metadatenschema durchführt. Wenn sich das im Archiv benutzte Metadatenschema ändert, muss dieses Stylesheet angepasst werden. Optional können weitere Stylesheets angegeben werden, die einen Harvester mit anderen Metadatenformaten versorgen, z.B. oaimetadata.transformer.rfc1806=MyCoReOAI-mycore2rfc.xsl. Diese Stylesheets benutzen als Eingabe das Ergebnis des ersten Stylesheets.

4.5.3.7 Instanzabhängige Properties

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

- `oai`repositoryname.OAIDataProvider=Test-Repository **(notwendig)** Der Name des OAI-Repositories.
- `oai`repositoryidentifier.OAIDataProvider=mycore.de **(notwendig)** Der Identifier des OAI-Repositories (wird vom Harvester abgefragt).
- `oai`setscheme.OAIDataProvider=MyCoReDemoDC\s\do6(c)lass\s\do6(1) **(notwendig)** Die MyCoRe-Klassifikation, die zur Bildung der Struktur des OAI-Repositories verwendet wird.
- `oai`restriction.classification.OAIDataProvider=MyCoReDemoDC\s\do6(c)lass\s\do6(2) **(optional)** Die MyCoRe-Klassifikation, die zur Beschränkung der Suche verwendet wird.
- `oai`restriction.category.OAIDataProvider=dk01 **(optional)** Die MyCoRe-Kategorie, die zur Beschränkung der Suche verwendet wird.
- `oai`friends.OAIDataProvider=miami.uni-muenster.de/servlets/OAIDataProvider **(optional)** Unter dieser Property können weitere (bekannte und zertifizierte) OAI-Repositories angegeben werden, um den Harvestern die Suche nach weiteren Datenquellen zu vereinfachen.

4.5.3.8 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.

4.5.3.9 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.

5.Hints & Tips / Troubleshooting