Programmer Guide

Release 2.0

26. Februar 2007

Frank Lützenkirchen (Duisburg-Essen)

Jens Kupferschmidt (Leipzig)

Detlev Degenhardt (Freiburg)

Heiko Helmbrecht (München)

Thomas Scheffler (Jena)

Anja Schaar (Rostock)

Harald Richter (Duisburg-Essen)

Kathleen Krebs (Hamburg)

Vorwort

Dieser Teil der Dokumentation ist vor allem für Applikationsprogrammierer gedacht. Er beschreibt die Designkriterien und ihre Umsetzung in der vorliegenden Version 2.0. Mit Hilfe dieser Dokumentation sollte es Ihnen möglich sein, Details von MyCoRe zu verstehen und eigene Anwendungen konzipieren und implementieren zu können. Dieses Dokument wird stetig erweitert[1]. Es sei an dieser Stelle auch auf die Handbücher **User Guide** und **Developer Guide** des Projektes verwiesen.

[1]Die aktuellste Version ist unter www.mycore.de/content/main/documentation zu finden.

Inhaltsverzeichnis

1 Allgemeines zur Implementierung

1.1 Struktur der Gesamtanwendung

MyCoRe ist entgegen anderen Projekten kein monolitisches Objekt. Vielmehr wurde ein modularer Ansatz gewählt, um die Vielfalt der Einsatzgebiete sinnvoll abzudecken. Dabei wird strickt zwischen allgemein gültigen Kernaufgaben und Funktionalitäten der schlussendlichen Anwendung unterschieden. Innerhalb dieser Bereiche gibt es wieder Teile, welche unabdingbar erforderlich sind und solche, die im Bedarfsfall in die Endanwendung integriert werden können. Die Einbeziehung der Module geschieht über zusätzliche Teile, welche in die Zielanwendung zu übernehmen sind. Die nachfolgende Skizze soll das verdeutlichen.

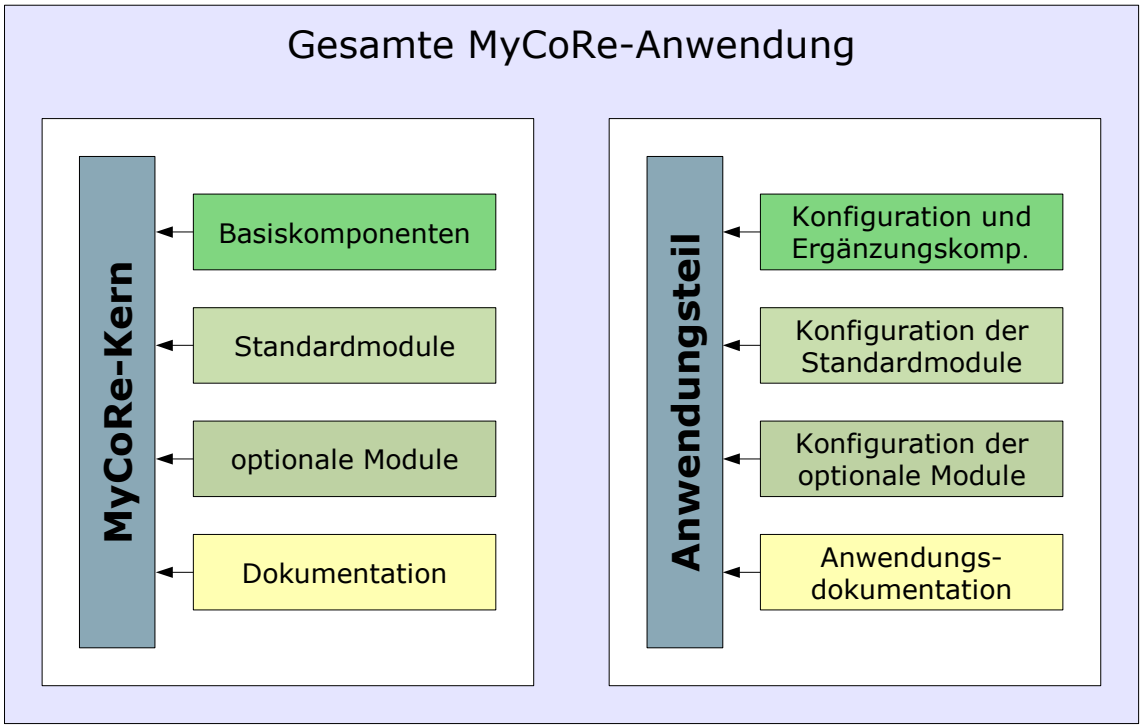


Abbildung 1.1: Grundstruktur des MyCoRe-Projektes

Das Kernsystem des MyCoRe-Projektes (Dateibaum /`mycore`) enthält die allgemein gültigen Dateien, die für die Gestaltung der eigenen Anwendung erforderlich sind. Dazu gehören neben den Java-Quellen auch Stylesheets, Schema-Dateien, Konfigurationen und benutzte Java-Archive von Drittanbietern sowie Dateien für den Build-Prozess. Weiterhin sind im Kern eine Reihe von Schriften enthalten, die den Umgang mit einer MyCoRe-Anwendung erklären sollen.

Ein Teil von Komponenten ist nicht für jede Art von MyCoRe-Anwendung notwendig. Dieser wurden in Module zusammengefasst und ist als solche im Verzeichnis `mycore/modules` abgelegt. Standardmodule werden von Hause aus beim Build-Prozess mit integriert. Wenn dies nicht gewünscht wird, so müssen diese Module explizit ausgeschlossen werden. Umgekehrt verhält es sich bei den optionalen Modulen (`UNINSTALLED_modulname`), welche ausdrücklich der Anwendung hinzugefügt werden müssen. Grund sind hier, wie z.B. beim `ImageViewer`,

Lizenzrechte der benutzten zusätzlichen Bibliotheken. Diese müssen vom Endanwender extern hinzugefügt werden und sind nicht im Distributionsumfang von MyCoRe enthalten. Die Integration ist im jeweiligen Modul enthalten. In den entsprechenden Verzeichnissen sind neben dem Quellcode auch Konfigurationsdateien und Dokumentation enthalten.

Die zweite Komponente einer MyCoRe-Anwendung ist die eigentliche Applikation. Sie benutzt die Bibliotheken und anderen Dateien des Kerns, ohne diese noch einmal gesondert anzupassen. Dateien, welche geändert Verwendung finden sollen, werden in der Anwendung gespeichert und überschreiben die Kerndaten. Dies gilt z. B. auch für Java-Klassen. Die Anwendung enthält weiterhin alle Konfigurationen und Ergänzungen, welche für die volle Funktion der Applikation erforderlich sind. Nach erfolgreichem Build-Prozess sind hier auch die Wurzelverzeichnisse der Anwendungsdaten platziert (es sei denn, die Konfiguration wurde geändert).

Zur Integration der Module sind auf der Seite des Anwendungsteils analoge Verzeichnisstrukturen unter `modules` erforderlich. Diese müssen aber nur eine Datei `build.xml` und alle für die spezielle Anwendung notwendigen Ergänzungen enthalten. Es wird auch hier zwischen optionalen und Standardmodulen unterschieden.

Um die Funktionsweise einer vollständigen MyCoRe-Applikation zu verdeutlichen, wird vom Projekt eine Beispielanwendung eines Dokumentenrepositories angeboten. Alle Dateien sind unterhalb des Verzeichnisses /`docportal` enthalten. Die Anwendungen kann auf die speziellen eigenen Erfordernisse angepasst oder für eine völlig neue Applikation adaptiert werden.

1.2 Benutzte externe Bibliotheken

Das MyCoRe-Projekt bemüht sich, bereits etablierte Techniken und Implementationen zu benutzen, um einerseits den eigenen Entwicklungsaufwand so gering wie möglich zu halten, andererseits dem Nachnutzer den Einstieg durch die Verwendung bekannter Komponenten zu erleichtern.

Der Folgende Abschnitt beschreibt alle eingesetzten Bibliotheken. Dabei ist die Lizenzpolitik der Hersteller sehr unterschiedlich. Gemeinsam ist allen jedoch, dass sie den Open Source Gedanken vertreten und für nichtkommerzielle Einsätze frei verfügbar sind. Die Nutzung von MyCoRe unter wirtschaftlichen Aspekten bedarf also einer gesonderten Betrachtung der Lizenzrechte durch den Anwender. Ob hier Rechte verletzt werden, ist im Einzelfall abzuklären.

Files	**Beschreibung**	**Lizenz**
lib/activation.jar	?	?
lib/ant-antlr-1.6.3.jar	ANother Tool for Language Recognition.	?
lib/ant-contrib.jar	?	?
lib/antlr-2.7.5H3.jar	ANother Tool for Language Recognition.	

Files	Beschreibung	Lizenz
lib/asm.jar	A Java bytecode manipulation framework.	ASM
lib/asm-attrs.jar	A Java bytecode manipulation framework.	ASM
lib/axis.jar	Apache Axis is an implementation of the SOAP for Web services support	AL 2.0
lib/axis-ant.jar	Apache Axis is an implementation of the SOAP for Web services support	AL 2.0
lib/BrowserLauncher2.jar	BrowserLauncher2, a continuation of the BrowserLauncher project, is a library that facilitates opening a browser from a Java application and directing the browser to a supplied url. In most cases the browser opened will be the user's default browser.	LGPL
lib/c3p0-0.9.0.jar	c3p0 is an easy-to-use library for making traditional JDBC drivers "enterprise-ready" by augmenting them with functionality defined by the jdbc3 spec and the optional extensions to jdbc2.	LGPL
lib/cglib-2.1.jar	Byte Code Generation Library is high level API to generate and transform JAVA byte code. It is used by AOP, testing, data access frameworks to generate dynamic proxy objects and intercept field access.	AL 2.0
lib/commons-collections-2.1.1.jar	Apache Jakarta Common - Extends or augments the Java Collections Framework.	AL 2.0
lib/commons-fileupload-1.0.jar	Apache Jakarta Common - File upload capability for your servlets and web applications.	AL 2.0
lib/commons-httpclient-2.0.jar	Apache Jakarta Common - Framework for working with the client-side of the HTTP protocol.	AL 2.0
lib/commons-lang-2.0.jar	Apache Jakarta Common - Provides extra functionality for classes in java.lang.	AL 2.0
lib/commons-logging.jar	Apache Jakarta Common - Wrapper around a variety of logging API implementations.	AL 2.0

Files	Beschreibung	Lizenz
lib/commons-net-1.4.1.jar	Apache Jakarta Common - Collection of network utilities and protocol implementations.	AL 2.0
lib/commons-vfs-1.0-dev.jar	Apache Jakarta Common - Virtual File System component for treating files, FTP, SMB, ZIP and such like as a single logical file system.	AL 2.0
lib/dom4j-1.6.jar	*dom4j* is an easy to use, open source library for working with XML, XPath and XSLT on the Java platform using the Java Collections Framework and with full support for DOM, SAX and JAXP.	W3C
lib/ehcache-1.1.jar	Ehcache is a widely used java distributed cache for general purpose caching, J2EE and light-weight containers.	AL 2.0
lib/fckEditor.zip	This HTML text editor brings to the web many of the powerful functionalities of desktop editors like MS Word. It's **lightweight** and doesn't require any kind of installation on the client computer.	LGPL
lib/ftp.jar	Java FTP client library.	LGPL
lib/hibernate3.jar	Hibernate is a powerful, high performance object/relational persistence and query service.	LGPL
lib/hsqldb_1_0_8_0_1.jar	Lightweight 100% Java SQL Database Engine.	HS
lib/icu4j_3-6.jar	ICU is a mature, widely used set of Java libraries for Unicode support, software internationalization and globalization (i18n/g11n).	IBM
lib/jaxen-1.1.jar	The Jaxen Java XPath Engine is an open source cross-API (DOM, JDOM, dom4j, and ElectricXML) XPath library for Java.	
lib/jaxrpc.jar	?	?
lib/jcifs-1.2.0.jar	JCIFS is an Open Source client library that implements the CIFS/SMB networking protocol in 100% Java.	LGPL
lib/jdom-1.0.jar	To provide a complete, Java-based solution for accessing, manipulating, and outputting XML data from Java code.	JDOM
lib/joda-time-1.2.jar	Joda-Time provides a quality	AL 2.0

Files	Beschreibung	Lizenz
	replacement for the Java *date* and *time* classes.	
lib/jsch-0.1.20.jar	JSch allows you to connect to an sshd server and use port forwarding, X11 forwarding, file transfer, etc., and you can integrate its functionality into your own Java programs.	BSD
lib/jta.jar	JTA specifies standard Java interfaces between a transaction manager and the parties involved in a distributed transaction system: the resource manager, the application server, and the transactional applications.	SUN
lib/jtidy.jar	We have two primary goals. First, to provide a home where all the patches and fixes that folks contribute can be collected and incorporated into the program. Second, a library form of Tidy has been created to make it easier to incorporate Tidy into other software.	W3C
lib/junit-3.8.1.jar	JUnit is a regression testing framework written by Erich Gamma and Kent Beck.	CPL
lib/log4j-1.2.13.jar	Inserting log statements into your code is a low-tech method for debugging it.	AL 2.0
lib/lucene-analyzers-1.9.1.jar	Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java.	AL 2.0
lib/lucene-core-1.9.1.jar	Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java.	AL 2.0
lib/mail.jar	The JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications.	SUN
lib/poi-2.5-final-20040302.jar	The POI project consists of APIs for manipulating various file formats based upon Microsoft's OLE 2 Compound Document format using pure Java.	AL 2.0
lib/serializer_2_7_0.jar	This is a part of Xalan.	AL 2.0
lib/tm-extractors-0.4.jar	tm-extractors wraps the Word extraction from POI in a nice API.	AL 2.0
lib/xalan_2_7_0.jar	The Apache Xalan Project is a	AL 2.0

Files	Beschreibung	Lizenz
	collaborative software development project dedicated to providing robust, full-featured, commercial-quality, and freely available XSLT support on a wide variety of platforms.	
lib/xercesImpl_2_7_1.jar	The Xerces Java Parser 1.4.4 supports the XML 1.0 recommendation and contains advanced parser functionality, such as support for the W3C's XML Schema recommendation version 1.0, DOM Level 2 version 1.0, and SAX Version 2, in addition to supporting the industry-standard DOM Level 1 and SAX version 1 APIs.	AL 2.0
lib/xml-apis.jar	API-access to XML.	AL 2.0
lib/xmldb-api-20021118.jar	API and connector to XMLDB's from eXist.	LGPL
lib/xmldb.jar	API and connector to XMLDB's from eXist.	LGPL

Tabelle 1.1: Übersicht der benutzten externen Bibliotheken

Lizenzen:

- **AL 2.0** - Apache License Version 2.0, January 2004 - http://www.apache.org/licenses/
- **ASM** - Copyright (c) 2000-2005 INRIA, France Telecom - http://asm.objectweb.org/license.html
- **BSD** - Copyright (c) 2002,2003,2004,2005,2006 Atsuhiko Yamanaka, Jcraft,Inc.
- CPL – Common Public License Version 1.0 - http://www.oopensource/licenses/cpl.php
- **HS** - Copyright (c) 1995-2000 by the Hypersonic SQL Group. - http://www.hsqldb.org/web/hsqlLicense.html
- **IBM** – Special license for Open Soucre of IBM like LGPL
- **JDOM** - Copyright (C) 2000-2002 Brett McLaughlin & Jason Hunter. - http://www.jdom.org
- **LGPL** - GNU Library or Lesser General Public License 2.1
- **SUN** – Sun.com Terms of Use - http://www.sun.com/termsofuse.html
- **W3C** - W3C® SOFTWARE NOTICE AND LICENSE Copyright © 1994-2001 World - http://www.w3.org/Consortium/Legal/copyright-software-19980720

1.3 Allgemeine Klassen / Exception-Modell / MCRCache

1.3.1 Allgemeine Klassen

Wie in jedem Projekt, so gibt es auch in MyCoRe allgemein verwendete Klassen. Diese sind im Paket `org.mycore.common` und den dort befindlichen Unterpaketen platziert. Hier finden Sie Klassen für

- die Behandlung des MyCoRe-internen EventHandler-Systems,
- die Behandlung und den Umgang mit XML-Daten,
- Ausnahmebehandlungen,
- Caches,
- MyCoRe-Sessions und
- weitere nützliche Funktionen.

In den JavaDocs können Sie sich leicht einen Überblick der implementierten Klassen und Methoden verschaffen.

1.3.2 Exception Modell

MyCoRe nutzt zwei verschiedene Exception-Modelle, die hier kurz erläutert werden sollen. Hinsichtlich der Funktionen und Methoden der Exceptions sollten Sie die JavaDocs des MyCoRe-Projektes konsultieren.

1. MCRException und alle davon abgeleiteten Klassen. Diese Exception-Klasse ist von der Java-Klasse RuntimeException abgeleitet und kann damit Ausnahmezustände nach außen tragen.
2. Die MCRCatchException ist hingegen von der Java-Klasse Exception abgeleitet und muss auch jeden Fall abgefangen werden.

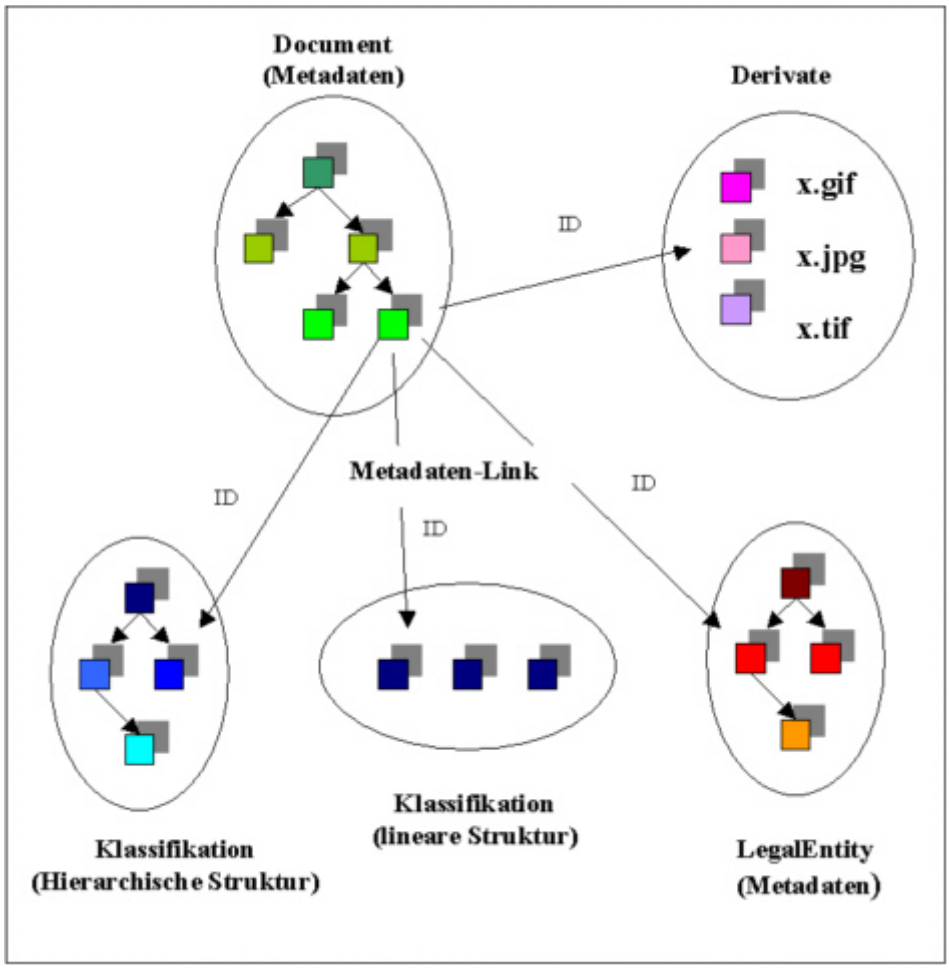
1.4 Das Datenmodell-Konzept allgemein

Das MyCoRe-Datenmodell kennt im seiner Grundkonzeption drei Komponeten. Zum einen gibt es Metadatensätze. Sie enthalten nur beschreibende Daten sowie ggf. strukturelle und organisatorische Informationen zu einem Datenobjekt. Dabei ist es nicht relevant, ob die Metadaten alleine stehen oder mit einem Anhang eines digitalen Objektes versehen werden sollen. Beispielsweise können Personendaten alleine existieren und die sie referenzierenden Dokumente besitzen noch zugeordnete digitale Objekte (Bilder, Dokumente, Videos usw.). Diese Objekte werden in MyCoRe als Derivate bezeichnet, da ein Objekt ggf. in mehreren Darstellungsvarianten an den selben Metadatensatz angebunden werden kann. Ein Derivat seinerseits besteht aus einem kurzen XML-Datensatz, welcher für den Im- und Export benötigt wird und den eigentlichen Daten in Form von Einzeldateien oder Dateibäumen.

Die dritte Art von Daten stellen Klassifikationen (in Bibliothekskreisen auch als Normdateien bezeichnet) dar. Sie definieren Sammlungen feststehender Begriffe auf die seitens der anderen Metadaten referenziert werden kann.

Die Verknüpfung der Metadaten untereinander und zu den Klassifikationen erfolgt mittels einer einheitlichen ID. Für Klassifikationen kommt hierzu noch die Kategorie-ID. Der Aufbau der ID's ist im User Guide und im Kapitel 3 näher beschrieben.

Das Datenmodell ist durch seine Gestaltung in XML-Strukturen so angelegt, dass es leicht möglich ist, Informationen in mehreren Sprachen gemeinsam abzulegen. Hierfür wird konsequent eine Codierung mit UTF-8 angewendet. bei zunehmender Globalisierung ist die Mehrsprachigkeit für viele Projekte zwingend erforderlich. Auch die Klassifikationen können mehrsprachig gespeichert werden.



1.5 Das Vererbungsmodell

In MyCoRe ist innerhalb des Datenmodells für die Metadaten die Möglichkeit einer Vererbung vorgesehen. Diese ist fest in den Kern implementiert und wird ausschließlich durch die steuernden Metadaten des jeweiligen Datensatzes festgelegt. Das heißt, für eine Datenmodell-Definition (z. B. document) können Datensätze mit (Buch mit Kapiteln) und ohne (Dokument) nebeneinander eingestellt werden. Wichtig ist nur, dass die Vererbung nur innerhalb eines Datenmodells oder

eines, welches die gleiche Struktur aufweist, jedoch andere Pflichtfelder hat, funktioniert. Vererbung zwischen Datenmodellen mit verschieden Metadaten ist ausgeschlossen.

Im Folgenden soll die Vererbung anhand der XML-Syntax zweier Metadaten-Objekte verdeutlicht werden. Beim Laden der Daten wird dann eine Eltern-Kind-Beziehung im System aufgebaut und abgespeichert.

Die beiden Datensätze (Abbildung 1.2 und 1.3) sollen folgendes Szenario widerspiegeln:

- Der Titel soll für die Kind-Daten übernommen werden und durch diese um die Kapitelüberschriften ergänzt werden.
- Die Autorendaten sind an alle Kinder zu vererben.
- Der Umfang des Werkes ist je nach Stufe anzugeben, also für das gesamte Buch die Gesamtzahl der Seiten und für ein Kapitel die Anzahl dessen Seiten.

```
1  <mycoreobject ... >
2      ...
3      <metadata xml:lang="de">
4      <titles class="MCRMetaLangText" heritable="true" notinherit="false"..>
5          <title xml:lang="de">Buchtitel</title>
6      </titles>
7      <authors class="MCRMetaLangText" heritable="true" notinherit="false"..>
8          <author xml:lang="de">Erwin der Angler</author>
9      </authors>
10     <sizes class="MCRMetaLangText" heritable="false" notinherit="false"..>
11         <size xml:lang="de">100 Seiten</size>
12     </sizes>
13     ...
14     </metadata>
15     ....
16 </mycoreobject>
```

Abbildung 1.2: Auszug aus dem Metadaten-Objektes des Elternsatzes

Entscheidend für die Umsetzung sind folgende Dinge:

- Das Attribut `heritable` sagt, ob ein Metadatum vererbbar (`true`) oder nicht (`false`) sein soll.
- Das Attribut `notinherit` sagt, ob das Metadatum von dem Elterndatensatz nicht geerbt werden soll (`true`). Andernfalls wird geerbt (`false`).
- Es muss erst der Elterndatensatz eingespielt werden. Anschließend können die Kinddatensätze der nächsten Vererbungsebene eingespielt werden. Enkeldatensätze folgen darauf, usw. Bitte beachten Sie das unbedingt beim Restore von Sicherungen in das System. MyCoRe ergänzt intern die Daten der Kind-Datensätze für die Suchanfragen um die geerbten Daten.
- Das Attribut `inherited` ist per Default auf `0` gesetzt und beschreibt die Anzahl der geerbten Daten. Das Attribut wird vom System automatisch gesetzt. Kinder erhalten, wenn sie Daten geerbt haben, `inherited=1` usw., je nach Stufe.

```
<mycoreobject ... >
    ...
    <metadata xml:lang="de">
        <titles class="MCRMetaLangText" heritable="true" notinherit="false"..>
            <title xml:lang="de">Kapitel 1</title>
        </titles>
        <sizes class="MCRMetaLangText" heritable="true" notinherit="true"..>
            <size xml:lang="de">14 Seiten</size>
        </sizes>
        ...
    </metadata>
    ....
</mycoreobject>
```

Abbildung 1.3: Auszug aus dem Metadaten-Objektes des Kindsatzes

Die Ausgabe des Eltern-Datensatzes nach einer Query entspricht dem der Dateneingabe, lediglich im XML-`structure`-Teil wurde ein Verweis auf die Kinddaten eingetragen[2].

```
<mycoreobject ... >
    ...
    <metadata xml:lang="de">
        <titles class="MCRMetaLangText" heritable="true" notinherit="false"..>
            <title inherited="1" xml:lang="de">Buchtitel</title>
            <title inherited="0" xml:lang="de">Kapitel 1</title>
        </titles>
        <authors class="MCRMetaLangText" heritable="true" notinherit="false"..>
            <author inherited="1" xml:lang="de">Erwin der Angler</author>
        </authors>
        <sizes class="MCRMetaLangText" heritable="false" notinherit="false"..>
            <size inherited="0" xml:lang="de">14 Seiten</size>
        </sizes>
        ...
    </metadata>
    ....
</mycoreobject>
```

Abbildung 1.4: XML-Syntax eines Kind-Datensatzes als Query-Resultat

Die Abbildung 1.4 zeigt den Kind-Datensatz, wie er vom System nach einer erfolgreichen Anfrage im Resultats-Container zurückgegeben wird. Dabei ist deutlich die Funktionalität der MyCoRe-Vererbungsmechanismen zu erkennen.

1.6 Die Session-Verwaltung

Mehrere verschiedene Benutzer und Benutzerinnen (oder allgemeiner Prinzipale) können gleichzeitig Sitzungen mit dem MyCoRe-Softwaresystem eröffnen. Während einer Sitzung werden in der Regel nicht nur eine, sondern mehrere Anfragen bearbeitet. Es ist daher sinnvoll, kontextspezifische Informationen wie die UserID, die gewünschte Sprache usw. für die Dauer der Sitzung mitzuführen. Da das MyCoRe-System mit mehreren gleichzeitigen Sitzungen konfrontiert werden kann,

[2]Siehe XML-Syntax im User Guide.

die zudem über verschiedene Zugangswege etabliert sein können (z.B. Servlets, Kommandozeilenschnittstelle oder Webservices), muss das System einen allgemein verwendbaren Kontextwechsel ermöglichen.

Bei der Bearbeitung einer Anfrage oder Transaktion muss nicht jede einzelne Methode oder Klasse Kenntnis über die Kontextinformationen besitzen. Daher ist es sinnvoll, die Übergabe des Kontextes als Parameter von Methode zu Methode bzw. von Klasse zu Klasse zu vermeiden. Eine Möglichkeit, dies zu bewerkstelligen ist der Einsatz von sog. Thread Singletons oder thread-local Variablen. Die Idee dabei ist, den Thread der den Request bearbeitet als Repräsentation des Request selbst anzusehen. Dazu müssen die Kontextinformationen allerdings an den Thread angebunden werden, was seit Java 1.2 mit Hilfe der Klassen `java.lang.ThreadLocal` bzw. `java.lang.InheritableThreadLocal` möglich ist. Jeder Thread hat dabei seine eigene unabhängig initialisierte Kopie der Variable. Die `set()` und `get()` Methoden der Klasse `ThreadLocal` setzen bzw. geben die Variable zurück, die zum gerade ausgeführten Thread gehört. Die Klassen der Sessionverwaltung von MyCoRe sind auf Basis dieser Technologie implementiert (siehe Abbildung 1.5).

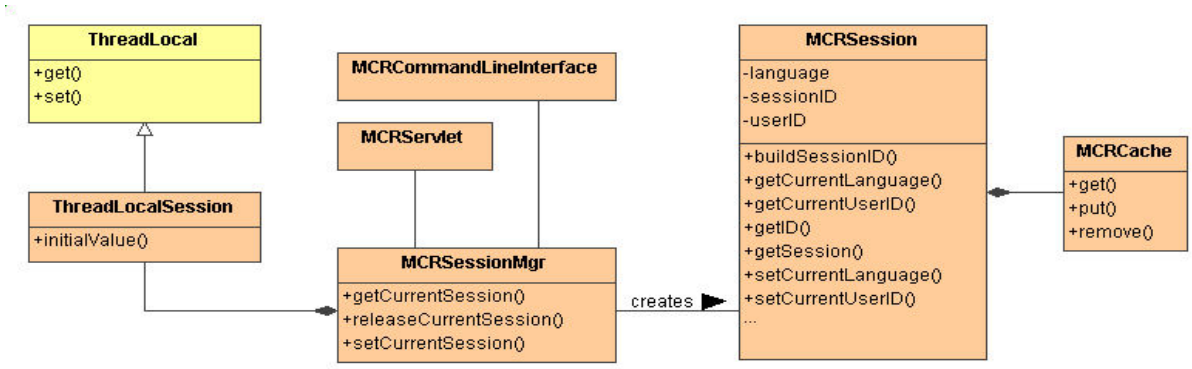


Abbildung 1.5: Die Klassen der Sessionverwaltung

Klienten der Sessionverwaltung sind alle Klassen, die Kontextinformationen lesen oder modifizieren wollen, wie zum Beispiel `MCRCommandLineInterface` und `MCRServlet`. Kontextinformationen werden als Instanzen der Klasse `MCRSession` abgelegt. Diese Klasse bietet Methoden zum Setzen und Lesen der Informationen, wie z. B. der UserID der aktuellen Benutzerin, der gewünschten Sprache usw.

Die Klasse `MCRSession` besitzt einen statischen Cache, realisiert durch die Klasse `MCRCache`. Bei der Konstruktion einer Instanz von `MCRSession` wird zunächst über die Methode `buildSessionID()` eine eindeutige Id erzeugt und diese als Schlüssel zusammen mit dem Session-Objekt selbst im Cache abgelegt. Auf diese Weise hat man über die statische Methode `getSession()` Zugriff auf die zu einer bestimmten SessionID gehörende Instanz.

Damit die Instanzen von `MCRSession` als thread-local Variablen an den aktuellen Thread angebunden werden können, werden sie nicht direkt, sondern über die statische Methode `getCurrentSession()` der Klasse `MCRSessionMgr` erzeugt und später gelesen. Beim ersten Aufruf von `getCurrentSession()` in einem Thread wird über die von `java.lang.ThreadLocal` erbende, statische innere Klasse

`ThreadLocalSession` gewährleistet, dass eine eindeutige Instanz von `MCRSession` erzeugt und als thread-local Variable abgelegt wird. Der Zugriff auf die thread-local Variablen eines Threads kann nur über die Klasse `ThreadLocal` (bzw. `InheritableThreadLocal`) erfolgen. Auf diese Weise ist sichergestellt, dass bei nachfolgenden Aufrufen von `getCurrentSession()` genau die zum aktuellen Thread gehörende Referenz auf die Instanz von `MCRSession` zurückgegeben wird.

Mit der statischen Methode `MCRSessionMgr.setCurrentSession()` ist es möglich, ein bereits vorhandenes Session-Objekt explizit als thread-local Variable an den aktuellen Thread zu binden. Dies ist z.B. In einer Servlet-Umgebung notwendig, wenn die Kontextinformationen in einem Session-Objekt über eine http-Session mitgeführt werden. (Aktuelle Servlet-Engines verwenden in der Regel zwar Thread-Pools für die Bearbeitung der Requests, aber es ist in keiner Weise sichergestellt, dass aufeinanderfolgende Requests mit dem selben Kontext wieder den selben Thread zugewiesen bekommen. Daher muss der Kontext in einer http-Session gespeichert werden.)

Die Methode `MCRSessionMgr.releaseCurrentSession()` sorgt dafür, dass das thread-local Session-Objekt eines Threads durch ein neues, leeres Objekt ersetzt wird. Dies ist in Thread-Pool-Umgebungen wichtig, weil es sonst möglich bzw. sogar wahrscheinlich ist, dass Kontextinformationen an einem Thread angebunden sind, dieser Thread aber bei seiner Wiederverwendung in einem ganz anderen Kontext arbeitet. Code-Beispiele zur Verwendung der Session-Verwaltung finden sich in `org.mycore.frontend.servlets.MCRServlet.doGetPost()`.

1.7 Das EventHandler-Modell

Mit Version 1.2 wurde in die MyCoRe-Implementierung ein EventHandler-Basispaket integriert. Ziel ist es, eine bessere Trennung der Code-Schichten des Datenmodells und der Backends zu erreichen. Im Datenmodell sollen nur noch Ereignisse ausgelöst werden (z. B. create, delete usw.), welche dann bestimmt durch die Konfiguration in den Property-Dateien verarbeitet werden. Es soll ein allgemein gültiges Template-Modell existieren, welches für die erforderlichen Anwendungsfälle ausgebaut werden kann. Ein singleton-Manager-Prozess nimmt nur ein Ereignis entgegen, wählt die dafür bestimmte Konfiguration aus und startet die Methode `doHandleEvent(MCREvent evt)`. Dies geschieht in der Reihenfolge, welche in der Konfiguration angegeben ist und stellt ein Pipeline-Verfahren dar. Das Event-Objekt wird dabei nacheinander an die Handler durchgereicht. Änderungen an den im Event-Objekt gespeicherten Daten werden also für alle folgenden Handler wirksam. Kommt es bei einem Handler zu einer Ausnahme, so wird diese vom Manager aufgefangen und es wird für alle in der Pipeline davor liegenden Handler die Methode `undoHandleEvent(MCREvent evt)` initiiert. Somit ist ein Rollback möglich. Je nach Anwendung ist es möglich, verschiedene Pipelines für unterschiedliche Abläufe unabhängig voneinander zu implementieren, z. B. eine Pipeline für die Verarbeitung der Metadaten und eine andere für die Volltextindizierung der Dokumente. Die Pipelines und die damit verbundenen Ereignisse unterscheiden sich am Namen der jeweiligen Pipeline.

1.7.1 Das EventHandler-Modell am Beispiel der Metadaten-Objekte

Das EventHandler-Modell wird beispielsweise eingesetzt, um Objekte vom Typ MCRObject persistent zu speichern. Das Klassendiagramm (Abbildung 1.6) soll die Zusammenhänge verdeutlichen.

1. Das `MCRObject` ruft in MyCoRe-Version 1.2 zuerst eine Persistence-Layer-Implementierung nach alter Konzeption auf. Hier wurde zur Nutzung des EventHandlers eine Dummy-Klasse `MCRDummySearchStore` geschaffen, welche keine Funktionalität ausführt.
2. Anschließend wird von `MCRObject` ein neues Ereignis erzeugt, welches in diesem Fall die vordefinierte Pipeline `OBJECT_TYPE` und das vordefinierte Ereignis `CREATE_EVENT` nutzt. Es können aber auch beliebige Strings eingetragen werden. Dabei ist aber auf die Konsistenz zu achten.
   ```
   MCREvent evt = new MCREvent(MCREvent.OBJECT_TYPE,
   MCREvent.CREATE_EVENT);
   ```
3. Nun wird dem neuen Ereignis das Datum übergeben, welches an die Handler weitergereicht werden soll. Ein Ereignis kann auch mehrere Daten beinhalten.
   evt.put("object",this);

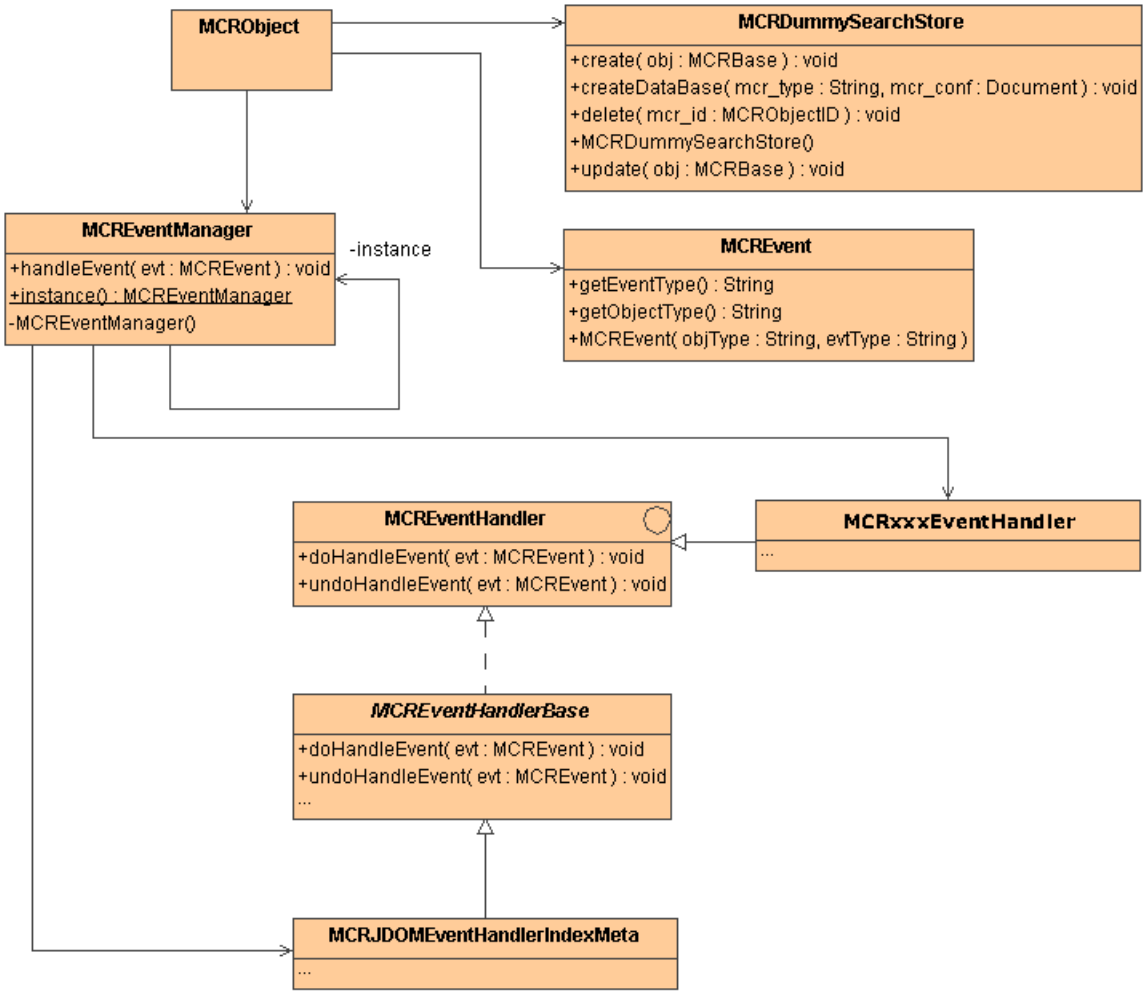


Abbildung 1.6: Klassendiagramm des EventHandler-Modells

4. Die folgende Zeile ruft abschließend den MCREventManager auf und stößt die Handler für die Pipeline an.
   `MCREventManager.instance().handleEvent(evt);`

1.7.2 Die Konfiguration des EventHandler-Managers

Alle Konfigurationen befinden sich im Verzeichnis der Applikation (z. B. DocPortal) in der Datei mycore.properties.private (bzw. mycore.properties.private.template).

In der Version 1.2 von MyCoRe ist es noch erforderlich, für den jeweiligen SearchStore die dummy-Klasse anzugeben:

```
MCR.persistence_jdom_class_name=org.mycore.common.events.MCRDummySearch
Store
```

Nun müssen noch die EventHandler für jede Pipeline (in diesem Fall ist es `MCRObject = MCREvent.OBJECT_TYPE`) in der Reihenfolge ihrer Ausführung angegeben werden. Jeder Handler bekommt dabei eine aufsteigende Nummer.

```
MCR.EventHandler.MCRObject.1.class=org.mycore.backend.jdom.MCRJDOMEvent
HandlerIndexMeta
```

Wollen Sie eigene EventHandler schreiben und diese einbinden, so ist es ratsam diese direkt von `MCREventHandler` abzuleiten und analog zu den bestehenden Handlern einzubinden. Sie können dafür auch frei neue Pipelines und Ereignisse definieren. Den `MCREventManager` können Sie nun an beliebiger Code-Stelle einbauen und ihm ein von Ihnen definiertes Ereignis übergeben. Diese Komponente ist allgemein verwendbar und nicht auf das MyCoRe-Datenmodell festgelegt.

2 Funktionsprinzipien und Implementierungen von Kernkomponenten

2.1 Das Datenmodellen

2.1.1 Struktur der Metadaten

ist in Arbeit

2.1.2 Struktur der Derivate

ist in Arbeit

2.1.3 Struktur der Klassifikationen

ist in Arbeit

2.1.4 Die API des Datenmodells

ist in Arbeit

2.1.5 Erstellen eigener Datenmodelle aus den Grundkomponenten

ist in Arbeit

2.1.6 Erweiterung des Datenmodells

ist in Arbeit

2.2 Suchen und Finden: MyCoRe Query Service

2.2.1 Datenmodell der Suche

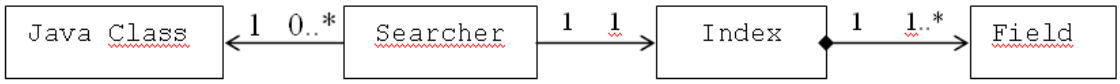
Aufgabe des MyCoRe Query Service ist es, gespeicherte Daten durchsuchbar zu machen. MyCoRe Anwendungen suchen dabei nicht direkt in den Metadaten von Dokumenten bzw. deren XML-Darstellung, oder den Volltexten der zugehörigen Dateien, sondern in daraus abgeleiteten Suchfeldern. Die Definition dieser Suchfelder wird für jede MyCoRe Anwendung in der Konfigurationsdatei `searchfields.xml` vorgenommen (vgl. `docportal/config/searchfields.xml`). Das XML Schema dieser Datei ist in der Datei `searchfields.xsd` definiert.

Mehrere Suchfelder werden zu einem Index logisch zusammengefasst. In DocPortal fasst z.B. der Index `metadata` alle Felder zusammen, die Dokumenten- und Personen-Metadaten durchsuchbar machen. Der Index `content` enthält alle Felder, die Dateien (deren Volltexte, Dateinamen, Dateigröße etc.) durchsuchbar machen. In

jeder MyCoRe Anwendung sind ein Index oder mehrere solcher Indizes in der Datei `searchfields.xml` definiert. Jeder Index besitzt eine eindeutige ID und besteht aus ein oder mehreren Suchfeldern, die jeweils einen innerhalb der Datei `searchfields.xml` eindeutigen Namen besitzen.

```
<searchfields>
  <index id="metadata">
    <field name="title" ... />
    <field name="dateCreated" ... />
    ...
  </index>
  <index id="content">
    <field name="fulltext" ... />
    <field name="size"     ... />
    <field name="numPages" ... />
  </index>
</searchfields>
```

Jedem Index muss ein Searcher zugeordnet werden, der die Felder dieses Index mit Hilfe eines zugrundeliegenden Datenbanksystems (z.B. SQL, IBM Content Manager) oder einer Suchmaschine (z. B. Lucene) durchsuchbar macht. Ein Searcher wird durch eine Java-Klasse implementiert und ist somit zur Laufzeit eine Instanz dieser Klasse.



Searcher und Index sind prinzipiell unabhängig, d.h. es ist möglich, je nach Anwendung für den gleichen Index zwischen verschiedenen Searcher-Implementierungen zu wählen, die jeweils Vor- und Nachteile besitzen. Die gleiche Java-Klasse, z.B. die auf Apache Lucene basierende Implementierung eines Searchers, kann auch für mehrere Indizes verwendet werden, so dass zur Laufzeit mehrere Instanzen der Java-Klasse verwendet werden.

Die Zuordnung von Searchern zu Indizes wird in den `mycore.properties` Konfigurationsdateien vorgenommen. Jeden Index wird ein Searcher zugeordnet. Jeder Searcher besitzt wiederum eine eindeutige ID. Dazu ein Beispiel:

```
# Searcher mit ID 'jdom-metadata' sucht im Index 'metadata'
MCR.Searcher.jdom-metadata.Index=metadata
# Searcher mit ID 'lucene-content' sucht im Index 'content'
MCR.Searcher.lucene-content.Index=content
```

also allgemein

```
MCR.Searcher.[SearcherID].Index=[IndexID]
```

Durch die Trennung von Searcher und Index kann man den für einen bestimmten Index verwendeten Searcher durch Änderung nur einer Konfigurationszeile wechseln (z.B. von Lucene auf Hibernate). Sie können so in Ihrer Anwendung zu einem

späteren Zeitpunkt die Suchimplementierung wechseln, ohne die Daten neu laden zu müssen.

2.2.2 Implementierungen der Suche

Jeder Searcher wird durch eine bestimmte Java-Klasse implementiert. Derzeit sind vier Implementierungen der Suche verfügbar, eine weitere ist in Entwicklung:

- **`org.mycore.backend.lucene.MCRLuceneSearcher`**
  verwendet die Apache Lucene Suchmaschine. Dies ist die für Produktionssysteme derzeit empfohlene Implementierung, die sich sowohl für die Metadatensuche, als auch für die Suche in Volltexten eignet.

- **`org.mycore.backend.hibernate.MCRHibSearcher`**
  verwendet Hibernate und eine zugrundeliegende relationale Datenbank (SQL). Diese Implementierung ist derzeit nicht geeignet für Volltextsuche.

- **`org.mycore.backend.sql.MCRSQLSearcher`**
  verwendet eine zugrundeliegende relationale Datenbank direkt via JDBC. Diese Implementierung ist derzeit nicht geeignet für Volltextsuche.

- **`org.mycore.backend.jdom.MCRJDOMSearcher`**
  verwendet einen Suchindex im Hauptspeicher, so dass keinerlei Daten auf der Festplatte bzw. einem Datenbanksystem gespeichert werden. Diese Implementierung dient als Referenz für Entwickler eigener Suchimplementierungen und hat den Vorteil, dass niemals Daten neu indiziert oder geladen werden müssen, eine Änderung der Suchfeld-Konfiguration ist nach Neustart der Anwendung unmittelbar wirksam. Gut geeignet für die Suche in Metadaten bei der Entwicklung einer Anwendung.

- **`IBM Content Manager`** Suche
  verwendet direkt IBM Content Manager für Suche in Metadaten und Volltext. Diese Suchimplementierung ist noch in Entwicklung.

Die gleiche Implementierung (Java-Klasse) kann auch für verschiedene Searcher-Instanzen mit unterschiedlicher Konfiguration verwendet werden. Jeder Searcher kann eine eigene, individuelle Konfiguration haben, deren Parameter abhängig von der Implementierung sind. Im folgenden Beispiel wird der Searcher mit der ID `lucene-content` durch die Klasse `MCRLuceneSearcher` implementiert. Diese Searcher-Instanz verwendet das Verzeichnis `lucene-index4content/`, um die Daten des Suchindex zu speichern:

```
# Konfiguration des Searchers 'lucene-content'
MCR.Searcher.lucene-content.Class=
  org.mycore.backend.lucene.MCRLuceneSearcher
MCR.Searcher.lucene-content.IndexDir=/repository/lucene-index4content/

# Gemeinsames Locks-Verzeichnis für alle Lucene-basierten Searcher:
MCR.Lucene.LockDir=/repository/lucene-locks/

# Konfiguration des Searchers 'jdom-metadata'
MCR.Searcher.jdom-metadata.Class=
  org.mycore.backend.jdom.MCRJDOMSearcher
MCR.Searcher.jdom-metadata.Index=metadata
```

also allgemein

```
MCR.Searcher.[SearcherID].Class=[ImplementierendeJavaKlasse]
MCR.Searcher.[SearcherID].[KonfigurationsEigenschaft]=[Wert]
```

2.2.3 Abbildung von Daten auf Suchfelder

Damit die Metadaten bzw. die Dateien der in einer MyCoRe Anwendung gespeicherten Dokumente durchsuchbar sind, müssen diese Daten auf logische Suchfelder abgebildet werden. Diese Abbildung kann je nach Implementierung an sich auf beliebige Weise erfolgen, in der Regel werden aber aus den XML-Darstellungen der Daten die Inhalte der Suchfelder mittels XPath-Anweisungen abgeleitet. Die hierarchische XML-Struktur wird dabei quasi „verflacht". Mittels Filtern (PDF, HTML, OpenOffice etc.) können die Volltexte aus Textdateien extrahiert und einem logischen Suchfeld zugeordnet werden.

Typischerweise werden nicht alle Inhalte auf Suchfelder abgebildet. Ein Suchfeld kann auch eine Aggregation von Daten mehrerer Metadatenfelder sein. Suchfelder sind grundsätzlich auch wiederholbar, da evtl. die zugrunde liegenden Metadatenfelder oder Inhalte wiederholt auftreten. Auch kann das gleiche Metadatenfeld auf mehrere Suchfelder abgebildet werden. Dazu einige Beispiele:

Suchfeld	**Entsprechung in den Metadaten bzw. Dateien**
title	`/mycoreobject/metadata/titles/title`
creator	`/mycoreobject/metadata/creators/creator`
author	`/mycoreobject/metadata/creators/creator | /mycoreobject/metadata/publishers/publisher`
content	Volltext der Dateien des Dokumentes
fileSize	`/file/@size`

Suchfelder können nicht nur aus Dokument-Metadaten und Volltexten gebildet werden. Derzeit werden die folgenden Datenquellen unterstützt:

Metadaten der Dokumente (Titel, Autor usw. je nach Datenmodell)

Metadaten der Dateien (Dateiname, Größe, Typ etc.)

Volltext der Dateien:
Dabei wird mittels der in MyCoRe bereitgestellten Filter der Volltext extrahiert und indiziert (z.B. OpenOffice, TXT, HTML, PDF Dateien).

XML-Inhalt der Dateien

Wenn eine gespeicherte Datei eine XML-Datei ist (z.B. eine Excel-Tabelle, als XML gespeichert, oder eine SCORM-Manifest-Datei eines E-Learning Moduls), können deren XML-Elemente qualifiziert durchsuchbar gemacht werden.

Zusatzdaten der Dateien:

In speziellen Anwendungen können damit z.B. extrahierte ID3-Tags aus MP3-Dateien, EXIF-Daten aus Bildern und ähnliche Quellen durchsucht werden).

Beliebige XML-Quellen:

Eigene Anwendungen können die Inhalte beliebiger XML-Quellen indizieren, ohne dass diese XML-Quellen Teil des MyCoRe Datenmodells sein müssen.

In der Datei `searchfields.xml` wird für jedes Feld über das Attribut `source` angegeben, aus welcher Quelle es gebildet wird:

source =	**`MCRFieldDef` Konst.**	**Quelle der Feldwerte**
`objectMetadata`	`OBJECT_METADATA`	`MCRObject.createXML()`
`objectCategory`	`OBJECT_CATEGORY`	alle Klassifikationskategorien
`fileMetadata`	`FILE_METADATA`	`MCRFile.createXML()`
`fileAdditionalData`	`FILE_ADDITIONAL_DATA`	`MCRFile.getAdditionalData()`
`fileXMLContent`	`FILE_XML_CONTENT`	`MCRFile.getContentAsJDOM()`
`fileTextContent`	`FILE_TEXT_CONTENT`	`MCRFile.getContent()` & PlugIn
`xml`	`XML`	beliebiges `org.jdom.Document`
`searcherHitMetadata`	`SEARCHER_HIT_METADATA`	durch `MCRSearcher` ergänzt

In allen Fällen außer `fileTextContent` und `searcherHitMetadata` (also immer, wenn der Feldwert aus einer XML-Quelle abgeleitet wird) wird über die Attribute `xpath` und `value` definiert, wie der Feldwert zustande kommt. Beispiele:

```
<field name="title" type="text" source="objectMetadata"
  xpath="/mycoreobject/metadata/titles/title" value="text()" />
<field name="fileSize" type="integer" source="fileMetadata"
  xpath="file/@size" value="." />
<field name="content" type="text" source="fileTextContent" />
```

Das Attribut **`xpath`** kann Werte enthalten, wie sie in einem **`xsl:select`** oder **`xsl:match`** Attribut erlaubt sind. Die in `searchfields.xml` definierten Felder sind grundsätzlich wiederholbar, d. h. wenn etwa ein Objekt mehrere Titel enthält, werden auch mehrere Feldwerte erzeugt und einzeln indiziert.

Ein Sonderfall stellt die Quelle `objectCategory` dar. Sie muss verwendet werden, wenn nach den Kategorien einer Klassifikation gesucht werden soll. In diesem Fall

gibt das `xpath` Attribut an, welches Element in den Objektmetadaten den Link auf die Klassifikationskategorien enthält. Das `value` Attribut gibt an, ob die ID oder die Labels der Klassifikationskategorie indiziert werden sollen. Beispiel:

```
<field name="origin" type="identifier" source="objectCategory"
   objects="document" xpath="/mycoreobject/metadata/origins/origin"
   value="@ID" />
```

Felder mit der Quellangabe `searcherHitMetadata` werden nicht aus den gespeicherten Daten gebildet, sondern erst bei Zusammenstellen der Trefferliste der Suche von der Suchimplementierung dynamisch ergänzt. Dieser Feldtyp ist für technische Metadaten eines Treffers (score, rank etc.) gedacht. Damit dieses Feld z. B. auch sortierbar ist, muss es in der Datei `searchfields.xml` definiert sein.

2.2.4 Die Attribute sortable und objects

Suchfelder, nach denen man die Trefferliste der Suche sortieren können möchte, müssen explizit über das Attribut `sortable='true'` gekennzeichnet werden.

Manche Suchfelder sollen bzw. können nur für bestimmte Objekttypen oder Dateitypen gebildet werden. Über das Attribut `objects` kann definiert werden, dass ein Suchfeld nur für bestimmte Typen von Metadatenobjekten oder für bestimmte Typen von Dateien (z.B nur denen, für die ein Volltextfilter vorliegt) gebildet werden. Beispiel:

```
<field name="title" type="text" source="objectMetadata"
  objects="document" sortable="true"
  xpath="/mycoreobject/metadata/titles/title" value="text()" />

<field name="content" type="text" source="fileTextContent"
  objects="html pdf ps txt xml msword95 msword97 msppt rtf otd sxw" />
```

2.2.5 Datentypen und Operatoren

Jedem Suchfeld ist ein definierter Datentyp zugeordnet. Der Datentyp bestimmt die möglichen Operatoren für Suchanfragen und legt implizit fest, wie Inhalte dieses Typs behandelt werden (Normalisierung von Umlauten, Stammwortbildung statt exakter Suche etc.). Für jeden Datentyp gibt es eine festgelegte Menge vordefinierter Standard-Operatoren, die jede Searcher-Implementierung unterstützen muss. Darüber hinaus kann eine Implementierung aber auch eigene Datentypen und eigene Operatoren mit erweiterten Suchmöglichkeiten definieren.

Die Definition der Standard-Datentypen und Operatoren erfolgt in der Datei `mycore/config/fieldtypes.xml`. Das XML Schema dieser Datei befindet sich in `mycore/schema/fieldtypes.xsd`. Hier ein Auszug als Beispiel:

```
<fieldtypes>
  <type name="text">
    <operator token="="/>
    <operator token="&gt;"/>
    <operator token="&lt;"/>
    <operator token="&gt;="/>
    <operator token="&lt;="/>
    <operator token="like"/>    <!-- wildcard search using * and ? -->
    <operator token="contains"/> <!-- words at any position  -->
    <operator token="phrase"/>   <!-- a phrase at any position -->
  </type>
  ...
</fieldtypes>
```

In zukünftigen Versionen von MyCoRe wird diese Datei auch proprietäre Datentypen (etwa GIS-Koordinaten) und Operatoren (z. B. proximity-Suche in Lucene) definieren und diese als nur durch bestimmte Implementierungen unterstützte Operatoren kennzeichnen. Diese Funktionalität ist derzeit noch nicht implementiert.

Bei der Konfiguration der Suchfelder ist insbesondere auf die richtige Wahl der Textdatentypen zu achten. Es wird zwischen drei verschiedenen Datentypen für Textfelder unterschieden: `identifier`, `name` und `text`. Die folgenden Standard-Datentypen sind derzeit implementiert:

Datentyp	**Suchoperatoren**	**Beschreibung**
`identifier`	`=, <, >, <=, >=, like`	ID, URN, Dateiname etc., also für exakte Werte, keine Normalisierung oder Stemming
`name`	`=, <, >, <=, >=, like, contains`	Personen- oder Ortsnamen etc., Umlaut-normalisierung, aber kein Stemming
`text`	`=, <, >, <=, >=, like, contains, phrase`	Volltext, Abstract, freier Text, Umlautnormalisierung, Stemming
`date`	`=, <, >, <=, >=`	Datum, vollständig im Format yyyy-MM-dd
`time`	`=, <, >, <=, >=`	Uhrzeit oder zeitliche Dauer, im Format HH:mm:ss
`timestamp`	`=, <, >, <=, >=`	Zeitpunkt, im Format 'yyyy-MM-dd HH:mm:ss'
`boolean`	`=`	'true' oder 'false'
`decimal`	`=, <, >, <=, >=`	Gleitkommazahl, "." als Trennzeichen
`integer`	`=, <, >, <=, >=`	Ganzzahl

Es ist Aufgabe der Suchimplementierung, diese Datentypen auf möglichst geeignete Suchstrukturen (Lucene-/SQL-Typen etc) abzubilden und die Standard-Operatoren in der späteren Suche umzusetzen. Zu beachten ist, dass für Datums-, Zeit- und Boolean-Werte das Format für die Indizierung (wie werden die Felder zur Indizierung übergeben) und die spätere Suche (wie wird ein Wert in einer Query formatiert) exakt festgelegt ist (siehe Formate in obiger Tabelle).

2.2.6 Suchanfragen formulieren

Eine Suchanfrage kann als XML-Dokument oder als Textausdruck formuliert werden. Für Programmierer besteht weiterhin die Möglichkeit, eine Suche als zusammengesetztes Java-Objekt zu formulieren.

Eine einfache Suchbedingung enthält das zu durchsuchende Feld, einen Suchoperator und den Vergleichswert, z. B. Suche nach dem Wort „Optik" im Titel:

```
title contains "Optik"

<condition field="title" operator="contains" value="Optik" />

MCRFieldDef titleField = MCRFieldDef.getDef("title");
new MCRQueryCondition( titleField, "contains", "Optik" );
```

Die Klassen `MCRQueryParser` und `MCRQueryCondition` implementieren die Java-Darstellung einer Query bzw. den Parser, um aus der String- oder XML-Darstellung die Java-Darstellung zu gewinnen und zwischen den Darstellungen zu wechseln.

Einfache Suchbedingungen können über `and`/`or`/`not`-Ausdrücke miteinander verknüpft und so zu komplexeren Suchanfragen zusammengesetzt werden:

```
( not (title contains "Optik") ) and ( date > "2006-02-22" )

<boolean operator="AND">
  <boolean operator="NOT">
    <condition field="title" operator="contains" value="Optik" />
  </boolean>
  <condition field="date" operator="&gt;" value="2006-02-22" />
</boolean>
```

Solche komplexen Suchbedingungen können über die Klassen `MCRAndCondition`, `MCROrCondition` und `MCRNotCondition` aus dem Paket `org.mycore.parsers.bool` auch als Java-Objekte gebildet werden.

2.2.7 Normalisierung von Suchanfragen

Suchanfragen werden vor der Ausführung normalisiert. Insbesondere werden Datumsangaben in Suchausdrücken vom eingegebenen Format (z.B. 22.04.1971) automatisch in das ISO8601-Format transformiert (also 1971-04-22).

Suchbedingungen, die den Operator `contains` verwenden, werden automatisch in einzelne `contains`/`like`/`phrase`/`not`-Bedingungen zerlegt. Beispiel:

```
title contains "-Optik Mecha* 'Lineare Algebra'"
```

wird normalisiert zu

```
(not (title contains Optik)) and (title like Mech*)
and (title phrase 'Lineare Algebra')
```

In der Konsequenz bedeutet das, dass man bei der Textsuche in aller Regel den Operator `contains` verwenden kann. Die Umwandlung in eine `like` und/oder `contains` und/oder `phrase` Suche erfolgt automatisch.

- Worte beginnend mit einem Minuszeichen werden zu einer `not-Bedingung`.
- Worte, die `*` oder `?` enthalten, werden zu einer `like`-Bedingung.
- Wortgruppen in einfachen Anführungszeichen werden zu einer `phrase`-Bedingung.

Eine Suchbedingung kann auch gleichzeitig für mehrere Suchfelder definiert werden. Dazu werden die einzelnen Feldnamen durch Kommata getrennt. Beispiel:

```
title,author contains Goethe
```

entspricht

```
(title contains Goethe) or (author contains Goethe)
```

Diese Funktionalität kann z.B. verwendet werden, um in einer Suchmaske über nur ein Eingabefeld parallel in mehreren Suchfeldern suchen zu können.

2.2.8 Suchen mit MCRSearchServlet

Das Servlet MCRSearchServlet führt Suchanfragen aus und stellt die resultierenden Trefferlisten dar. Die Suchanfrage wird auf verschiedene Weisen akzeptiert:

- **Suche in vordefiniertem Feld:**
  In diesem Fall wird nur ein Parameter übergeben, z. B. `search=Optik`.
  Es wird in einem vordefiniertem Feld mit vordefiniertem Operator gesucht, entsprechend der Konfiguration in `mycore.properties`:
  `MCR.SearchServlet.DefaultSearchField=allMeta`
  `MCR.SearchServlet.DefaultSearchOperator=contains`
  Ein Aufruf mit `search=Optik` sucht dann nach `allMeta contains Optik`.
- **Suche mit komplexem Suchausdruck:**
  Die Suchanfrage wird dabei im Parameter `query` übergeben, z.B.
  `query=title contains Optik`
- **Suche über Namens- und Operator-Parametern:**
  Es können mehrere Parameter übergeben werden, deren Namen den definierten Suchfeldern entsprechen müssen. Die einzelnen Bedingungen werden mit UND verknüpft. Beispiel:
  `title=Optik&author=Kupferschmidt`
  entspricht einer Suche nach
  `(title contains Optik) and (author contains Kupferschmidt)`.

Wenn, wie im obigen Beispiel, kein Suchoperator angegeben ist, wird der in `MCR.SearchServlet.DefaultSearchOperator` definierte Wert verwendet. Alternativ kann auch ein Operator angegeben werden:
`title=Opti*&title.operator=like`
entspricht einer Suche nach
`title like Opti*`
Wenn mehrere Werte für ein Suchfeld angegeben werden, werden diese mit dem Operator `or` verknüpft:
`title=Optik&title=Magnetismus`
entspricht einer Suche nach
`(title contains Optik) or (title contains Magnetismus)`

Mit den drei zuvor beschriebenen Methoden kann jede Suche als statischer Link in einer Webseite eingebunden werden. Als weitere HTTP-Request-Parameter können die Anzahl Treffer pro Seite (`numPerPage`) und die maximale Anzahl auszugebender Treffer (`maxResults`) angegeben werden. Beispiel:

```
/servlets/MCRSearchServlet?title=Optik&maxResults=100&numPerPage=5
```

- **Suchanfrage als XML-Dokument:**
  Hier ist dem `MCRSearchServlet` ein MyCoRe Editor-Formular vorgeschaltet, das eine Suchmaske darstellt. Nach Abschicken der Suchmaskeneingaben durch den Benutzer generiert das Editor Framework daraus ein XML-Dokument, das die Suchanfrage enthält. Die Syntax dieses XML-Dokumentes entspricht der im vorangehenden Kapitel beschriebenen Syntax für Suchanfragen. Das Wurzelelement `query` enthält drei Attribute:

`mask`	Dateiname der Suchmaske
`maxResults`	maximale Trefferzahl
`numPerPage`	Anzahl Treffer pro Seite

  Das Element `conditions` enthält die eigentliche Suchbedingung, entweder formuliert als Menge von verschachtelten XML-Elementen (`format=xml`) oder als textueller Suchausdruck (`format=text`). Beispiel:

```
<query
  mask='editor_form_search-simpledocument.xml'
  maxResults='100' numPerPage='10'
>  <conditions format='xml'>
    <boolean operator='and'>
      <condition field='title' operator='contains' value='Optik' />
      <condition field='author'
                 operator='contains' value='Kupferschmidt' />
    </boolean>
  </conditions>
</query>
```

2.2.9 Indizierung von Feldern

Der Query Service ist logisch von der Speicherung der Inhalte unabhängig, so dass Persistenz- und Suchimplementierungen sehr flexibel kombinierbar und austauschbar sind. Dies bedeutet jedoch nicht, dass eine bestimmte Implementierung nicht dennoch beide Dienste gemeinsam realisieren kann. In einer reinen Open Source Umgebung kann z. B. zur Speicherung von Metadaten und Dateien ein lokales Dateisystem dienen, zur Suche in Metadaten und Dateien kann Lucene verwendet werden. In einer IBM Content Manager Installation kann dieser sowohl die Speicherung als auch Suche in Dateien übernehmen. Es ist aber grundsätzlich z. B. auch möglich, im Content Manager gespeicherte Inhalte mittels einer Lucene Implementierung durchsuchbar zu machen.

Jedem Index (Menge von Suchfeldern) ist ein `Searcher` zugeordnet, der diese Felder durchsuchbar macht. Es gibt nun drei Fälle, auf welche Weise diese Felder für eine spezielle Implementierung des Searchers durchsuchbar werden:

- Es ist keine separate Indizierung notwendig, weil schon durch die Art und Weise der Speicherung der Daten z. B. im IBM Content Manager automatisch eine Durchsuchbarkeit gegeben ist (Ausnahmefall), ODER:
- Wenn sich ein `MCRObject` (Metadaten eines Dokumentes) oder `MCRFile` (Datei mit Volltext) ändert, wird bei diesem Ereignis über einen speziellen `EventHandler`, dem `Indexer`, die Feldwerte aus den Daten extrahiert und in einer eigenen Struktur zwecks Durchsuchbarkeit abgelegt. In der Implementierung gibt es dann zwei Varianten:
  - Der `Indexer` tut dies selbst, indem er die entsprechenden Event-Methoden überschreibt (Ausnahmefall), oder:
  - Der `Indexer` wird als Unterklasse von `MCRSearcher` implementiert und überschreibt die Methoden `addToIndex` und `removeFromIndex` (Regelfall). Alle Feldwerte werden dann automatisch durch die Hilfsklasse `MCRData2Fields` anhand der Konfiguration aus den Daten gebildet.

Aufgabe eines `Indexers` ist es also, Inhalte auf Suchfelder abzubilden und diese in geeigneten Strukturen in einer Datenbank oder einer Suchmaschine durchsuchbar zu machen. Im Regelfall ist daher nach der Speicherung der Inhalte eine Abbildung von

Suchfeldinhalten auf geeignete Backend-Strukturen (SQL-Tabellen, Lucene Index etc). erforderlich. Die Trennung von Such- und Persistenzdiensten und der Aufruf der Indexer erfolgt über den MyCoRe-EventManager. Bei `create`/`update`/`delete` Operationen auf `MCRObject` (Metadaten) und `MCRFile` (Dateien) Objekte wird über den Event Manager ein Ereignis ausgelöst, das ein oder mehrere konfigurierte `Indexer` aufruft. Der `Indexer` ist dafür verantwortlich, auf die für ihn relevanten Änderungen an Inhalten zu reagieren und ggf. seine Suchfeldeinträge zu aktualisieren.

Jeder `Searcher` muss als Unterklasse der abstrakten Klasse `MCRSearcher` implementiert werden. `MCRSearcher` implementiert bereits das `MCREventHandler` Interface. Es sind daher drei Fälle realisierbar:

- Searcher ohne Indexer: Es ist nichts weiter zu tun
- Searcher mit Indexer, wobei die Indizierung mit eigenen Mitteln erfolgt: Die MCRSearcher-Unterklasse überschreibt dann die `MCREventHandlerBase` Methoden
- Searcher mit Indexer, wobei die Feldwerte automatisch gebildet werden sollen: Die MCRSearcher-Unterklasse überschreibt die Methoden `addToIndex()` und `removeFromIndex()`.

Eine Searcher-Implementierung mit Indexer (die also Event Handler ist) kann z.B. wie folgt in den Event-Mechanismus eingebunden werden:

```
MCR.EventHandler.MCRFile.1.Indexer=lucene-content
MCR.EventHandler.MCRObject.4.Indexer=lucene-metadata
```

also allgemein:

```
MCR.EventHandler.[MCRObject|MCRFile].[Nr].Indexer=[SearcherID]
```

Nun werden bei `create`/`update`/`delete` Ereignissen auf Metadaten bzw. Dateien die EventHandler-Methoden der implementierenden Klassen aufgerufen. Der typische Ablauf sieht dann in der Implementierung wie in Abbildung 2.1 dargestellt aus.

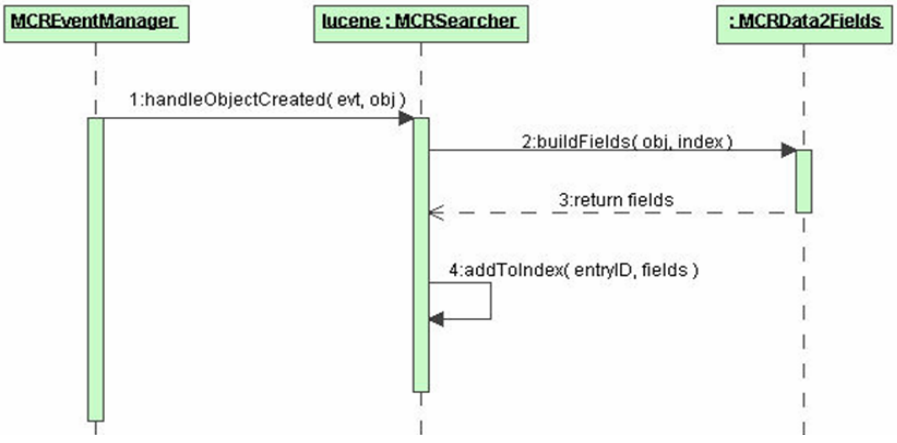


Abbildung 2.1: Typischer Event-Ablauf bei der Suche

2.2.10 Suche über Referenzen

In diesem Abschnitt geht es um die Rückgabe von Suchergebnissen über interne Referenzen. Damit ist gemeint, dass in einem Datensatz vom Typ B ein Verweis in Form des Datentyps *MCRMetaLinkID* auf den Datensatz vom Typ A vorhanden ist. Wird nun im Datensatz B z. B. über eine Klassifikation gesucht, so sollen in der Hit-Liste die ID's des referenzierten Datensätze vom Typ A erscheinen.

Um dies zu realisieren, muss die Klasse *org.mycore.services.fieldquery.MCRSearcher* in den Source-Zweig der betreffenden Applikation kopiert werden. Dort sind die Methoden *handleObjectCreated* und *handleObjectUpdated* wie folgend umzugestalten.

```
...
String returnID = entryID;
if (obj.getId().getTypeId().equals("text")) {
    MCRMetaInterface inter =
                    obj.getMetadataElement("te43s").getElement(0);
    if (inter != null) {
        returnID = ((MCRMetaLinkID)inter).getXLinkHref();
    }
}
addToIndex(entryID, returnID, fields);
...
```

2.3 Die Benutzerverwaltung

Dieser Teil der Dokumentation beschreibt Funktionalität, Design, Implementierung und Nutzung des MyCoRe Subsystems für die Benutzerverwaltung.

2.3.1 Die Geschäftsprozesse der MyCoRe Benutzerverwaltung

Das Benutzermanagement ist die Komponente von MyCoRe, in der die Verwaltung derjenigen Personen geregelt wird, die mit dem System umgehen (zum Beispiel als Autoren Dokumente einstellen). Zu dieser Verwaltung gehört auch die Organisation von Benutzern in Gruppen. Eine weitere Aufgabe dieser Komponente ist das Ermöglichen einer Anmelde-/Abmeldeprozedur.

Ein UseCase-Diagramm (siehe Abbildung 2.2) soll eine Reihe typischer Geschäftsprozesse des Systems zeigen (ohne dabei den Anspruch zu haben, alle Akteure zu benennen oder alle Assoziationen der Akteure mit den Geschäftsprozessen zu definieren).

Offensichtlich dürfen nicht alle Akteure des Systems die Berechtigung haben, alle Geschäftsprozesse durchführen zu können. Daher musste ein System von Privilegien und Regeln implementiert werden: Benutzer/innen haben Privilegien (z.B. die Berechtigung, neue Benutzer/innen zu erstellen). Die Vergabe der Privilegien wird durch die Mitgliedschaft der Benutzer/innen in Gruppen geregelt. Darüber hinaus muss das System definierten Regel gehorchen. So genügt z.B. das Privileg `'add`

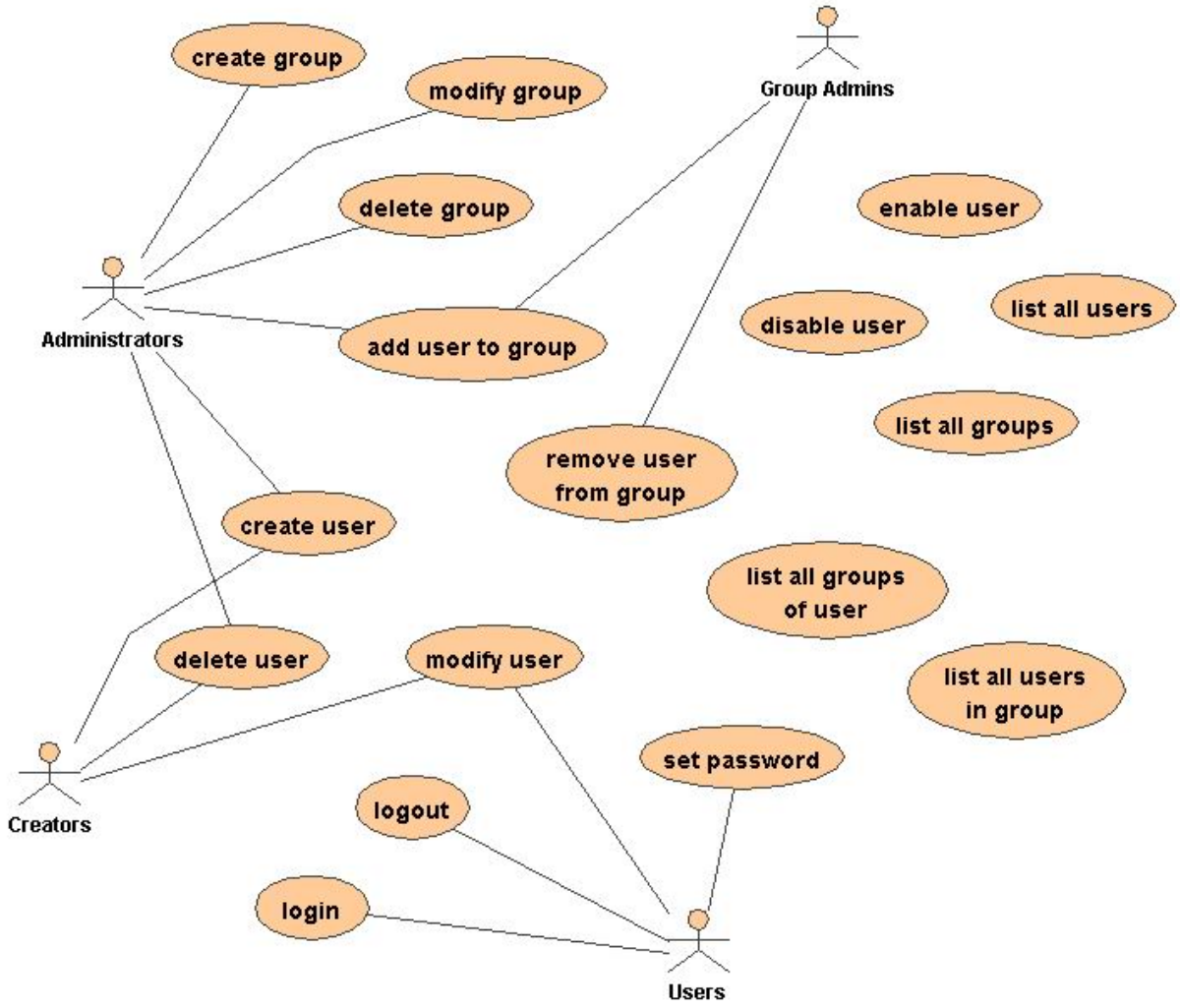


Abbildung 2.2: Geschäftsprozesse der Benutzerverwaltung in MyCoRe

`user to group'` allein nicht, um genau das Hinzufügen eines Benutzers zu einer Gruppe definieren zu können. Die Regel ist in diesem Fall, jeder andere Benutzer mit diesem Privileg aber maximal Zugehörigkeit zu Gruppen vergeben kann, in denen er oder sie selbst Mitglied ist. Auf diese Weise wird verhindert, dass sich ein Benutzer oder eine Benutzerin selbst höhere Privilegien zuweisen kann. Die Privilegien und Regeln der MyCoRe-Benutzerverwaltung werden weiter unten ausgeführt.

2.3.2 Benutzer, Gruppen, Privilegien und Regeln

Die Attribute von Benutzern/innen des Systems können in drei Bereiche klassifiziert werden, den Account-Informationen wie ID, Passwort, Beschreibung usw., den Address-Informationen wie Name, Anrede, Fakultätszugehörigkeit usw. sowie den Informationen über die Mitgliedschaft zu Gruppen. Die aktuell implementierten Benutzerattribute kann man an folgender beispielhafter XML-Darstellung erkennen:
`[ToDo]: to be continued ...`

2.4 ACL-Integration

2.4.1 Strategien der Validierung

Das ACL-System ist nur lose gekoppelt an das Datenmodell von MyCoRe und so sind ACL-Regeln nicht zwangsweise an `MCRObjectID`s gebunden, sondern nehmen als ID jeden String entgegen. Diese Flexibilität kann man sich zu Nutze machen, wenn es um die Überprüfung der Zugriffsrechte geht. Bei MyCoRe gibt es drei vordefinierte Methoden, die über Properties ausgewählt werden.

Methode 1: ObjectID

Diese Methode ist der Standardfall von DocPortal. Zu jeder ObjectID wird die ACL-Regel mit gleicher ID genommen. Existiert diese nicht, wird der Zugriff verweigert. Die Pflege der ACL-Regeln, z.B. die Integration von Standardregeln, übernimmt das SimpleWorkFlow-Modul, das im UsersGuide beschrieben wird. Dabei wird jedem neu angelegtem Objekt eine objektspezifische Regel angehangen. Beim Einstellen in den MyCoRe-Server, entfernt ein Eventhandler die dort vorhandene Regeldefinition und legt eine entsprechende Regel für das Dokument an. Methode 1 ähnelt in diesem Zusammenhang der Unix-Rechteverwaltung und dem dort benutzten Befehl `umask`. Änderungen an den Standardregeln gelten für neu eingestellte Objekte. Folgende Properties sind für Methode 1:

```
MCR.EventHandler.MCRObject.1.class=org.mycore.access.MCRAccessEventHandler
MCR.EventHandler.MCRDerivate.1.class=org.mycore.access.MCRAccessEventHandler
MCR.Access_strategy_class_name=org.mycore.access.strategies.MCRObjectIDStrategy
```

Methode 2: Objekt-Typ

Diese Methode arbeitet wie Methode 1, nutzt jedoch einen anderen Eventhandler, der nicht für jedes Objekt eine Regel anlegt, sondern diese ignoriert. Das bedeutet, dass man für einzelne Objekte explizit eine Regel anlegen muss oder es tritt beim Überprüfen die erweiterte Behandlung in Kraft. Diese sieht ein Zurückfallen auf die Regel des Objekttyps vor und notfalls die Anwendung einer Standardregel. Die Regel für ein Objekttyp lässt sich über die Kommandozeile anlegen.

```
update permission read for id default_<objekttyp> with rulefile grant-all.xml
update permission writedb for id default_<objekttyp> with rulefile grant-editors.xml
update permission deletedb for id default_<objekttyp> with rulefile grant-admin.xml
```

Heißt der Objekttyp `document`, so lautet die ID für das ACL-System `default_document`. Die Standardregel, die notfalls nach der Objekttyp-Regel überprüft wird, lautet `default`. Beispiele für die oben genannten Regeldateien (`grant-*.xml`), finden sich in DocPortal unter `config/acl`. Methode 2 reduziert gegenüber Methode 1 den Verwaltungsaufwand, sowohl auf Administratorseite, wie auch auf Datenbankseite, wegen der reduzierten Zahl an Regelzuweisungen. So treten Änderungen an den Standardregeln sofort für alle entsprechenden Objekte in Kraft.

Folgende Properties sind für die Methode 2:

```
MCR.EventHandler.MCRObject.1.class=org.mycore.access.MCRRemoveAclEventHandler
MCR.EventHandler.MCRDerivate.1.class=org.mycore.access.MCRRemoveAclEventHandler
MCR.Access_strategy_class_name=org.mycore.access.strategies.MCRObjectTypeStrategy
```

Methode 3: Vererbung von Regeln

Diese Methode arbeitet wie Methode 1, nutzt jedoch wieder den Eventhandler von Methode 2. Entsprechend müssen Regeln für MCRObjectIDs selbst angelegt und gepflegt werden. Sollte für eine MCRObjectID keine ACL-Regel hinterlegt sein, so wird Methode 3 rekursiv mit der MCRObjectID des Vaterobjekts angewandt, bis zu einer MCRObjectID eine ACL-Regel existiert. Sollte es keine ACL-Regel geben, wird der Zugriff verweigert. Methode 3 ähnelt also dem Vererbungsmodell von MyCoRe. Folgende Properties sind für die Methode 3:

```
MCR.EventHandler.MCRObject.1.class=org.mycore.access.MCRRemoveAclEventHandler
MCR.EventHandler.MCRDerivate.1.class=org.mycore.access.MCRRemoveAclEventHandler
MCR.Access_strategy_class_name=org.mycore.access.strategies.MCRParentRuleStrategy
```

2.5 Die Backend-Stores

Backend	Organisation	Suche	Ablage	Volltexte	Bemerkung
HSQLDB	x				frei verfügbar; für kleine Lösungen
MySQL	x				frei verfügbar; für mittlere Lösungen
IBM DB2	x				kommerzielles Produkt; für große Lösungen
JDOM-Tree		x			in MyCoRe enthalten; für kleine Lösungen
nativ SQL		x			benötigt ein SQL / Hibernate-Backend; je nach Backend
Lucene		x		x	frei verfügbar; für mittlere bis große Lösungen
eXist		x			frei verfügbar; für mittlere Lösungen
IBM CM 8.x		x	x	x	kommerzielles Produkt, für große Lösungen
FileSystem			x		je nach Plattenplatz
Helix			x		kommerzielles Produkt; für Audio-/Video-Lösungen

Tabelle 2.1: Übersicht der MyCoRe-Backends

Im Backend-Bereich muss zwischen den verschieden Aufgaben der Stores unterschieden werden. Es gib Stores für die Speicherung organisatorischer Informationen (z. B. User-Daten, XML-Daten usw.), Stores für die Suche und für die Ablage der eigentlichen digitalen Objekte und deren Volltexte. Dabei kann die genutzte Backend-Software ggf. auch für mehrere Stores verwendet werden. Eine Übersicht gibt Tabelle 2.1.

2.5.1 Hibernate oder nativ SQL?

MyCoRe in der Version 1.2 bietet 2 Möglichkeiten, die Stores für die organisatorischen Daten einzubinden.

Zum einen wird über das Package `org.mycore.backend.sql` ein direkter Zugriff auf relationale Datenbanken via JDBC realisiert. Der Vorteil davon sind optimale Zugriffszeiten. Nachteilig kann sich auswirken, dass nur einige der am Markt

verfügbaren Datenbanken integriert und getestet haben. Es kann also bei Verwendung anderer Datenbanken ggf. zu Problemen kommen (besonders beim automatischen Anlegen der Tabellen), da diese nicht getestet wurden.

Der zweite Weg ist die Nutzung unserer Hibernate-Integration. Hier übernimmt das freie Paket „Hibernate" die Anpassung an die jeweils darunter liegende Datenbank. Es wird also der gesamte Zugriff über ein fest definiertes API geregelt. Der Nachteil ist ein leichter Performance-Verlust, da ja alle Daten durch das API verwaltet werden. Die Klassen zur Arbeit mit Hibernate stehen in `org.mycore.backend.hibernate`.

Welche der beiden Zugriffsarten nun in Ihrem konkreten Projekt genommen wird hängt von den ganz spezifischen Eigenschaften der Anwendung und deren Umgebung sowie den personellen Ressourcen ab. Das MyCoRe-System wurde mit beiden Varianten getestet.

Hinweis:
Während es in der nativ-SQL-Anwendung möglich ist, für jeden XML-Daten-Store eine eigene Tabelle anzugeben, werden die XML-Daten unter Hibernate immer in einer gemeinsamen Tabelle gehalten (Property `MCR.xml_store_sql_table`).

2.5.2 Das Search-Backend JDOM-Tree

Das MyCoRe-Paket bietet eine simple Standard-Lösung für die Suche in kleinen Beispielanwendungen, ohne dass zusätzliche externe Produkte verwendet werden müssen. Die XML-Daten werden im JDOM-Tree zum Startzeitpunkt der Applikation direkt aus der XML-SQL-Tabelle gelesen, teilweise hinsichtlich der Umlaute normalisiert und im weiteren Verlauf der Anwendung im Hauptspeicher verwaltet. `Create`, `Update`, `Delete` auf die XML-SQL-Tabellen wird direkt mit dem im Speicher befindlichen Datenbaum synchronisiert.

Die XPath-Suchanfrage wird in ein XSL-Stylesheet umgewandelt, welches gegen die XML-Daten im Hauptspeicher läuft. Als Ergebnis wird die Liste der Treffer-IDs zurückgegeben. Für die Datumssuche wurde eine zusätzliche XSL-Funktion implementiert. Suchdaten im Operator `contains` werden hinsichtlich der Umlaute normalisiert (z. B. `contains("Eindrücke")` → `contains("eindruecke")`).

Metadaten-Typ	Felder
MCRMetaLangText	`tag/subtag/text()`

MCRMetaPersonName	`tag/subtag/firstname/text()`
	`tag/subtag/callname/text()`
	`tag/subtag/surname/text()`
	`tag/subtag/fullname/text()`

MCRMetaInstitutionName	`tag/subtag/fullname/text()`

Tabelle 2.2: Felder mit Umlautnormalisierung im Search-Store

Hinweis:
Dieser Search-Store ist nur bis zu einigen 100 Datensätzen performant und nicht für Produktionssysteme gedacht.

2.5.3 Das Search-Backend für IBM Content Manager 8.x

Hinweis:
Strings, welche NUR Zahlen enthalten, werden als Zahl interpretiert. Läuft die Anfrage gegen ein als `VarChar` definiertes Attribut, kommt es zum Fehler durch den Content Manager (CM). Ergänzen Sie den Zahlen-String in der Suche z.B. mit einem * und benutzen Sie den Like-Operator. Beispiel `like 0004*`.

2.5.4 Das Search-Backend für XML:DB

Auf dem Markt gibt es eine Reihe von XML-DBs. Im Bereich der kommerziellen Systeme wurde für MyCoRe einmal prototypisch Tamino getestet. Aus Kostengründen wurde jedoch auf eine Produktionsumgebung verzichtet. Im Bereich freier Software kommt bei MyCoRe das Produkt eXist ins Spiel. Für Projekte, die mittlere Leistungsanforderungen haben (bis ca. 8000 Datensätze) ist eXist auch im Produktionsbetrieb im Einsatz. Bei größeren Datenmengen kann es derzeit zu Performance-Problemen im Bereich Update kommen.

eXist ist extern zu beziehen und zu installieren. Anschließend wird es über die Konfiguration eingebunden (siehe UserGuide). Analog zum JDOM-Backend wird auch für das XML:DB Backend eine Umlautnormalisierung für die durchsuchbaren Daten und die Queries vorgenommen. Als Ergebnis der Suche wird die Liste der Treffer-IDs zurückgegeben.

Hinweis:
eXist ignoriert derzeit einige Schachtelungskonstrukte einer Query mit '[...]'. Daher kann das Ergebnis der Anfrage mehr Treffer haben, als bei korrekter Ausführung der Query.

2.6 Die Frontend Komponenten

2.6.1 Erweiterung des Commandline-Tools

Dieser Abschnitt beschäftigt sich mit der Struktur des Commandline-Tools und dessen Erweiterung mit eigenen Kommandos. Dem Leser sei vorab empfohlen, den entsprechenden Abschnitt im MyCoRe-UserGuide durchzuarbeiten.

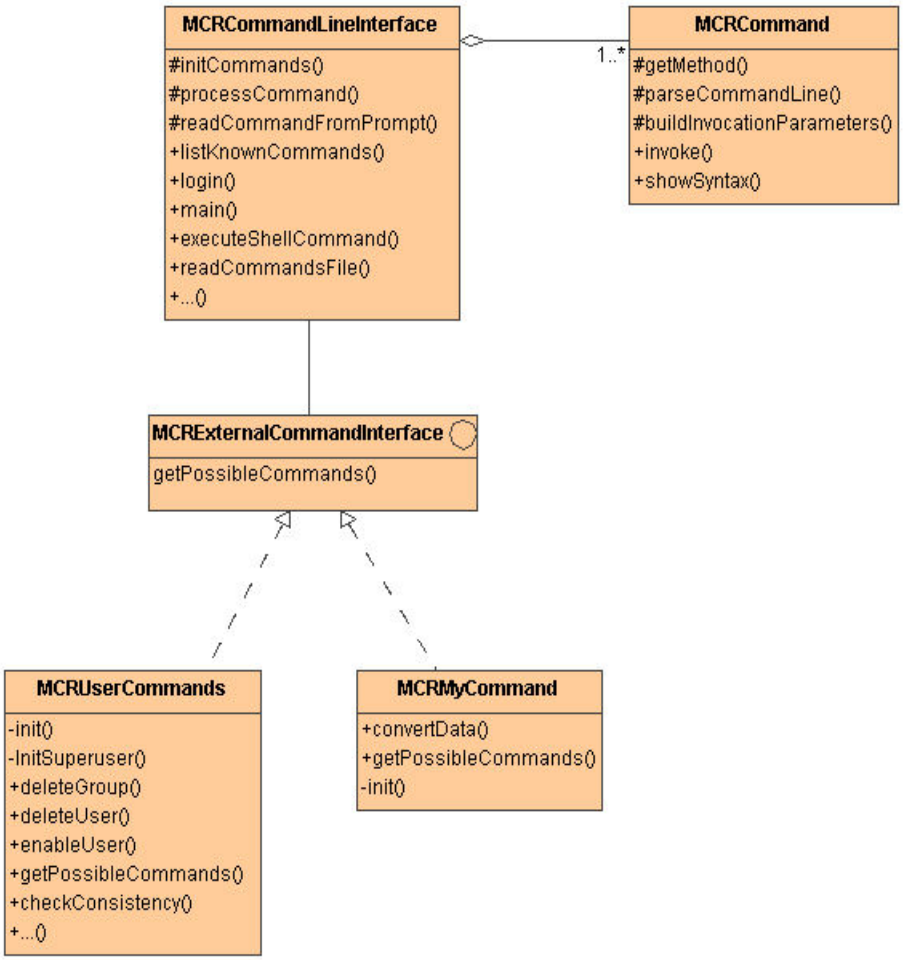


Abbildung 2.3: Zusammenhang der Java-Klassen

Das Commandline-Tool ist die Schnittstelle für eine interaktive Arbeit mit dem MyCoRe-System auf Kommandozeilen-Basis. Sie können dieses System ebenfalls dazu verwenden, mittels Skript-Jobs ganze Arbeitsabläufe zu automatisieren. Dies ist besonders bei der Massendatenverarbeitung sehr hilfreich. In DocPortal werden Ihnen schon in den Verzeichnissen unixtools bzw. dostools eine ganze Reihe von hilfreichen Skripts für Unix bzw. MS Windows mitgegeben.

All diese Skripte basieren auf dem Shell-Skript `bin/mycore.sh` bzw. `bin/mycore.cmd`, welches im Initialisierungsprozess der Anwendung via ant mit gebaut wird (`ant create.unixtools` bzw. `ant create.dostools`). Sollten Sie zu einem späteren Zeitpunkt eventuell einmal *.jar-Dateien in den lib-Verzeichnissen ausgetauscht haben oder sonstige Änderungen hinsichtlich des Java-CLASSPATH durchgeführt haben, so führen Sie für ein Rebuild des MyCoRe-Kommandos ein `ant scripts` durch.

Die Abbildung 2.3 soll einen Überblick über die Zusammenhänge der einzelnen Java-Klassen im Zusammenhang mit der nutzerseitigen Erweiterung des Commandline-Tools geben.

Es ist relativ einfach, weitere Kommandos hinzuzufügen. In DocPortal sind bereits alle nötigen Muster vorhanden.

1. Im Verzeichnis `~/docportal/sources/org/mycore/frontend/cli` finden Sie eine Java-Klasse `MCRMyCommand.java`. Diese ist ein Muster, kopieren Sie sie in eine Java-Klasse z. B. `MCRTestCommand.java`. Die Klasse kann im Package `org.mycore.frontend.cli` liegen, sie können Sie aber auch in den Bereich tun, zu dem es logisch gehört.
2. Ersetzen Sie alle `MCRMyCommand`-String durch `MCRTestCommand`.
3. Im Konstruktor werden nun alle neuen Kommandos definiert. Hierzu werden der `ArrayList command` jeweils zwei weitere Zeilen hinzugefügt. Die erste enthält den Text-String für das Kommando. Stellen, wo Parameter eingefügt werden sollen, sind mit {...} zu markieren, wobei ... eine fortlaufende Nummer beginnend mit 0 ist.[3] In der zweiten Zeile ist nun der Methodenaufruf anzugeben. Für jeden Parameter ist das Schlüsselwort `String` anzugeben.
4. Nun muss das eigentliche Kommando als Methode dieser Kommando-Klasse implementiert werden. Orientieren Sie sich dabei am mitgelieferten Beispiel.[4]
5. Compilieren Sie nun die neue Klasse mit `cd ~/docportal`; `ant jar`
6. Als letztes müssen Sie die Klasse in das System einbinden. Die mit dem MyCoRe-Kern mitgelieferten Kommandos sind bis auf die Basis-Kommandos über die Property-Variable `MCR.internal_command_classes` in `~/docportal/mycore.properties` dem System bekannt gemacht. Für externe Kommandos steht hierfür in der Konfigurationsdatei `mycore.properties.application` die Variable `MCR.external_command_classes` zur Verfügung. Hier können Sie eine mit Komma getrennte Liste Ihrer eigenen Kommando-Java-Klassen angeben.
7. Wenn Sie nun `mycore.sh` bzw. `mycore.cmd` starten und DEBUG für den Logger eingeschaltet haben, so sehen Sie Ihre neu integrierten Kommandos.

2.6.2 Das Zusammenspiel der Servlets mit dem MCRServlet

Als übergeordnetes Servlet mit einigen grundlegenden Funktionalitäten dient die Klasse `MCRServlet`. Die Hauptaufgabe von `MCRServlet` ist dabei die Herstellung der Verbindung zur Sessionverwaltung (siehe Abschnitt „Die Session-Verwaltung"). Das Zusammenspiel der relevanten Klassen ist im Klassendiagramm (Abbildung 2.4) verdeutlicht.

[3] siehe Beispielcode

[4] Die Methode `convertData` sollten Sie in Ihrer Klasse löschen. Ebenso die Definition in der commands-ArrayList.

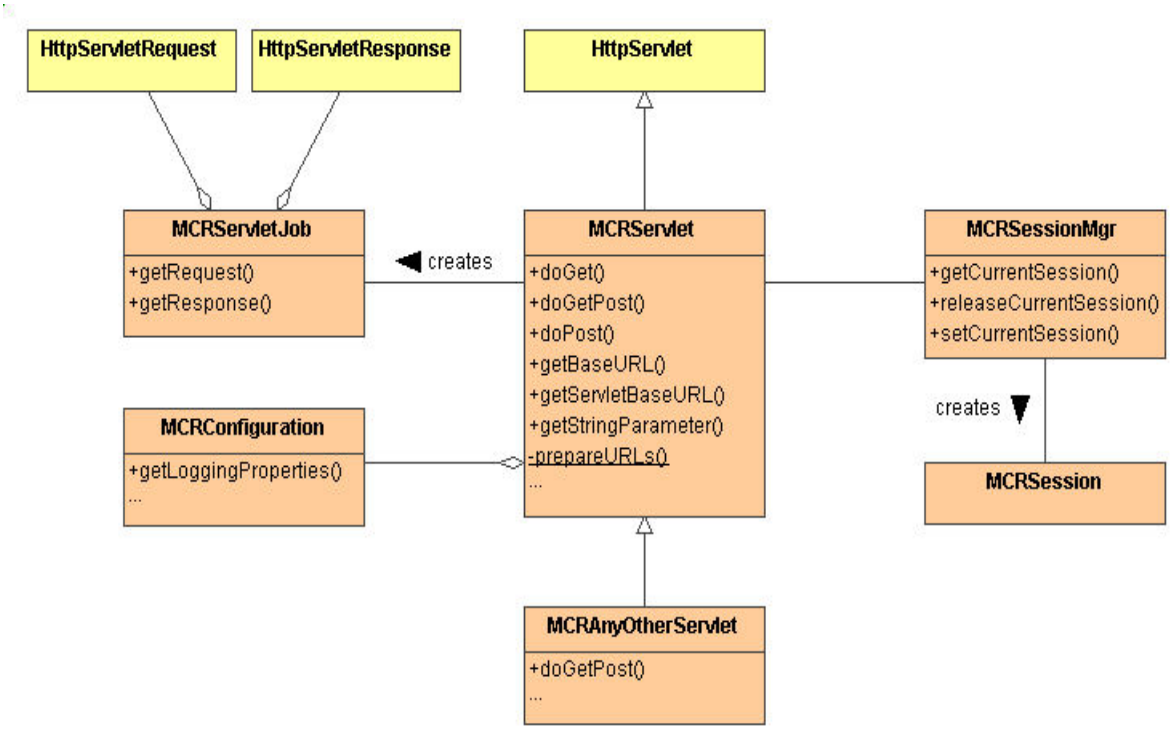


Abbildung 2.4: Klassendiagramm `Common Servlets`

Wie an anderen Stellen im MyCoRe-System auch, kann auf Konfigurationsparameter wie zum Beispiel den Einstellungen für das Logging über das statische Attribut `MCRConfiguration` zugegriffen werden. Dies wird ausführlich in einem anderen Kapitel beschrieben.

`MCRServlet` selbst ist direkt von `HttpServlet` abgeleitet. Sollen andere Servlets im MyCoRe-Softwaresystem die von MCRServlet angebotenen Funktionen automatisch nutzen, so müssen sie von `MCRServlet` abgeleitet werden. Im Klassendiagramm ist das durch die stellvertretende Klasse `MCRAnyOtherServlet` angedeutet. Es wird empfohlen, dass die abgeleiteten Servlets die Methoden `doGet()` und `doPost()` nicht überschreiben, denn dadurch werden bei einem eingehenden Request auf jeden Fall die Methoden von `MCRServlet` ausgeführt.

Der Programmablauf innerhalb von `MCRServlet` ist im folgenden Sequenzdiagramm (siehe Abbildung 2.5) dargestellt. Bei einem eingehenden Request (`doGet()` oder `doPost()`) wird zunächst an `MVRServlet.doGetPost()` delegiert.[5]

[5]Bei dieser Delegation wird ein Parameter mitgeführt, über den feststellbar ist, ob es sich um einen GET- oder POST-Request gehandelt hat.

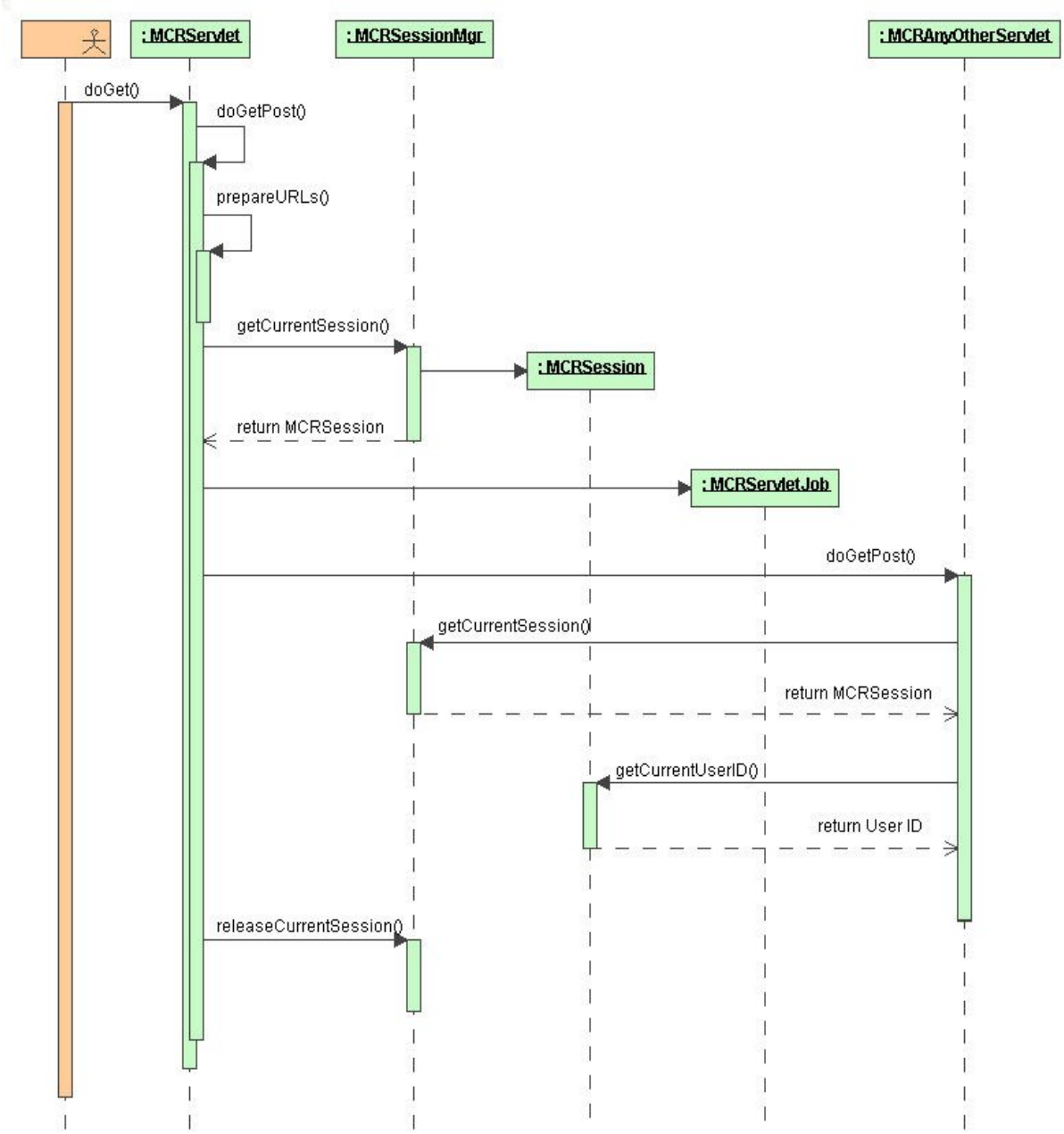


Abbildung 2.5: Sequenzdiagramm `Common Servlets`

Falls nicht schon aus vorhergehenden Anfragen an das `MCRServlet` bekannt, werden in `doGetPost()` die Base-URL und die Servlet-URL des Systems bestimmt. Dabei besteht die Servlet-URL aus der Base-URL und dem angehängten String '`servlets/`'. Darauf folgend wird die für diese Session zugehörige Instanz von `MCRSession` bestimmt. Das Verfahren dazu ist im Ablaufdiagramm (Abbildung 2.6) dargestellt.

Die Session kann bereits durch vorhergehende Anfragen existieren. Falls dies der Fall ist, kann das zugehörige Session-Objekt entweder über eine im `HttpServletRequest` mitgeführte SessionID identifiziert oder direkt der HttpSession entnommen werden. Existiert noch keine Session, so wird ein neues Session-Objekt über den Aufruf von `MCRSessionMgr.getCurrentSession()` erzeugt. Nachfolgend wird das Session-Objekt an den aktuellen Thread gebunden und zusätzlich in der HttpSession abgelegt.

Im Sequenzdiagramm gehen wir davon aus, dass die Sitzung neu ist und deswegen ein Session-Objekt über `MCRSessionMgr.getCurrentSession()` erzeugt werden muss. Schließlich wird eine Instanz von `MCRServletJob` erzeugt. Diese Klasse ist nichts weiter als ein Container für die aktuellen HttpServletRequest und HttpServletResponse Objekte und hat keine weitere Funktionalität (siehe Klassendiagramm, Abbildung 2.4).[6]

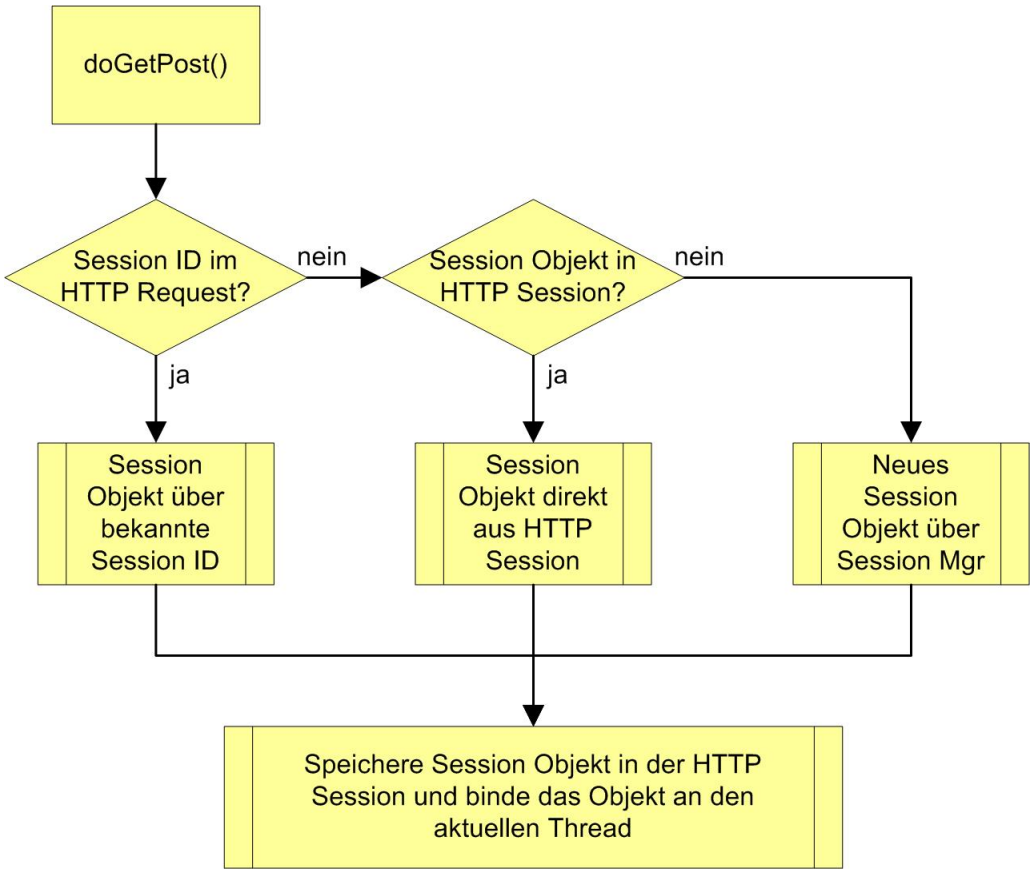


Abbildung 2.6: Ablaufdiagramm für MCRServlet.doGetPost()

An dieser Stelle wird der Programmfluss an das abgeleitete Servlet (in diesem Beispiel `MCRAnyOtherServlet`) delegiert. Dazu muss das Servlet eine Methode mit der Signatur

```
public void doGetPost(MCRServletJob job) {}
```

implementieren. Wie das Sequenzdiagramm beispielhaft zeigt, kann `MCRAnyOtherServlet` danach gegebenenfalls auf das Session-Objekt und damit auf die Kontextinformationen zugreifen. Der Aufruf an den SessionManager dazu wäre:

```
MCRSession mcrSession=MCRSessionMgr.getCurrentSession();
```

Es sei bemerkt, dass dies nicht notwendigerweise genau so durchgeführt werden muss. Da wegen der geschilderten Probleme mit `threadlocal` Variablen in Servlet-Umgebungen das Session-Objekt auch in der HttpSession abgelegt sein muss,

[6]Das Speichern des Session-Objekts in der HttpSession ist notwendig, weil in einer typischen Servlet-Engine mit Thread-Pool Umgebung nicht davon ausgegangen werden darf, dass bei aufeinander folgenden Anfragen aus demselben Kontext auch derselbe Thread zugewiesen wird.

könnte man die Kontextinformationen auch aus der übergebenen Instanz von `MCRServletJob` gewinnen.

2.6.3 Das Login-Servlet und MCRSession

Das `LoginServlet`, implementiert durch die Klasse `MCRLoginServlet`, dient zum Anmelden von Benutzern und Benutzerinnen über ein Web-Formular. Die Funktionsweise ist wie folgt: Wie in Abschnitt 3.7.2 empfohlen, überschreibt `MCRLoginServlet` nicht die von `MCRServlet` geerbten Standard-Methoden `doGet()` und `doPost()`. Meldet sich ein Benutzer oder eine Benutzerin über das `MCRLoginServlet` an, so wird daher zunächst die Funktionalität von `MCRServlet` ausgenutzt und die in Abschnitt 3.7.2 beschriebene Verbindung zur Sessionverwaltung herstellt. Wie dort ebenfalls beschrieben, wird der Programmfluss an das Login-Servlet über die Methode `MCRLoginServlet.doGetPost()` delegiert. Der Ablauf in `doGetPost()` wird im Diagramm auf Abbildung 2.7 dargestellt und ist

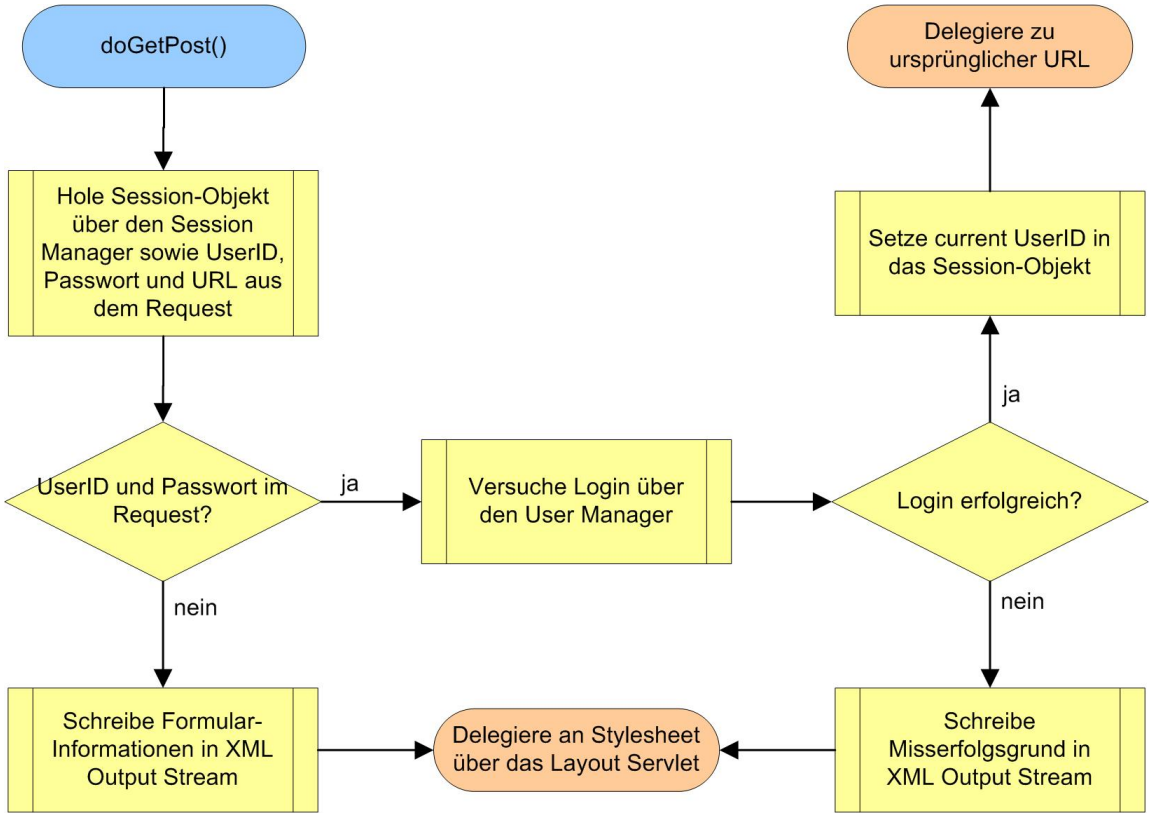


Abbildung 2.7: Ablaufdiagramm für MCRLoginServlet.doGetPost()

selbsterklärend.

Der resultierende XML Output-Stream muss vom zugehörigen Stylesheet verarbeitet werden und hat die in Abbildung 2.8 gezeigte Syntax.

Bei einer missglückten Anmeldung wird der Grund dafür in Form eines Attributes auf `true` oder `false` gesetzt. Das Stylesheet kann dann die entsprechende Meldung ausgeben. Die Gast-User-ID und das Gast-Passwort werden aus einer Konfigurationsdatei gelesen. Die URL schließlich wird dem Http-Request entnommen

```
<mcruser unknownuser="true||false" userdisabled="true||false" invalidpassword="true||false">
    <guestid>...</guestid>
    <guestpwd>...</guestpwd>
    <url>...<url>
</mcruser>
```

Abbildung 2.8: XML Output des LoginServlets

und sollte dort von der aufrufenden Seite bzw. vom aufrufenden Servlet gesetzt sein. Ist sie nicht gesetzt, so wird die Base-URL des MyCoRe-Systems verwendet.

2.6.4 Generieren von Zip-Dateien

Das Zip-Servlet, implementiert durch die Klasse `MCRZipServlet`, dient dem Ausliefern der Derivate und der Objektmetadaten als gepackte Zip-Datei. Aus der Konfigurationsdatei `mycore.properties.zipper` holt sich das Servlet über die Variable `MCR.zip.metadata.transformer` den Namen des Stylesheets, welches die Metadatentransformation in das gewünschte Auslieferungsformat vornimmt. In DocPortal verwenden wir hierfür Qualified Dublin Core.

Aufrufmöglichkeiten des Servlets:

```
$ServletsBaseURL/MCRZipServlet?id=MCRID

$ServletsBaseURL/MCRZipServlet?id=MCRID/foldername
```

MCRID ist die ID eines Objekts vom Typ `<mycoreobject>` oder `<mycorederivate>`. Im Fall von `<mycoreobject>` werden die Dateien aller dem Objekt zugeordneten Derivate und ein XML-File mit den Metadaten des Objekts zusammengepackt. Im Fall von `<mycorederivate>` werden alle Dateien des angegebenen Derivats zusammengepackt. Die Option `MCRID/foldername` ist nur zulässig, wenn `MCRID` ein Objekt vom Typ `<mycorederivate>` bezeichnet. Dann wird nur der mit `foldername` angegebene Ordner des betreffenden Derivats gezippt.

Wer geschützte Inhalte anbietet, sollte das Zip-Servlet erst dann in seine Anwendung integrieren, wenn die Zugriffskontrolle in MyCoRe gewährleistet werden kann. Dies ist momentan (04.2005) noch nicht der Fall, das Zip-Servlet lässt sich mit jeder MCRID aufrufen.

2.7 XML Funktionalität

2.7.1 URI-Resolver

Die Klasse `org.mycore.common.xml.MCRURIResolver` implementiert einen Resolver, mit dem an verschiedenen Stellen im MyCoRe-System XML-Daten über URI's gelesen werden können. Der Resolver wird zur Zeit an folgenden Stellen eingesetzt:

- Bei der Verarbeitung von Stylesheets im LayoutServlet, wenn XML-Daten über die XSL-Funktion `document()` in ein Stylesheet nachgeladen werden oder wenn ein untergeordnetes Stylesheet mittels `xsl:include` nachgeladen wird.

- Beim Import von Editor-Definitionsteilen mittels des `include`-Elementes des Editor-Frameworks.

Der Resolver unterstützt die folgenden Schemata bzw. Protokolle:

```
file://[Pfad]
```

liest eine statische XML-Datei vom Dateisystem des Servers

Beispiel: `file:///usr/local/tomcat/conf/server.xml`
liest die Datei `/usr/local/tomcat/conf/server.xml`

```
webapp:[Pfad]
```

liest eine statische XML-Datei vom Dateisystem der Web-Applikation. Im Gegensatz zur file()-Methode kann der Pfad der zu lesenden Datei relativ zum Wurzelverzeichnis der Web-Applikation angegeben werden. Der Zugriff erfolgt direkt, d.h. ohne HTTP Request oder Anwendung eines Stylesheets.

Beispiel: `webapp:config/labels.xml`

```
http://[URL]
https://[URL]
```

liest eine XML-Datei von einem lokalen oder entfernten Webserver

```
request:[Pfad]
```

liest eine XML-Datei durch einen HTTP Request an ein Servlet oder Stylesheet innerhalb der aktuellen MyCoRe-Anwendung. Im Gegensatz zur http/https Methode ist der Pfad relativ zur Basis-URL der Web-Applikation anzugeben, die MCRSessionID wird automatisch als HTTP GET Parameter ergänzt.

Beispiel:
```
request:servlets/MCRQueryServlet?XSL.Style=classif-to-
items&type=class&hosts=local&lang=de&query=/mycoreclass[
@ID='DocPortal_class_00000002']
```

```
resource:[Pfad]
```

liest eine XML-Datei aus dem CLASSPATH der Web-Applikation, d.h. die Datei wird zunächst im Verzeichnis `WEB-INF/classes/` und als nächstes in einer der jar-Dateien im Verzeichnis `WEB-INF/lib/` der Web-Applikation gesucht. Diese Methode bietet sich an, um statische XML-Dateien zu lesen, die in einer jar-Datei abgelegt sind.

Beispiel: `resource:ContentStoreSelectionRules.xml`

```
session:[Key]
```

liest ein XML-Element, das als JDOM-Element in der aktuellen MCRSession abgelegt ist. Mittels der `put()` Methode der Klasse MCRSession kann analog zu einer Java-Hashtable unter einem Schlüssen ein Objekt abgelegt werden. Ein Servlet kann so z. B. ein JDOM-Element in der `MCRSession` ablegen, den Schlüssel einem Stylesheet

über einen XSL-Parameter mitteilen. Der MyCoRe Editor kann dieses JDOM-Element dann mittels der `get()` Methode aus der Session lesen.

Beispiel: `session:mylist`
liest das JDOM XML-Element, das als Ergebnis von
`MCRSessionMgr.getCurrentSession().get("mylist");`
zurückgegeben wird.

```
mcrobject:[MCRObjectID]
```

liest die XML-Darstellung der Metadaten eines MCRObject.

Beispiel: `mcrobject:DocPortal_document_07910401`

```
classification:[Classification Query]
```

gibt eine Klassifikation in unterschiedlichen Formaten aus, wobei „Classification Query" folgendes Format hat:

```
{editor['['formatAlias']']|metadata}:{Levels}:{parents|children}:{ClassID}[:CategID]
```

Die einzelnen Parameter sind durch Doppelpunkte getrennt

1. Rückgabetyp ist wahlweise im MyCoRe `metadata` Format oder für eine Editor-Selectbox (`editor`). Letztere kann für den Label-Text noch unterschiedliche Formatanweisungen enthalten, die mit `formatAlias` referenziert werden.
   Das Property `MCR.UriResolver.classification.format.{formatAlias}` enthält dann die Formatieranweisung. Diese besteht aus beliebigem Text kombiniert mit Platzhaltern:
   1. `{id}` steht für die Kategorie-ID,
   2. `{count}` steht für die Zahl der zugeordneten MyCoRe-Objekte,
   3. `{text}` steht für das Attribut text im label-Tag der Klassifikationsdefinition,
   4. `{description}` steht für das Attribut description im label-Tag der Klassifikationsdefinition.
2. `Levels` gibt an, wieviel Hierarchiestufen dargestellt werden. Bei Angabe der CategID ist dies die Anzahl der Kindkatergoriehierarchiestufen. Ist Levels „`-1`" angegeben, so bedeutet dies „komplette Hierarchie".
3. `parents` oder `children` gibt an, ob bei Angabe einer `CategID` zusätzlich alle übergeordneten Kategorien mit zurückgegeben werden (`parents`) oder ob nur die Kinder der Kategorie berücksichtigt werden sollen. Bei Angabe eines positiven Levels und „`parents`" werden sowohl die Eltern ausgegeben, wie auch `{Levels}` Hierarchieebenen der Kinder.
4. `ClassID` ist die Klassifikations-ID
5. `CategID` ist Kategorie-ID

Beispiele:

- `classification:editor:-1:children:DocPortal_class_00000001`
- `classification:metadata:0:parents:DocPortal_class_00000001:Unis.Jena`

Bei der Verarbeitung von `include`-Anweisungen in Editor-Definitionen dürfen die folgenden URI-Schemata verwendet werden:

```
classification file http https request resource session webapp mcrobject
```

Beim Aufruf der XSL-Funktion `document()` innerhalb eines Stylesheets können die folgenden URI-Schemata verwendet werden:

```
classification file http https resource session query webapp mcrobject
```

2.7.2 Erweiterung des URI-Resolvers

Unter Umständen kann es nötig sein den URIResolver für eigene Anwendungen zu erweitern. Dabei ist es nicht möglich vorhandene URI-Schemas zu überschreiben, jedoch neue den bereits vorhandenen hinzuzufügen. Für jedes Schema z.B. `file` gibt es einen Resolver, der entsprechende URIs auflösen kann. Dieser Resolver muss die Schnittstelle `MCRURIResolver.MCRResolver` im Paket `org.mycore.common.xml` implementieren. Für die Zuweisung von Schema zur `MCRResolver`-Implementierung ist ein `MCRResolverProvider` verantwortlich, der diese Schnittstelle aus `MCRURIResolver` implementieren muss. Letzterer stellt eine Abbildung von Schema-Strings zu `MCRResolver`-Instanzen zur Verfügung. Der `MCRResolverProvider` kann also beliebig viele `MCRResolver` zu den bereits in MyCoRe integrierten hinzufügen. Eingebunden wird ein zusätzlicher `MCRResolverProvider` mittels folgendem Property:

```
MCR.UriResolver.externalResolver.class = <voller Klassenname>
```

2.8 Das MyCoRe Editor Framework

2.8.1 Funktionalität

Das Metadatenmodell einer MyCoRe Anwendung ist frei konfigurierbar. Dementsprechend benötigt ein MyCoRe System auch einen Online-Editor für diese Metadaten, der frei konfigurierbar ist. Aus dieser Anforderung heraus entstand das MyCoRe Editor Framework, das aus einem XSL Stylesheet und einer Menge von Java-Klassen besteht.

Verschiedene MyCoRe Anwendungen können über XML-Definitionsdateien nahezu beliebige Online-Eingabemasken für Metadaten gestalten. Das Framework verarbeitet diese Editor-Definitionsdateien und generiert daraus den HTML-Code der Webseite, die das Online-Formular enthält. Nach Abschicken des Formulars generiert das Framework aus den Eingaben ein dem Metadatenmodell entsprechendes XML-Dokument, das an ein beliebiges Servlet zur endgültigen Verarbeitung (z. B. zur Speicherung) weitergereicht wird. Ebenso können existierende XML-Dokumente als Eingabe in die Formularfelder des Editors dienen, so dass sich vorhandene Metadaten

bearbeiten lassen. Das Framework regelt dabei die Abbildung zwischen den XML-Elementen und –Attributen und den Eingabefeldern der resultierenden HTML-Formularseite, indem es die in der Editor-Definitionsdatei hinterlegten Abbildungsregeln verarbeitet.

Inzwischen ist das Editor Framework auch in der Lage, einzelne Dateien zusammen mit den Formulareingaben in das System hochzuladen und zur Weiterverarbeitung an ein Servlet durchzureichen. Die Validierung der Eingabefelder ist strukturell vorbereitet, aber derzeit noch nicht implementiert. Prinzipiell erlaubt das Editor Framework, beliebige XML-Dokumente in HTML-Formularen zu erzeugen oder zu bearbeiten.

2.8.2 Architektur

Die folgende Abbildung zeigt die Architektur des MyCoRe Editor Frameworks:

[ToDo]: Bild folgt

Ein HTML Formular, das man mit Hilfe des Frameworks realisieren möchte, wird im folgenden *Editor* genannt. Jeder Editor besitzt eine eindeutige ID (z. B. `document`[7]) und eine Definitionsdatei im XML-Format (`editor-document.xml`). In dieser Definitionsdatei ist festgelegt, aus welchen Eingabefeldern in welcher optischen Anordnung der Editor besteht und wie die Abbildung zwischen den Eingabefeldern und der zugrunde liegenden XML-Darstellung der Daten aussieht.

Ein Editor ist üblicherweise in eine umgebende Webseite eingebunden, die das Formular enthält. Die umgebenden Webseiten referenzieren über die Editor ID den Editor, der an einer bestimmten Stelle der sichtbaren Webseite eingebunden werden soll. Der HTML-Code der Webseite selbst wird in einem MyCoRe System ja aus einem beliebigen XML-Dokument (z.B. anypage.xml[8]) mittels eines dazu passenden XSL Stylesheets (anypage.xsl[9]) generiert. Um in eine solche Webseite einen Editor integrieren zu können, muss `anypage.xml` ein XML-Element enthalten, das auf den einzubindenden Editor verweist , zusätzlich muss `anypage.xsl` das Stylesheet `editor.xsl` einbinden. Das Stylesheet verarbeitet die Referenz auf den Editor und integriert den HTML-Code des Formulars in die umgebende Webseite. So ist es möglich, Editor-Formulare in beliebige Webseiten einzubinden, unabhängig von ihrem Layout oder ihrer Struktur.

```
1  <editor id="document">
2      <include uri="webapp:editor/editor-document.xml"/>
3  </editor>
```

Abbildung 2.9: Einbindung des Editors in eine Webseite

Das Stylesheet `editor.xsl` aus dem Framework verarbeitet die Definition in `editor-document.xml` und erzeugt daraus eine HTML-Tabelle mit den entsprechenden Beschriftungen und Formularfeldern für die Eingabe der Metadaten. Es lassen sich jedoch nicht nur neue Daten eingeben, auch existierende

[7] In den MyCoRe-Applikationen meist eine Mischung aus dem MCRObjectID-Typ und der Bezeichnung eines Verarbeitungsschrittes (editor-commit-document.xml).
[8] z. B. editor_form_...xml in DocPortal
[9] z. B. MyCoReWebPage.xsl in DocPortal

Metadatenobjekte können bearbeitet werden. Das Stylesheet kann dazu von einer beliebigen URL ein XML-Dokument einlesen, verarbeitet dieses dann entsprechend den Abbildungsregeln aus `editor-person.xsl` und füllt die Eingabefelder des Formulars mit den Inhalten der XML-Elemente und –Attribute.

Wenn der Benutzer im Browser die Eingaben in die Formularfelder getätigt hat und das Formular abschickt, werden die Eingaben durch das Servlet `MCREditorServlet` aus dem Framework verarbeitet. Das Servlet generiert aus den Eingaben ein XML-Dokument, entsprechend den Regeln aus `editor-document.xml`. Dieses XML-Dokument wird anschließend an ein beliebiges anderes Servlet weitergereicht, welches in der Formulardefinition angegeben ist und welches dann die Daten endgültig verarbeiten kann und z. B. das XML-Dokument als MyCoRe Metadaten-Objekt speichert.

`MCREditorServlet` übergibt dabei dem Ziel-Servlet (z. B. `AnyTargetServlet`) ein Objekt vom Typ `MCREditorSubmission`, das nicht nur das XML-Dokument, sondern auch eventuell gemeinsam mit den Formulardaten übertragene Dateien als InputStream enthält. Damit ist es möglich, auch einfache Datei-Upload-Formulare über das Editor Framework zu realisieren.

Das Ziel-Servlet erhält zusätzlich auch die ursprünglichen HTTP Request Parameter aus dem abgeschickten Formular, so dass es bei Bedarf auch direkt auf die Werte bestimmter Eingabefelder zugreifen kann und nicht zwingend unbedingt das resultierende XML-Dokument beachten muss. Dadurch ist es möglich, das Editor Framework auch mit bereits existierenden Servlets zu nutzen, die bisher keine XML-Dokumente als Eingabe verarbeiten. Das Framework ist daher auch in der Lage, die Eingaben an eine beliebige URL statt an ein Servlet zu senden, so dass das Ziel der Eingaben an sich auch ein externes CGI-Skript oder ein Servlet in einem entfernten System sein könnte. Das schafft ein hohes Maß an Flexibilität, durch das das Editor Framework auch genutzt werden kann, um z. B. Suchmasken oder Login-Formulare zu erzeugen. Die Nutzung ist nicht allein auf die Realisierung von Metadaten-Editoren beschränkt.

2.8.3 Workarounds

Einige Funktionen sind derzeit noch nicht implementiert, es existieren aber Workarounds dafür:

- Eingabevalidierung:
  Soll: Die Editor-Definitionsdateien werden dafür Regeln enthalten, die nach Abschicken des Formulars durch `MCREditorServlet` überprüft werden.
  Workaround: Die Validierung kann im Ziel-Servlet erfolgen, entweder auf Basis der ursprünglichen Formularfelder oder auf Basis des daraus erzeugten XML-Dokumentes.

- Namespaces:
  Soll: Namespaces können in der Editor Definition deklariert werden, zusammen mit der Abbildung zwischen den Eingabefeldern und der XML-Darstellung der Daten.
  Workaround: Namespaces werden im Ziel-Servlet hinzugefügt, indem das

generierte JDOM-Dokument entsprechend nachbearbeitet wird. Dies könnte auch ein Stylesheet tun.

2.8.4 Beschreibung der Editor-Formular-Definition

Die Grundstruktur

Im Folgenden soll die Gestaltung eines Editor-Formulars näher beschrieben werden. Als Beispiel ist ein Formular für ein Dokument gedacht. Die Daten werden z. B. in der Datei `editor-document.xml` gespeichert.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE editor>
3
4  <editor id="document">
5      ...
6  </editor>
```

Abbildung 2.10: Rahmen der Formular-Definition

Als nächstes definieren wir die Eingabefelder des Editors. Ein Editor besteht aus verschiedenen Komponenten, die innerhalb des Elements `components` deklariert werden. Dies können einfache Komponenten z. B. für Auswahllisten und Texteingabefelder sein, oder aber zusammengesetzte Panels, mit deren Hilfe man einfache Komponenten in einer Tabellengitterstruktur anordnen kann. Panels können wiederum andere Panels beinhalten, so dass sich komplexere Layouts erzeugen lassen. Jede Komponente besitzt eine innerhalb des Editors eindeutige ID, über die sie referenziert werden kann. Jeder Editor besitzt ein Root-Panel, die äußerste Struktur, mit der die Anordnung der Komponenten begonnen wird. Die ID dieses Root-Panels wird als Attribut im Element `components` angegeben.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE editor>
3
4  <editor id="document">
5    <components root="document">
6      <panel id="document" lines="on">
7        <cell row="1" col="1">
8          <text><label xml:lang="de">Titel:</label></text>
9        </cell>
10       <cell row="1" col="2">
11         <textfield width="80" />
12       </cell>
13       <cell row="2" col="1">
14         <text><label xml:lang="de">Autor:</label></text>
15       </cell>
16       <cell row="2" col="2">
17         <textfield width="80" />
18       </cell>
19     </panel>
20   </components>
21 </editor>
```

Abbildung 2.11: Definition eines einfachen Formulares

Innerhalb eines Panels lassen sich andere Komponenten oder Panels anordnen. Solche Komponenten werden im einfachsten Fall einfach in die Gitterzellen eines

Panels eingebettet. Jede Zelle entspricht einem `cell`-Element innerhalb des Panels. Jede Zelle besitzt eine `row`- und `col`-Koordinate, mit deren Hilfe die Zelleninhalte von links nach rechts in Spalten (`col`) und von oben nach unten in Zeilen (`row`) angeordnet werden. Dabei ist für Textfelder noch die Angabe der Sprache möglich.

Innerhalb der Gitterzellen füllen die Komponenten in der Regel nicht den gesamten Platz aus, daher können sie in ihren Zellen mittels des `anchor`-Attributs anhand der Himmelsrichtungen (`NORTH`, `NORTHEAST`, `EAST`, ... bis `CENTER`) angeordnet werden. Wir ordnen die Beschriftungen rechtsbündig, die Texteingabefelder linksbündig an. Außerdem wird nun in der untersten rechten Zelle rechtsbündig ein "Speichern" Button zum Absenden des Formulars ergänzt.

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <!DOCTYPE editor>
3
4  <editor id="document">
5    <components root="document">
6      <panel id="document" lines="on">
7       <cell row="1" col="1" anchor="EAST" ref="form">
8       <cell row="2" col="1" anchor="EAST">
9          <submitButton width="100px" >
10           <label xml:lang="de">[Speichern]</label>
11         </submitButton>
12       </cell>
13     </panel>
14
15     <panel id="form">
16        <cell row="1" col="1" anchor="EAST">
17          <text><label xml:lang="de">Titel:</label></text
18       </cell>
19       <cell row="1" col="2" anchor="WEST">
20          <textfield width="80" />
21       </cell>
22       <cell row="2" col="1" anchor="EAST">
23          <text><label xml:lang="de">Autor:</label></text
24       </cell>
25       <cell row="2" col="2" anchor="WEST">
26          <textfield width="80" />
27       </cell>
28     </panel>
29   </components>
30 </editor>
```

Abbildung 2.13: Auslagern von Definitionen aus dem Root-Panel

Im nächsten Schritt soll nun die Definition aus dem Root-Panel ausgegliedert werden. Sie verwenden dazu ein eigenes Panel. Diese Technik wird verwendet, um Panels mehrfach zu nutzen und den Quellcode des Formulars zu strukturieren. Dabei wird gezeigt, wie sich Panels untereinander aufrufen (Abbildung 2.13).

Soll das Panel `form` nun mehrfach genutzt werden, z. B. in mehreren Formularen, so ist es sinnvoll das Panel in eine gemeinsame Definitionsdatei `imports-common.xml` auszulagern. Im eigentlichen Formular wird dann nur noch auf das `form`-Panel verwiesen. Die Einbindung der ausgelagerten Panels erfolgt mit der `include`-Anweisung (Abbildung 2.14). Die einzufügende Datei (Abbildung 2.15) wird nun im System mittels `EntityResolver` gesucht und eingebunden. Auf diese Weise lassen sich elegante und wartungsarme Formulare gestalten.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE editor>

<editor id="document">
 <components root="document">
   <include uri="webapp:editor/imports-common.xml" />
   <panel id="document" lines="on">
     <cell row="1" col="1" anchor="EAST" ref="form">
     <cell row="2" col="1" anchor="EAST">
        <submitButton width="100px" >
           <label xml:lang="de">[Speichern]</label>
        </submitButton>
      </cell>
   </panel>
 </components>
</editor>
```

Abbildung 2.14: Auslagern von Definitionen aus dem Editor-Formular

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE imports>

<imports>
     <panel id="form">
       <cell row="1" col="1" anchor="EAST">
         <text><label xml:lang="de">Titel:</label></text
       </cell>
       <cell row="1" col="2" anchor="WEST">
         <textfield width="80" />
       </cell>
       <cell row="2" col="1" anchor="EAST">
         <text><label xml:lang="de">Autor:</label></text
       </cell>
       <cell row="2" col="2" anchor="WEST">
         <textfield width="80" />
       </cell>
    </panel>
 </imports>
```

Abbildung 2.15: Die imports-Formular-Definition

Abbildung zwischen Eingabefeldern und XML-Strukturen

Als nächstes definieren wir, wie die Eingabefelder auf XML-Elemente abgebildet werden. Dies geschieht über das Attribut `var` verschiedener Elemente der Editor-Definition. Zunächst wird im var–Attribut des components–Elements der Name des Root-Elements der XML-Darstellung (document) festgelegt. Die Abbildung zwischen Eingabefeldern oder Panels und ihrer Zuordnung zu XML-Elementen geschieht über das var–Attribut in den cell-Elementen, welche die Eingabekomponente enthalten (Abbildung 2.16).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE editor>

<editor id="document">
 <components root="document" var="/document">
   <include uri="webapp:editor/imports-common.xml" />
   <panel id="document" lines="on">
     <cell row="1" col="1" anchor="EAST" ref="form">
     <cell row="2" col="1" anchor="EAST">
       <submitButton width="100px" >
         <label xml:lang="de">[Speichern]</label>
       </submitButton>
      </cell>
   </panel>
 </components>
</editor>
```

Abbildung 2.16: Einfügen des XML-Main-Tag-Namen

Es gibt für Tags und Attribute innerhalb der Root-Komponente verschiedene Möglichkeiten der Gestaltung des Ausgabe-XML-Baumes. Die einfachste Art ist die direkte Zuordnung. Hier werden die Werte des Eingangs-XML-Files dem Feld in der Editormaske zugeordnet. Weiterhin kann dem Feld noch ein Standardwert durch das Attribut `default` mitgegeben werden. Weiterhin können Felder, welche von der Dateneingabe ausgenommen werden sollen, deren Inhalte aber an die Ausgabeseite durchgereicht werden sollen, als `hidden`-Felder mitgeführt werden. Auch hier

können Standardwerte angegeben werden, welche eingesetzt werden, wenn keine Daten für das Tag/Attribut vorhanden sind.

```
1      <cell row="1" col="2" anchor="WEST">
2        <textfield width="80" var="dc/title" default="Mein Buch" />
3      </cell>
4      <hidden var="dc/title/@lang" default="de" />
```

Abbildung 2.17: Integration von einfachen XML-Tags

Sollten Sie mehrere gleiche XML-Strukturen im Editor zur Bearbeitung anbieten wollen, so wird dies über eine Integer-Zahl in Klammer organisiert. Achtung, es werden immer nur soviel XML-Elemente übernommen und an die Ausgabe weitergereicht, wie angegeben sind. Um eine beliebige Anzahl zu präsentieren und zu bearbeiten nutzen sie die weiter unten angegeben Möglichkeit des Repeaters. Für `hidden`-Felder gibt es noch die Variante, mittels des `descendants="true"` - Attributs alle Kindelemente, Attribute und Wiederholungen des Elementes an die Ausgabe durchzureichen.

```
1      <cell row="1" col="2" anchor="WEST">
2        <textfield width="80" var="dc/title" default="Mein Buch" />
3      </cell>
4      <hidden var="dc/title/@lang" default="de" />
5      <cell row="2" col="2" anchor="WEST">
6        <textfield width="80" var="dc/title[2]" default="" />
7      </cell>
8      <hidden var="dc/title[2]/@lang" default="de" />
9      <hidden var="dc/source" descendants="true" />
```

Abbildung 2.18: Integration von mehrfachen XML-Tags

Integration und Aufruf des Editor Framework

Bereits am Anfang des Kapitels wurde eine einfache Syntax zum Aufruf des Editor Framework vorgestellt. Dieser Abschnitt soll nun umfassend zu diesem Thema informieren.

```
1 <editor id="document">
2     <include uri="webapp:editor/editor-document.xml"/>
3 </editor>
```

Abbildung 2.19: Einbindung des Editors in eine Webseite

Das Editor Framework ist so konzipiert, dass es als XML-Sequenz in einer durch das `MCRLayoutServlet` darzustellenden Webseite integriert werden kann. D.h. Sie erstellen eine XML Seite, welche durch das Servlet und die zugehörigen XSLT-Stylesheets nach HTML konvertiert wird. Durch den Import (`include`) der Editor-Framework-XSL-Dateien in das Layout zum Erzeugen der HTML-Seiten ermöglichen Sie die Einbindung ihrer Editor-Formulare. In der oben gezeigten XML-Sequenz wird jedoch nur die Definitionsdatei des Formulars beschrieben: „nimm aus der Web-Applikation die Datei editor-document.xml aus dem Verzeichnis editor". Dabei ist noch nichts über die Datenquelle bekannt. Hierfür gibt es noch einige Parameter, welche durch das Framework ausgewertet werden. Diese können sowohl direkt im Browser in der Aufrufsequenz oder über Parametereinstellungen in einem Servlet mitgegeben werden.

```
http://.../document.xml?XSL.editor.source.new=true&type=...
```

Oder in einem Servlet als Codestück, wie Abbildung 2.20 zeigt.

```
1  String base = getBaseURL() + myfile;
2  Properties params = new Properties();
3  params.put( "XSL.editor.source.new", "true" );
4  params.put( "XSL.editor.cancel.url", getBaseURL()+cancelpage);
5  params.put( "type",mytype );
6  params.put( "step",mystep );
7  job.getResponse().sendRedirect(job.getResponse().encodeRedirectURL(
8   (buildRedirectURL( base, params )));
```

Abbildung 2.20: Codesequenz zum Aufruf des Editor Framework

Über XSL-Parameter kann beim Aufruf des Editors gesteuert werden, ob bzw. aus welcher Quelle die zu bearbeitenden XML-Daten gelesen werden. Die zu bearbeitenden XML-Daten werden von einer URL gelesen. Falls die URL mit `http://` oder `https://` beginnt, wird sie als absolute URL direkt verwendet, andernfalls wird

die URL als relativ zur `WebApplicationBaseURL` (context root), dem Startpunkt der WebApplication, interpretiert. Es gibt die folgenden vier Optionen:

1. `XSL.editor.source.new=true`
   Es wird das Formular mit leeren Datenfeldern gestartet.
2. Statische URL eines XML-Dokumentes (z. B. eines Metadaten-Templates), die in der Editor-Definition als Top-Level-Element angegeben ist (d.h. auf gleicher Ebene wie die target-Deklaration), z. B.
   `<source url="templates/document.xml" />`
3. URL, die durch ein Servlet oder Stylesheet zur Laufzeit gebildet wird, und die beim Aufruf des Editors über einen XSL Parameter übergeben wird:
   `XSL.editor.source.url=templates/document.xml`
4. URL, die aus einer Kombination einer statischen URL und eines ID-Tokens gebildet wird, das beim Aufruf des Editors über einen XSL Parameter übergeben wird. Im folgenden Beispiel wird zur Laufzeit das Token XXX durch den Parameterwert 4711 ersetzt:
   `XSL.editor.source.id=4711`
   `<source url="servlets/SomeServlet?id=XXX" token="XXX" />`

Start-Parameter für die Definition des Abbrechen-Buttons:

`XSL.editor.cancel.url / XSL.editor.cancel.id`
Diese Parameter werden im Abschnitt zur Verwendung eines Cancel-Buttons (Abbrechen-Knopf) erläutert, sie steuern die URL, zu der bei Klick auf einen „Abbrechen" Button zurückgekehrt wird.

Start-Parameter für die Weitergabe an das Zielservlet:

In der Regel werden die Eingaben nach der Bearbeitung mit dem Editor an ein Zielservlet gesendet (`target type="servlet"`). Man kann diesem Zielservlet auch HTTP Parameter mitschicken, die der Editor bei Abschicken des Formulars mitsendet. So können Informationen an das Zielservlet weitergegeben werden, die nicht Teil des zu bearbeitenden XML sind, z. B. die Information, ob es sich um einen Update- oder Create-Vorgang handelt, die Objekt-ID etc. Der Editor reicht automatisch alle HTTP Request Parameter an das Zielservlet durch, deren Name nicht mit XSL beginnt.

Beispiel:
```
http://.../editor-document.xml?XSL.editor.source.id=4711&
action=update&mode=bingoBongo
```

Ausgabeziele

Das Editor-Framework ist auch hinsichtlich der Verarbeitung der entstandenen Daten flexibel konfigurierbar. Wie bereits erwähnt, wird nach dem Betätigen des Submit-Buttons das `MCREditorServlet` für eine erste Verarbeitung angestoßen. Diese reicht die Daten dann an ein weiteres konfigurierbares Ziel weiter. Welches das ist, wird mit der target-Zeile in der Formulardefinition festgelegt. Das Attribut `method` definiert, ob dazu `HTTP GET` oder `POST` verwendet wird. Zu empfehlen ist in der Regel immer `POST`, was auch der Default-Wert ist und zwingend bei File Uploads gesetzt wird. Möglich ist:

- die Ausgabe als HTML-Seite zum Debugging des Ergebnisses:
  `<target type="debug" method="post" format="xml" />`,

- die Weiterleitung der Daten in ein nachgeschaltetes Servlet im XML-Format:
  `<target type="servlet" method="post" name="MCRCheckDataServlet" format="xml" />`,
- die Weiterleitung der Daten an ein nachgeschaltetes Servlet in der Form `name=value`, d. h. ohne Generierung eines XML-Dokumentes:
  `<target type="servlet" method="post" name="MCRCheckDataServlet" format="name=value" />`,
- die Weiterleitung an das LayoutServlet zur Anzeige des generierten XML-Dokumentes als Test:
  `<target type="display" method="post" format="xml" />`.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE editor>

<editor id="document">
    <!--
    <target type="debug" />
    -->
    <target type="servlet" name="MCRCheckDataServlet" method="post" format="xml" />
    ...
</editor>
```

Abbildung 2.21: Rahmen der Formular-Definition

- die Weiterleitung an eine beliebige URL als HTTP GET oder POST Request:
  `<target type="url" method="post" format="name=value" url="cgi-bin/ProcessMe.cgi" />` oder
- die Weiterleitung an einen anderen Editor, für den dieser Editor als „Subdialog" fungiert. Dies wird in einem separaten Abschnitt erläutert.
  `<target type="subselect" method="post" format="xml" />`

Mit der nachfolgenden Java-Code-Sequenz kann nun auf die vom MCREditorServlet durchgereichten XML- und -File-Daten zugegriffen werden.

```
/**
 * This method overrides doGetPost of MCRServlet.<br />
 */
public void doGetPost(MCRServletJob job) throws Exception
  {
  // read the XML data
  MCREditorSubmission sub = (MCREditorSubmission)
    (job.getRequest().getAttribute("MCREditorSubmission"));
  org.jdom.Document indoc = sub.getXML();
  List files = sub.getFiles();

...
  }
```

Abbildung 2.22: Java-Code-Sequenz für den Zugriff

Neben der Übernahme der Ausgabewerte des `MCREditorServlets` als XML-Baum können diese auch in der Form '`name=value`' durchgereicht werden. Hierzu müssen u. a. die `var`-Attribute entsprechend in eine einfache Form gebracht werden. Der Ausschnitt der Editor-Definition soll das verdeutlichen.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE editor>

<editor id="document">
    <target type="servlet" name="MCRStartEditorServlet" method="get"  format="name=value" />
    <components root="formular">
        <panel id="document" lines="on">
            <cell row="1" col="1" anchor="EAST">
                <text><label xml:lang="de">Label:</label></text
            </cell>
            <cell row="1" col="2" anchor="WEST" var="label">
                <textfield width="80" />
            </cell>
            ...
        </panel>
        ...
    </components>
</editor>
```

Abbildung 2.23: Rahmen der Formular-Definition mit name=value

2.8.5 Syntax der Formularelemente

Aufbau einer Zelle

Das `cell`-Element stellt ein elementares Gebilde innerhalb eines Formulargitters dar. Seine Position wird durch die Attribute `col`, `row` und `anchor` beschrieben. `col` gibt dabei die relative Spaltenzahl, `row` die relative Zeilenzahl an. Mit `anchor` kann die Ausrichtung gesteuert werden. Mögliche Werte sind hier `NORTHWEST`, `NORTH`, `NORTHEAST`, `WEST`, `CENTER`, `EAST`, `SOUTHWEST`, `SOUTH` und `SOUTHEAST`. Mit dem Attribut `ref` kann auf ein mehrfach verwendbares Element via ID referenziert werden. Über das Attribut `sortnr` kann die Reihenfolge der zu generierenden XML-Daten für die Ausgabe geregelt werden. Dieses Attribut bezieht sich dabei auf das gesamte Formular.

```
<cell var="..." col="..." row="..." anchor="..." ref="..." sortnr="..." />

oder

<cell var="..." col="..." row="..." anchor="..." sortnr="...">
    [Formularfelddefinition]
</cell>
```

Abbildung 2.24: Syntax des cell-Elements

Texteingabefelder

Ein einzeiliges Texteingabefeld wird mittels eines Elementes `textfield` erzeugt, ein mehrzeiliges mittels eines Elementes `textarea`. Das Attribut `width` gibt die Breite des Texteingabefeldes in Anzahl Zeichen an. Das Attribute `height` gibt für mehrzeilige Texteingabefelder die Anzahl Zeilen an. Das Attribut `maxlenght` gibt bei textfield-Elementen die maximale Länge der Eingabe an.

Texteingabefelder können einen Vorgabewert enthalten, der wahlweise über ein Attribut oder ein Kindelement mit dem Namen `default` angegeben wird. Die Verwendung eines default-Elementes statt eines default-Attributes ist sinnvoll, wenn mehrzeilige default-Werte angegeben werden sollen.

```
<textfield id="tf.name" width="80" default="Standardtext" maxlength="200" / >
<textarea id="tf.anschrift" width="80" height="5">
    <default>
        Musterstrasse 69
        4711 Musterstadt
    </default>
</textarea>
```

Abbildung 2.25: Syntax von textfield und textarea

Alternativ kann auch ein AutoFill-Wert über das Attribut bzw. Element `autofill` angegeben werden. Während default-Werte immer Teil der Eingabe werden, werden AutoFill-Werte nur dann als Eingabe betrachtet, wenn der Nutzer sie ergänzt oder ändert. Unveränderte AutoFill-Werte werden also ignoriert. Für alle Mitarbeiter der Universität sind z. B. Teile der Email-Adresse oder der Anschrift in den meisten Fällen identisch. Falls der Nutzer diese Angaben im Eingabefeld ändert oder ergänzt, wird dies als Eingabe betrachtet. Ansonsten wird der AutoFill-Wert ignoriert. Default-Werte dagegen werden immer als Eingabe betrachtet. Wichtig ist, dass textfield- und textarea-Elemente ein innerhalb des Editors eindeutiges ID-Attribut besitzen.

```
<textfield id="tf.email" width="80" autofill="@uni-duisburg-essen.de" />
<textarea id="ta.anschrift" width="80" height="5">
    <autofill>
        Universitätsstr. 9-11
        45141 Essen
    </autofill>
</textarea>
```

Abbildung 2.26: AutoFill-Werte in textfield und textarea

Passwort-Eingabefelder

Passwort-Eingabefelder werden über das Element `password` erzeugt. Sie können keine default- oder autofill-Werte besitzen. Bei der Eingabe werden Sternchen statt der eingegebenen Zeichen angezeigt. Das Attribut `width` gibt die Breite des Eingabefeldes in Anzahl Zeichen an. Leider ist es bei allen Browsern so, dass dieses Feld keine vorhandenen Inhalte bearbeiten kann, d. h. Passworte sind aus Sicherheitsgründen immer neu einzugeben.

```
1  <password width="10" />
```

Abbildung 2.27: Syntax des password-Elements

Einfache Checkboxen

Das Element `checkbox` generiert eine alleinstehende Checkbox, bei der über einen „Haken" ein Wert ein- oder ausgeschaltet werden kann. Das Attribut `value` gibt den Wert an, der gesetzt werden soll, wenn die Checkbox markiert ist. Das Attribut `checked` gibt mit den möglichen Werten `true` und `false` an, ob als default die Checkbox markiert sein soll oder nicht. Die Beschriftung der Checkbox erfolgt über ein Attribut `label` bzw. über untergeordnete mehrsprachige label-Elemente, vgl. dazu den Abschnitt zu Beschriftungen.

```
<cell ... var="@raucher">
    <checkbox value="true" checked="false" label="Ja, ich bin Raucher" />
</cell>
```

Abbildung 2.28: Syntax für Checkboxen

Auswahllisten

Auswahllisten werden über das Element `list` erzeugt. Die in der Liste darzustellenden Werte werden über geschachtelte `item`-Elemente angegeben. Das Attribut `type` gibt an, in welchem Format die Listenwerte dargestellt werden. Der Typ `dropdown` stellt die Listenwerte als einzeilige Dropdown-Auswahlliste dar. Das Attribut `width` gibt dabei die Breite des Auswahlfeldes in CSS-Syntax an, d.h. es sind Angaben in Anzahl Zeichen oder Pixeln etc. möglich. Der Typ `multirow` stellt die Listenwerte als mehrzeiliges Auswahlfeld dar. Das Attribut `rows` gibt dabei die Anzahl Zeilen an, die maximal zu sehen sind. Enthält die Liste mehr Werte als Zeilen, werden Scrollbalken dargestellt. Das Attribut `multiple` mit den möglichen Werten `true` und `false` gibt an, ob eine Mehrfachauswahl möglich ist. Das Attribut `default` gibt ggf. eine Vorauswahl vor.

```
<list type="dropdown" width="100px" default="2" />
    <item value="1"><label xml:lang="de">eins</label></item>
    <item value="2"><label xml:lang="de">zwei</label></item>
    <item value="3"><label xml:lang="de">drei</label></item>
</list>
<list type="multirow" width="100px" rows="3" multiple="true">
    <item value="a"><label xml:lang="de">Fred</label></item>
    <item value="b"><label xml:lang="de">Willi</label></item>
    <item value="c"><label xml:lang="de">Otto</label></item>
</list>
```

Abbildung 2.29: Syntax einer Auswahlliste

Die in einer Liste dargestellten Werte werden über `item`-Elemente angegeben. Diese können auch mehrstufig geschachtelt sein, um eine hierarchische Struktur darzustellen (d.h. item-Elemente können wiederum item-Elemente enthalten). Eine Schachtelung ist nur für die Listentypen `dropdown` und `multirow` erlaubt und sinnvoll. Die Hierarchie wird dabei automatisch durch Einrückungen dargestellt. Die Beschriftung der item-Elemente wird über geschachtelte label-Attribute oder Elemente definiert und kann mehrsprachig angegeben werden, vgl. dazu den folgenden Abschnitt zu Beschriftungen. Die Verwendung von `<list type="multirow" multiple="true">`, d. h. mehrzeiliger Auswahllisten mit Mehrfachauswahl ist nur sinnvoll, wenn das aktuelle var-Attribut der Zelle auf ein Element verweist, denn Attribute sind ja grundsätzlich in XML nicht wiederholbar.

```
<cell ... var="programmiersprache">
    <list type="multirow" multiple="true" rows="3">
        <item label="Java" value="java" />
        <item label="C++" value="cpp" />
        <item label="Pascal" value="pas" />
    </list>
</cell>
```

Abbildung 2.30: Beispiel für Mehrfachauswahl

Das Beispiel erzeugt eine dreizeilige Mehrfachauswahllist, die ggf. in mehrfach generierte XML-Elemente „Programmiersprache" resultiert.

Radio-Buttons und Checkbox-Listen

Der Typ `radio` stellt die Listenwerte als eine Menge von Radiobuttons dar, es kann also nur ein Wert ausgewählt werden. Der Typ `checkbox` stellt die Listenwerte als eine Menge von Checkboxes dar, in diesem Fall können mehrere Werte ausgewählt sein. Bei diesen beiden Typen werden alle Werte der Liste nacheinander in einer Tabellenstruktur mit einer gewissen Anzahl an Zeilen und Spalten ausgegeben. Es kann entweder die Anzahl an Zeilen (Attribut `rows`) oder die Anzahl an Spalten (Attribut `cols`) dieser Tabelle angegeben werden. Der jeweils nicht angegebene Wert folgt automatisch aus der Anzahl der Werte in der Liste. Der Spezialfall `rows="1"` entspricht also einer Anordnung aller Listenwerte von links nach rechts in einer Zeile, der Fall `cols="1"` entspricht einer Anordung von oben nach unten in nur einer Spalte.

```
<list type="radio" rows="2" default="new">
    <item value="1"><label xml:lang="de">eins</label></item>
    <item value="2"><label xml:lang="de">zwei</label></item>
    <item value="3"><label xml:lang="de">drei</label></item>
</list>
<list type="checkbox" cols="3">
    <item value="a"><label xml:lang="de">Fred</label></item>
    <item value="b"><label xml:lang="de">Willi</label></item>
    <item value="c"><label xml:lang="de">Otto</label></item>
</list>
```

Abbildung 2.31: Syntax von Radio-Button und Checkbox-Listen

Analog zu multirow/multiple Listen ist die Verwendung von Checkbox-Listen nur sinnvoll für die Abbildung auf XML-Elemente, da XML-Attribute nicht wiederholbar sind (vgl. vorangehenden Abschnitt).

Beschriftungen

Beschriftungen von Eingabefeldern lassen sich mit den Elementen `text` und `label` definieren. Das Element `text` erzeugt eine frei platzierbare Beschriftung etwa für eine Texteingabefeld. Beschriftungen dürfen auch XHTML Fragmente enthalten, etwa um eine fette, kursive oder mehrzeilige Darstellung zu erreichen. Mehrsprachige Beschriftungen von Elementen können über mehrere `label`-Elemente realisierte werden, wobei das Attribut `xml:lang` die Sprache angibt.

Die aktive Sprache wird wie bei anderen MyCoRe-Stylesheets über den `XSL.Lang` Parameter definiert. Wenn ein `label` existiert, bei dem `xml:lang` der aktiven Sprache entspricht, wird dieses ausgegeben. Ist dies nicht der Fall, wird als erster Rückfallschritt das `label` der Default-Sprache (Parameter `XSL.DefaultLang`) ausgegeben. Existiert auch dies nicht, wird das Label ausgegeben, das im `label`-Attribut oder im ersten `label`-Element enthalten ist, das kein `xml:lang`-Attribut enthält. In alles anderen Fällen wird das erste `label`-Element verwendet, das überhaupt existiert.

```
<text>
    <label xml:lang="de"><b>Eins</b></label>
    <label xml:lang="en"><b>One</b></label>
</text>
```

Abbildung 2.32: Syntax eines Textfeldes

Internationalisierung

Das Editor-Framework verfügt über mehrere Möglichkeiten der Mehrsprachigkeit einzelner Auswahl- oder Textfelder. Bis MyCoRe-Version 1.2 war hier nur der Einsatz von `<label xml:lang="...">`-Tags (wie oben beschrieben) möglich. Mit Version 1.3 ist die direkte Nutzung von I18N[10] möglich. Um eine Abwärts-Kompatibilität zu erreichen sind die `<label>`-Tags auch weiterhin im Framework gültig. Es ist aber zu empfehlen, langfristig ältere MyCoRe-Anwendungen entsprechend umzustellen. damit wird eine strikte Trennung von Layout und sprachabhängigem Text erreicht.

Mit MyCoRe v1.3 wurde das `i18n`-Attribut in alle Editor-Framework-Stellen eingebaut, in den bisher `<label>`-Tags vorgesehen waren. Entsprechend der I18N-Spezifikation sind alle Texte nun über eine Property-Variable bekannt zu machen und in sprachabhängigen Dateien anzulegen. Diese tragen die Namen `messages_{lang}.properties` und sind bei MyCoRe im `config`-Verzeichnis untergebracht. Sollte das System für die angegebene Sprache keine solche Datei finden, so wird als Fallback die Datei `messages.properties` gesucht. Sind beide Angaben vorhanden, hat das `i18n`-Attribut Vorrang. Die nachfolgende Abbildung verdeutlicht die neue Funktion.

```
<text i18n="some.key.from.messages.file" />
```

Abbildung 2.33: Beschriftung mit I18N

Abstandshalter

Ein Abstandshalter erzeugt leeren Raum als Platzhalter zwischen Elementen. Die Attribute `width` und `height` geben Breite und Höhe des Abstandes in CSS-Syntax, etwa in Anzahl Pixeln, an.

[10] http://www.debian.org/doc/manuals/intro-i18n/

```
1  <space width="0px" height="50px" />
```

Abbildung 2.34: Syntax eines Abstandhalters

Externes Popup-Fenster

Ein externes Popup-Fenster kann z. B. ein Hilfetext zu einem Eingabefeld im HTML-Format enthalten. Es können aber auch beliebige andere Informationen und Funktionalitäten wie Eingabehilfen dargestellt werden. Der Startpunkt ist ein Button im Formular, der mit einer frei wählbaren Beschriftung versehen werden kann. Standardmäßig wird er als [?] Button im Formular dargestellt. Bei Anklicken des Buttons erscheint das codierte HTML in einem Popup-Fenster. Die Attribute `width` und `height` steuern die Breite und Höhe dieses Fensters. Das Attribut `css` gestattet die Verwendung eines externen CSS-Files. Wichtig ist, dass jedes `helpPopup` Element eine eindeutige ID besitzt. Optional kann das Popup-Fenster auch von einer externen URL geladen werden. Falls die URL mit „`http://`" oder „`https://`" beginnt, wird sie als absolute URL direkt verwendet, andernfalls wird die URL als relativ zur `WebApplicationBaseURL` (`context root`), dem Startpunkt der WebApplication, interpretiert.

Innerhalb des helpPopup-Tags gibt es mehrere Tags, welche das Aussehen und den Inhalt des Aufruf-Buttons und des Popup-Fensters bestimmen. Jedes Tag verfügt über das xml:lang-Attribut. Somit kann jedes Tag für die im Projekt benötigten Sprachen implementiert werden. Zusätzlich gibt es das Konstrukt `xml:lang="all"`, das anzeigt, dass das entsprechende Tag immer unabhängig von der Spracheinstellung des Systems präsentiert werden soll.

Im Einzelnen sind die Tags wie folgend definiert:

- `button` gibt die Zeichenkette an, welche der Start-Button haben soll,
- `title` gibt den Titel des Popup-Fensters an,
- `close` gibt den Text zum Schließen des Popup-Fensters an,
- `label` beinhaltet den eigentlichen darzustellenden HTML-Code oder Text.

```
<helpPopup id="url.help" width="400px" height="200px" css="/mycss.css">
    <button xml:lang="all">!</button>
    <title xml:lang="de">Hilfetext</title>
    <close xml:lang="de">Beenden</close>
    <label xml:lang="de">
        In diesem Feld geben Sie bitte die <b>WWW Adresse (URL)</b>
        der Webseite ein. Bitte achten Sie darauf, dass die Adresse mit
        <i>http://</i> beginnt, da sonst der Link unter Umständen nicht
        funktioniert.
    </label>
</helpPopup>

<helpPopup id="url.help" width="400px" height="200px" url="hilfe/url.html" />
```

Abbildung 2.35: Syntax eines Popup-Fensters

Buttons

Über das Element `button` wird ein Knopf erzeugt, der bei Anklicken zu einer externen URL wechselt. Das Attibut `width` legt die Breite des Knopfes fest, das Attribut `label` oder ggf. mehrsprachige enthaltene label-Elemente legen die Beschriftung des Knopfes fest. Falls die URL mit „`http://`" oder „`https://`" beginnt, wird sie als absolute URL direkt verwendet, andernfalls wird die URL als relativ zur `WebApplicationBaseURL` (`context root`), dem Startpunkt der WebApplication, interpretiert.

```
1  <button width="100px" url="index.html">
2      <label xml:lang="de">Start</label>
3  </button>
```

Abbildung 2.36: Syntax eines einfachen Buttons

SubmitButton

Über das Element `submitButton` wird ein Knopf erzeugt, der beim Anklicken die Eingaben an das in der Konfiguration angegebene Ziel sendet. Das Attibut `width` legt die Breite des Knopfes fest, das Attribut `label` oder ggf. mehrsprachige enthaltene label-Elemente legen die Beschriftung des Knopfes fest.

```
<submitButton width="100px">
    <label xml:lang="de">Start</label>
</submitButton>
```

Abbildung 2.37: Syntax des Submit-Buttons

CancelButton

Über das Element `cancelButton` wird ein Knopf erzeugt, der bei Anklicken die Bearbeitung des Formulars abbricht. Das Attibut `width` legt die Breite des Knopfes fest, das Attribut `label` oder ggf. mehrsprachige enthaltene label-Elemente legen die Beschriftung des Knopfes fest.

```
<cancelButton width="100px">
    <label xml:lang="de">Start</label>
</cancelButton>
```

Abbildung 2.38: Syntax des Cancel-Buttons

Die Ziel-URL des Buttons kann auf drei verschiedene Weisen gebildet werden. Falls die URL mit „`http://`" oder „`https://`" beginnt, wird sie als absolute URL direkt verwendet, andernfalls wird die URL als relativ zur `WebApplicationBaseURL` (`context root`), dem Startpunkt der WebApplication, interpretiert.

Statische URL, die in der Editor-Definition als Top-Level-Element angegeben ist (d.h. auf gleicher Ebene wie die target-Deklaration), z. B.
`<cancel url="pages/goodbye.html" />`

1. URL, die durch ein Servlet oder Stylesheet zur Laufzeit gebildet wird, und die beim Aufruf des Editors über einen XSL-Parameter übergeben wird:
   `XSL.editor.cancel.url=pages/goodbye.html`
2. URL, die aus einer Kombination einer statischen URL und eines ID=Tokens gebildet wird, das beim Aufruf des Editors über einen XSL Parameter übergeben wird. Im folgenden Beispiel wird zur Laufzeit das Token XXX durch den Parameterwert 4711 ersetzt:
   `XSL.editor.cancel.id=4711`
   `<cancel url="servlets/SomeServlet?id=XXX" token="XXX" />`

Ausgabe von Werten aus dem Quelldokument

Das Element `output` kann bei der Bearbeitung einer existierenden XML-Quelle verwendet werden, um den Inhalt von Attributen oder Elementen im Formular als nicht bearbeitbaren Text auszugeben. Dabei ist zu beachten, dass der ausgegebene Wert bei Abschicken des Formulars nicht weitergegeben wird. Hierfür muss ggf. ein `hidden`-Element verwendet werden. Das Attribut `default` gibt den Wert an, der ggf. ausgegeben wird, wenn das Dokument keinen Wert enthält.

Beispiel:

```
<text><label>Dokument-ID:</label></text>
<output var="@id" default="(neu)" />
```

gibt den Wert des Attributes `id` aus, sofern dieses im XML-Quelldokument existiert, ansonsten wird „(neu)" ausgegeben.

Wiederholbare Elemente mit Repeatern erstellen

Häufig sind einzelne Eingabefelder oder ganz Panels wiederholbar. Das Element `repeater` schachtelt auf einfache Weise ein Eingabeelement oder ein ganzes Panel und macht dieses wiederholbar. Das var-Attribut der umgebenden Zelle muss auf ein Element verweisen, da Attribute nicht wiederholbar sind. Das Attribut `min` gibt an, wie oft minimal das wiederholte Element im Formular dargestellt wird, das Attribut `max` gibt die Maximalzahl an Wiederholungen an. In jedem Fall wird das Element minimal so oft dargestellt, wie es in der Eingabe auftritt.

```
<cell ... var="creators/creator">
    <repeater min="1" max="10">
        <textfield id="tf.creator" width="80" />
    </repeater>
</cell>
wiederholt ein einzelnes Eingabefeld oder eine beliebige andere Komponente.

<cell ... var="creators/creator">
    <repeater min="3" max="6">
        <panel ...>
        </panel>
    </repeater>
</cell>
wiederholt ein ganzes Panel, dass wie hier im Beispiel auch direkt eingebettet definiert
werden kann.

<cell ... var="creators/creator">
    <repeater min="3" max="6" ref="pcreator" />
</cell>
```

Abbildung 2.39: Beispiel Repeator

Das Beispiel wiederholt die Komponente mit der ID `pcreator`. Wiederholbare Elemente werden mit +/- Buttons dargestellt, mit denen neue Eingabefelder hinzugefügt bzw. dargestellte gelöscht werden können. Bei mehr als einer aktuell dargestellten Wiederholung werden Pfeile angezeigt, mit deren Hilfe die Reihenfolge der Elemente vertauscht werden kann.

Der Framework-interne FileUpload

Das Element `file` erlaubt es, eine einzelne Datei zusammen mit den Formulareingaben hochzuladen oder eine auf dem Server vorhandene Datei zu löschen oder zu aktualisieren. Vorausetzung ist hierfür die vollständige Integration des Upload-Services in den jeweiligen Java-Klassen. Das Attribut `maxlenght` gibt optional die maximale Grösse der hochzuladenen Datei an. Der FileUpload über ein HTML-Formular ist nur für relativ kleine Dateien mit wenigen MB zu empfehlen, andernfalls ist der in MyCoRe implementierte externe FileUpload zu nutzen. Das Attribut `accept` gibt optional den akzeptierten MIME Typ an. Ob diese Angaben beachtet werden, ist jedoch vom Browser abhängig und daher nicht verlässlich. Die Darstellung eines Datei-Upload Feldes hängt ebenfalls sehr stark vom Browser ab und kann nicht über CSS gestaltet werden. In der Regel bestellt ein solches Feld aus einem Texteingabefeld (dessen Breite in Anzahl Zeichen das `width`-Attribut angibt) und einem Button „Durchsuchen“. Beschriftung und Layout dieser Elemente sind nicht steuerbar und fest vom Browser vorgegeben, CSS Angaben funktionieren leider nicht zuverlässig. Über den Durchsuchen-Button kann eine Datei von der lokalen

Platte gewählt werden, alternativ kann der Dateipfad im Texteingabefeld eingegeben werden. Mit Abschicken des Formulars wird der Dateiinhalt und der Dateiname an den Server übertragen, was unter Umständen lange dauert.

```
<cell row="1" col="1" anchor="WEST" var="pathes/path">
    <file width="60" maxlength="5000000" />
</cell>
```

Abbildung 2.40: Syntax des FileUpload

Zu beachten ist noch, dass für den FileUpload einige Property-Werte des Systems von Bedeutung sind.

- Dateien bis zu dieser Größe werden im Hauptspeicher des Servers gehalten: `MCR.Editor.FileUpload.MemoryThreshold=1000000`
- Dateien bis zu dieser Größe werden für HTTP Uploads akzeptiert: `MCR.Editor.FileUpload.MaxSize=5000000`
- Temporärer Speicherort für hochgeladene Dateien: `MCR.Editor.FileUpload.TempStoragePath=/tmp/upload`

Das Auslesen der Dateien zur weiteren Verarbeitung kann z. B. wie folgt durchgeführt werden.

```
import org.apache.commons.fileupload.*;
...
public void doGetPost(MCRServletJob job) throws Exception
{
  MCREditorSubmission sub = (MCREditorSubmission)
    (job.getRequest().getAttribute("MCREditorSubmission"));
  List files = sub.getFiles();
  for( int i=0; i<files.size(); i++)
  {
    FileItem item = (FileItem)( files.get(i) );
    String fname = item.getName().trim();
...
    File fout = new File(dirname,fname);
    FileOutputStream fouts = new FileOutputStream(fout);
    MCRUtils.copyStream( item.getInputStream(), fouts );
    fouts.close();
  }
...
  }
```

Abbildung 2.41: Java-Code zum Lesen des FileUpload

Weitere Hinweise zum Auslesen der hochgeladenen Dateien finden Sie in der JavaDoc-Dokumentation der Klassen `org.mycore.frontend.editor2.MCREditorSubmission` und `MCREditorVariable` sowie in der Online-Dokumentation des Apache File Upload Paketes (http://jakarta.apache.org/commons/fileupload/).

Die Methode `MCREditorSubmission.listFiles()` liefert eine `java.util.List` von hochgeladenen FileItem-Objekten. Die Apache-Klasse `FileItem` besitzt Methoden `getName()` und `getInputStream()`, die den Dateinamen und den hochgeladenen Dateiinhalt liefern. Die Methode `getFieldName()` liefert den Pfad im XML-Dokument, zu dem die hochgeladene Datei gehört.

Alternativ kann auch über diesen Pfad auf eine bestimmte hochgeladene Datei direkt zugegriffen werden: Die Eingaben aus dem Editor werden als JDOM-Dokument an

das Zielservlet übergeben und stehen über `MCREditorSubmission.getXML()` zur Verfügung. Mit JDOM-Operationen kann man nun zu dem JDOM-Element oder -Attribut navigieren, für das ggf. eine Datei im Formular hochgeladen wurde. Die Methode `MCREditorSubmission.getFile()` erwartet als Argument dieses JDOM-Objekt und liefert das dazu gehörende FileItem.

FileItem-Objekte sollten unmittelbar verarbeitet werden, da die Dateiinhalte nur temporär gespeichert sind. Der hochgeladene Inhalt kann mittels `FileItem.write()` in ein persistentes `java.io.File` kopiert werden oder in einem `MCRFile` gespeichert werden.

Zu beachten ist, dass FileUploads nicht in Editoren verwendet werden sollten, die Repeater enthalten. Will man mehr als eine Datei hochladen, sollte das file-Element „manuell" mehrfach im Editor definiert werden, es darf jedoch kein Repeater verwendet werden.

Integration externer Datenquellen

Das Editor-Framework gestattet die Einbindung externer Datenquellen in das Formular. Dies ermöglicht eine flexible Gestaltung von Eingabewerten zur Laufzeit. Dabei werden Teile der Formulardefinition „on the fly" erzeugt und im Moment der Darstellung erst integriert.

Einfügen eines Request aus einem Servlet

Dieses Beispiel zeigt den Zugriff auf ein Servlet zur Laufzeit. Dabei werden die Ausgabedaten der Servlet-Antwort vor der Integration in das Framework mittels XSLT in eine passende Form gebracht. In Beispiel wird eine MyCoRe-Klassifikation in

```
1 <list id="COrigin" width="300" type="dropdown">
2     <item value=""><label xml:lang="de">(bitte wählen)</label></item>
3     <include uri="request:servlets/MCRQueryServlet?XSL.Style=
4         classif-to-items&am p;type=class&amp;hosts=local&amp;lang=
5         de&amp;query=/mycoreclass%5b@ID='DocPortal_class_00000002'%5d" />
6 </list>
```

Abbildung 2.42: Include Servlet-Request

eine Auswahlliste eingefügt.

Einfügen eines Definitionsabschnittes aus der Session

Es besteht auch die Möglichkeit, Daten dynamisch in der `MCRSession` zu hinterlegen und von dort in die Gestaltung des Formulars zu integrieren. Im ersten Schritt wird in einer Java-Klasse ein JDOM-Element erzeugt, welches dann im zweiten Schritt in das Formular eingebaut wird. Das Attribut `cacheable` spezifiziert dabei, ob die Daten permanent vorgehalten werden sollen. Für ständig wechselnde Daten ist hier `false` anzugeben.

```
org.jdom.Element items = new org.jdom.Element("items");
org.jdom.Element item = new org.jdom.Element("item");
item.setAttribute(...)
items.addContent(item);
...
MCRSessionMgr.getCurrentSession().put("mylist",items);
```

Abbildung 2.43: Java-Code

2.8.6 Eingabevalidierung

Einführung

Die Eingabefelder von Editor-Formularen können durch Integration von Validierungsregeln direkt durch das Editor-Servlet geprüft werden, noch bevor die Eingaben an das weiterverarbeitende Zielservlet weitergegeben werden. Wenn eine Eingabe fehlerhaft ist, wird das Formular noch einmal angezeigt. Es erscheint eine Fehlermeldung. Eingabefelder mit fehlerhaften Eingaben werden im Formular optisch markiert. Zunächst ein einfaches Beispiel:

```
<editor id="validation.demo">
  <source ... />
  <target ... />

  <validationMessage>
    <label xml:lang="de">
      Eingabefehler: Bitte korrigieren Sie die markierten Felder.
    </label>
  </validationMessage>

  <components root="root" var="/document">
    ...
    <panel id="root" lines="on">
      <cell row="1" col="1" anchor="EAST">
        <text label="Titel:" />
      </cell>
      <cell row="1" col="2" anchor="WEST" var="title">
        <textfield id="tf.title" width="80">
          <condition id="cond.title" required="true">
            <label xml:lang="de">Bitte geben Sie einen Titel ein!</label>
          </condition>
        </textfield>
      </cell>
    </panel>
    ...
  </components>
</editor>
```

Abbildung 2.44: Einfaches Beispiel für Eingabevalidierung

Das Element `validationMessage` enthält die Meldung, die am Kopf des Formulars erscheint, wenn ein oder mehrere Eingabefelder fehlerhaft ausgefüllt wurden. Dieses Element ist optional, es kann ein sprachunabhängiges label-Attribut oder ein oder mehrere sprachabhängige label-Elemente enthalten.

Jedes zu validierende Eingabefeld muss eine eindeutige ID besitzen! Neben Texteingabefeldern können auch alle anderen Feldtypen wie z. B. Auswahllisten validiert werden. Nur hidden-Felder können momentan nicht validiert werden.

Das zu validierende Feld muss ein oder mehrere `condition`-Elemente enthalten, die wiederum eine eindeutige ID besitzen müssen. Die Attribute dieser `condition`-Elemente enthalten die zu prüfenden Validierungsregeln. Ein `condition`-Element kann ein sprachunabhängiges `label`-Attribut oder ein oder mehrere sprachabhängige label-Elemente enthalten, die eine Meldung für den Benutzer enthalten. Bei einer fehlerhaften Eingabe wird diese Meldung angezeigt, wenn der Benutzer mit der Maus über das Fehlersymbol fährt, das neben dem Eingabefeld angezeigt wird.

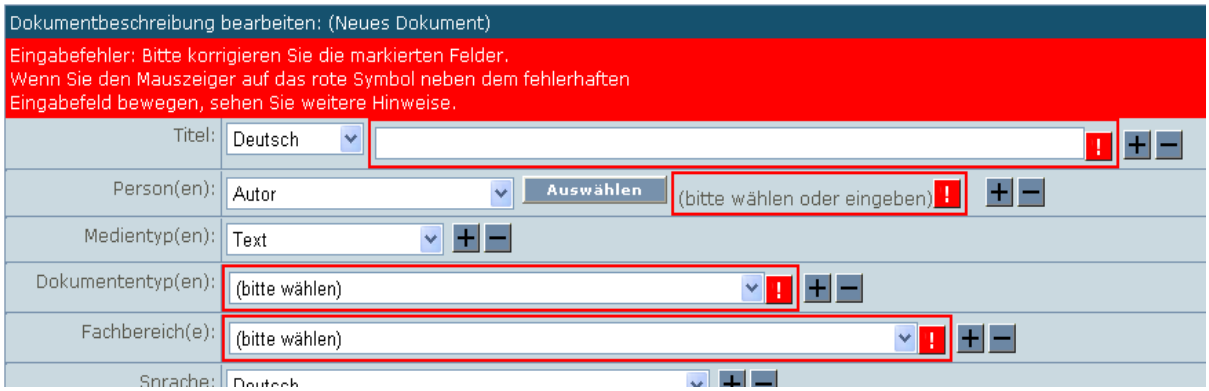


Abbildung 2.45: Fehlgeschlagene Eingabevalidierung

Ein Eingabefeld kann mehr als ein condition-Element enthalten. Dadurch können komplexe Validierungsregeln schrittweise geprüft werden, und der Nutzer bekommt eine exaktere Rückmeldung, welche dieser Regeln verletzt wurde. Wenn die Validierungsregeln auf mehrere condition-Elemente aufgeteilt wird, sollte die „required" Regel ggf. die erste sein.

```
1  <condition id="cond.title.required" required="true">
2      <label>Bitte geben Sie einen Titel ein!</label>
3  </condition>
4  <condition id="cond.title.length" maxLength="250">
5      <label>Bitte geben Sie maximal 250 Zeichen ein!</label>
6  </condition>
```

Abbildung 2.46: Beispiel für eine Reihenfolge von Regeln

Oder

```
1  <condition id="cond.title" required="true" maxLength="250">
2      <label>Bitte geben Sie einen Titel ein (maximal 250 Zeichen)!</label>
3  </condition>
```

Abbildung 2.47: Beispiel für Regeln in einer `condition`

Validierungsregeln für einzelne Felder

Die Validierungsregeln für Eingabefelder werden über Attribute oder Attributkombinationen des condition-Elements definiert. Die folgenden Regeln können geprüft werden:

`required="true"`
Es ist eine Eingabe erforderlich. Auch Eingaben, die nur aus Leerzeichen bestehen, werden abgewiesen. Ist diese Regel verletzt, werden ggf. weitere vorhandene Regeln nicht mehr geprüft.
`minLength="10"`
Die Eingabe muss aus minimal 10 Zeichen bestehen.
`maxLength="250"`
Die Eingabe darf aus maximal 250 Zeichen bestehen.
`type="integer" min="0" max="100"`
Die Eingabe muss eine ganze Zahl sein. Optional kann über die Attribute `min` und/oder `max` der Wertebereich eingeschränkt werden.
`type="decimal" format="de" min="0" max="3,5"`
Die Eingabe muss eine Zahl (ggf. mit Nachkommastellen) sein. Optional kann über die Attribute `min` und/oder `max` der Wertebereich eingeschränkt werden. Das Attribut `format` muss einen ISO 639 Sprachcode enthalten (z.B. „de" oder „en"), der bestimmt, wie Kommazahlen formatiert werden. Die `min`- und `max`-Werte müssen diesem Format entsprechend angegeben werden.
`type="datetime" format="dd.MM.yyyy" min="01.01.1970" max="31.12.2030"`
Die Eingabe muss ein Datums- oder Zeitwert sein. Optional kann über die Attribute `min` und/oder `max` der Wertebereich eingeschränkt werden. Das Attribut `format` muss ein `java.text.SimpleDateFormat` Pattern enthalten (z.B. „dd.MM.yyyy"), das bestimmt, wie der Wert formatiert sein muss. Die `min`- und `max`-Werte müssen diesem Format entsprechend angegeben werden.
`type="string" min="a" max="zzz"`
Eine Eingabe kann ein beliebiger Text sein. Optional kann über die Attribute `min` und/oder `max` der Wertebereich eingeschränkt werden.
`regexp=".+@.+\..+"`
Die Eingabe muss den angegebenen regulären Ausdruck erfüllen. Die Syntax des regulären Ausdrucks ist durch die Klasse `java.util.regex.Pattern` definiert.
`xsl="contains(.,'@')"`
Die Eingabe wird gegen eine XSL Bedingung geprüft, wie sie in XSL `if` oder `when` Elementen verwendet werden kann. Der Eingabewert wird in der Bedingung durch einen Punkt referenziert.

class="foo.bar.Validator" method="validateFooBar"
Die Eingabe wird durch Aufruf einer beliebigen Java-Methode validiert, die true oder false zurückgibt. Im oben gezeigten Beispiel würde die Methode `public static boolean validateFooBar( String value )` aus der Klasse `foo.bar.Validator` aufgerufen. Es sind so sehr spezielle Validierungsregeln wie z. B. das Prüfen einer ISBN implementierbar.

Validierungsregeln für Feldkombinationen

Es ist ebenfalls möglich, die Eingaben zweier Felder gegeneinander zu prüfen. Dazu wird ein `condition`-Element in das die beiden Felder enthaltende Panel eingefügt. Die Werte der beiden Felder können dann über Vergleichsoperatoren gegeneinander geprüft werden. Beispiele:

- Ein Eingabeformular zum Ändern eines Passwortes enthält zwei Passwortfelder, so dass das Passwort wiederholt eingegeben werden muss. Die Eingaben in diesen beiden Feldern müssen identisch sein.
- Ein Eingabeformular enthält zwei Eingabefelder für einen Datumsbereich. Das Datum im Feld `validFrom` muss kleiner oder gleich dem Datum im Feld `validTo` sein.

```
<textfield id="tf.date" width="10">
  <condition id="cond.date" type="datetime" format="dd.MM.yyyy">
    <label xml:lang="de">
      Bitte geben Sie das Datum im Format TT.MM.JJJJ ein,
      z. B. 01.09.2005
    </label>
  </condition>
</textfield>

<panel lines="off">

  <cell row="2" col="1" anchor="EAST">
    <text label="Gültig von:"/>
  </cell>
  <cell row="2" col="3" anchor="EAST">
    <text label="Gültig bis:"/>
  </cell>

  <cell row="2" col="2" anchor="WEST" var="validFrom" ref="tf.date" />
  <cell row="2" col="4" anchor="WEST" var="validTo"   ref="tf.date" />

  <condition id="cond.valid" field1="validFrom" field2="validTo"
     operator="&lt;=" type="datetime" format="dd.MM.yyyy">
    <label>Datum "Gültig von" muss &lt;= Datum "Gültig bis" sein!</label>
  </condition>

</panel>
```

Abbildung 2.48: Codebeispiel

Zunächst werden die Werte beider Eingabefelder `validFrom` und `validTo` nach den gleichen Regeln auf die Eingabe eines korrekten Datums geprüft. Wenn mindestens eines der Felder eine Eingabe enthält (was ggf. über ein `required`-Attribut erzwungen werden könnte), werden die Werte der Felder miteinander verglichen.

```
field1="validFrom"
```

Erstes zu vergleichende Feld. Syntax und Funktionsweise entsprechen dem Verhalten des Attributes `var` für Zellen.

```
field2="validTo"
```

Zweites zu vergleichende Feld. Syntax und Funktionsweise entsprechen dem Verhalten des Attributes `var` für Zellen.

```
operator="&lt;="
```

Vergleichsoperator (hier `>=`). Es können die Operatoren `=`, `>`, `<`, `>=`, `<=`, `!=` (für ungleich) verwendet werden.

```
type="datetime"
format="dd.MM.yyyy"
```

Datentyp und Format der Eingabe, analog zu den gleichnamigen Attributen, die im vorangehenden Abschnitt bereits erläutert wurden.

Alternativ kann auch hier über eine **externe Java-Methode validiert** werden. Das folgende Beispiel würde für zwei Zahlen prüfen, ob die eine das Quadrat der anderen ist:

```
<condition id="cond.quadrat" field1="zahl1" field2="zahl2"
  class="math.Validator" method="validateSquare">
  <label>Zahl 1 muss gleich dem Quadrat der Zahl 2 sein!</label>
</condition>
```

In der Klasse math.Validator könnte die Methode zur Validierung wie folgt aussehen:

```
public static boolean validateSquare( String value1, String value2 )
{
  int zahl1 = Integer.parseInt( value1 );
  int zahl2 = Integer.parseInt( value2 );
  return ( zahl1 == (zahl2 * zahl2) );
}
```

Bitte beachten Sie, dass bei externer Validierung die Werte immer als String übergeben werden. Durch Vorschalten einer weiteren Validierungsregel kann aber z.B. leicht erreicht werden, dass nur Zahlen oder bestimmte Datentypen eingegeben werden können.

Validierung ganzer Panels oder XML-Bereiche

Mehrere Eingabefelder können gemeinsam über eine XSL-Bedingung oder eine externe Java-Methode validiert werden. So kann im Extremfall das gesamte aus der Eingabe resultierende XML-Dokument validiert werden. Die Eingabevalidierung bezieht sich in diesem Fall auf ein Panel der Editor-Definition und dem daraus resultierendem XML-Element. Dies kann auch das Wurzelpanel sein. Das condition-Element für die Validierungsregel muss dabei ein Kindelement dieses Panels sein.

Über die Attribute **class** und **method** kann eine externe Java-Methode angegeben werden, die ein XML-Element validiert:

```
<components var="/input" root="root"
...
  <panel lines="off" id="root">
    <condition id="validateExt"
     class="foo.bar.Validator" method="validateInput">
      <label xml:lang="de"> Eingabefehler! </label>
    </condition>
    ...
```

ruft dann zur Validierung in der Klasse `foo.bar.Validator` die folgende Methode auf:

```
public static boolean validateInput( Element input )
```

Über das Attribut **xsl** kann eine XSL-Bedingung angegeben werden. Im folgenden Beispiel wird in einer Suchmaske sichergestellt, dass mindestens ein Eingabefeld mit einem Wert ausgefüllt ist. Dies entspricht der XSL-Bedingung: „Die Anzahl der Elemente, bei denen das value-Attribut nicht leer ist, ist größer null".

```
<components var="/query" root="root">
  ...
  <panel lines="off" id="root">
    <condition id="someInputNeeded"
     xsl="count(conditions/boolean/*[string-length(@value) &gt; 0])
        &gt; 0">
      <label xml:lang="de">Bitte einen Suchausdruck eingeben!</label>
    </condition>
    ...
```

2.9 Klassifikationsbrowser

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

Für jede Klassifikation die über einen Link im Navigationsmenü erreicht werden soll, muss ein Beschreibungsblock für das Browsen angelegt werden.

Beispiel für Browsen in Dissertationen:
Der Klassifikationsbrowser für die Klassifikation der Dissertationen wird (z.B. bei einer lokalen Beispiel-Installation) durch folgenden Request aufgerufen:

```
http://localhost:8080/docportal/browse/dissertationen
```

Dabei werden alle Requests von http://localhost:8080/docportal/browse/* vom Servlet `MCRClassificationBrowser` entgegengenommen. Hier wird der Browserpfad, der nach `…browse/` stehen muss, ausgewertet und die dem Browserpfad entsprechende Klassifikation der Navigation zugrunde gelegt.

In der Properties-Datei muss zum zugehörigen Browserpfad `dissertationen` ein Konfigurationsblock angelegt sein. Ist ein Paramter im Konfigurationsblock nicht enthalten, wird versucht in einem Default-Konfigurationsblock den entsprechenden Default-Paramter zu lesen. Ist dieser nicht vorhanden erhält man einen `MCRConfigurationException` für die Pflichtfelder.

Abbildung 2.49: Klassifikationsbrowser in einer MyCoRe-Anwendung

2.9.1 Der Konfigurationsblock

MCR.ClassificationBrowser.dissertationen.Classification

Dieser Parameter ordnet dem Browserpfad eine Klassifikation (MyCoRe ID der Klassifikation) zu.

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.Classification
     = Docportal_class_00000009
```

Hinter der Klassifikation mit der ID Docportal_class_00000009 verbirgt sich in diesem Fall die DDC-Klassifikation, nach der typischerweise Dissertationen klassifiziert werden.

MCR.ClassificationBrowser.dissertationen.EmbeddingPage

Legt fest, in welche Datei die Klassifikation eingebettet werden soll. Diese Datei muss das Tag `<classificationbrowser/>` enthalten. An der Position dieses Tags wird der Inhalt (der generierte XML-Klassifikationsbaum) zur Laufzeit eingefügt. Durch die Variabilität ist es möglich, Klassifikationen nach Belieben in beliebige XML-Dokumente einzubetten.

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.EmbeddingPage
     = mcr_doc_browse.xml
```

Die Datei `mcr_doc_browse.xml` ist im DocPortal enthalten.

MCR.ClassificationBrowser.dissertationen.Style

Legt das Stylesheet fest, mit dem das erzeugte XML-Document transformiert wird. Dabei setzt sich der Name des auszuwählenden Stylesheets aus der EmbeddingPage und dem Style zusammen.

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.Style = search
```

Es wird die Datei `mcr_doc_browse-search.xsl` zur Transformation verwendet.

MCR.ClassificationBrowser.dissertationen.EmptyLeafs

Klassifikationen besitzen im Allgemeinen eine baumartige Struktur. Zu jeder im Baum dargestellten Kategorie wird die Anzahl der damit klassifizierten Objekte ermittelt und (je nach Stylesheet) dargestellt. Kategorien die keine Objekte referenzieren (Anzahl der Dokumente=0) können durch diese Option ausgeblendet werden. Mögliche Werte sind `yes|no`.

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.EmptyLeafs = yes
```

Zeigt auch die leeren Kategorien der Klassifikation an.

MCR.ClassificationBrowser.dissertationen.View

Bei der Darstellung einer Klassifikation kann gewählt werden, ob der gesamte Baum oder nur die jeweils aktuelle Ebene angezeigt werden soll.

Durch Zu- und Aufklappen wird die nächste Ebene dann ein- bzw. ausgeblendet. Mit `View=flat` wird dann in die aktuelle Ebene gewechselt und die vorherige ist nicht mehr sichtbar. Mit `View=tree` wird immer der gesamte Baum dargestellt und die gewählten Ebenen werden auf- und zugeklappt, sodass es möglich ist, gleichzeitig mehrere Kategorien verschiedener Ebenen darzustellen.
Mögliche Werte sind `flat|tree`

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.View = flat
```

MCR.ClassificationBrowser.dissertationen.Doctype

Über den Parameter Doctype wird die Auswahl der Dokumenttypen gesteuert, die bei der Abrage der Dokumente berücksichtigt werden sollen.

Der angegebene Typ muss in der Datei `mycore.properties` über den Parameter `MCR.type_[Doctype]=[Dokumententypliste]` definiert sein. Als möglicher Wert, kann jede Angabe aus MCR.type_* verwendet werden.

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.Doctype = mydisshabs
```

Es wird `MCR.type_mydisshabs` in der Datei `mycore.properties` herausgesucht, der z.B. mit `MCR.type_mydisshabs=disshab` belegt ist. Somit werden bei dieser Klassifikation alle Dokumente vom Typ `disshab` berücksichtigt.

Für `Doctype=alldocs` wird z.B. der Wert `MCR.type_alldocs=document, article, codice, disshab, professorum, portrait` gefunden und alle Dokumente, die einem Typ aus der Typliste entsprechen, bei der Suche berücksichtigt.

MCR.ClassificationBrowser.dissertationen.Restriction

Es ist möglich für die referenzierten Objekte eine Restriktion zu setzen um die Auswahl durch eine weitere Kategorie einer beliebigen Klassifikation weiter einzuschränken. Es werden dann nur die Objekte berücksichtigt, die beiden Kriterien genügen.

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.Restriction
     = Docportal_class_00000005##TYPE0007
```

Mit dieser Restriktion werden nur die Dissertationen berücksichtigt, die auch der Klassifikation `Docportal_class_00000005` mit der Kategorie `TYPE0007` genügen. Das sind im Beispiel als Ergebnis alle Dissertationen im PDF-Format.

MCR.ClassificationBrowser.dissertationen.searchField

Hier wird das bei der Ausführung der Query zu durchsuchende Feld festgelegt.

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.searchField = ddc1
```

Es wird das DDC-Tag in den Metadaten der Dokumente als Suchparameter genutzt.

MCR.ClassificationBrowser.dissertationen.Comments

Kommentare, die optional beim Erstellen der Klassifikation für Kategorien angegeben sein können, werden im Baum mit dargestellt. Mögliche Werte sind `true|false`

Beispiel:
```
MCR.ClassificationBrowser.dissertationen.Comments = true
```

Der Default-Konfigurationsblock

Die Defaultbelegung für die notwendigen Parameter enthält folgende Werte:

```
MCR.ClassificationBrowser.default.EmbeddingPage = mcr_doc_browse.xml
MCR.ClassificationBrowser.default.Style         = search
MCR.ClassificationBrowser.default.EmptyLeafs    = yes
MCR.ClassificationBrowser.default.View          = tree
MCR.ClassificationBrowser.default.Comments      = true
MCR.ClassificationBrowser.default.Doctype       = alldocs
MCR.ClassificationBrowser.default.searchField   = cltype
```

Insbesondere die Belegung des `searchField`-Parameters kann natürlich zu falschen Ergebnissen bei der Dokumentreferenzierung und der damit verbundenen Anzeige der Anzahl der Dokumente, führen!

2.10 Klassifikationseditor

Mit dem Klassifikationseditors als Webinterface, gibt es neben der Kommandozeilen-Schnittstelle eine weitere Möglichkeit, Klassifikationen im System zu erzeugen, zu bearbeiten und zu löschen.

Um Klassifikationen editieren zu können, muss der angemeldete Nutzer je nach Bedarf über folgende Rechte verfügen:

`create-classification`	zum Erzeugen von Klassifikationen
`modify-classification`	zum Modifizieren von Kategorieeinträgen in einer Klassifikation
`delete-classification`	zum Löschen einer Klassifikation

(siehe Kapitel 5.3.4 Administrative Privilegien im UserGuide).

Konfiguration

Die Konfiguration des Klassifikationseditors erfolgt über die Angaben in der Datei `mycore.properties.classification`. Wie man Seiten, die im Fehlerfall angezeigt werden, definieren kann, zeigt nachstehender Auszug aus der Datei:

```
MCR.classeditor_page_error_user   = editor_error_user.xml
MCR.classeditor_page_cancel       = classeditor_cancel.xml
MCR.classeditor_page_error_id     = classeditor_error_clid.xml
MCR.classeditor_page_error_move   = classeditor_error_move.xml
MCR.classeditor_page_error_delete = classeditor_error_delete.xml
```

Analoge Properties zum Klassifikationsbrowser:

```
MCR.classeditor.EmbeddingPage = mcr_doc_browse.xml
MCR.classeditor.Style         = edit
```

Für die Klassifikationen, die über die ID definiert sind, wird analog zum Browsen der Browserpfad definiert. Dies dient dazu, um auch hier die richtige Zuordnung zu den, durch die Klassifikation referenzierten Dokumente zu besitzen:

```
MCR.classeditor.atlibri_class_00000012 = codices
MCR.classeditor.atlibri_class_00000013 = archiv
MCR.classeditor.atlibri_class_00000009 = ddc
MCR.classeditor.atlibri_class_00000008 = eval
MCR.classeditor.atlibri_class_00000007 = dnb
MCR.classeditor.atlibri_class_00000006 = medien
MCR.classeditor.atlibri_class_00000005 = type
MCR.classeditor.atlibri_class_00000003 = origin
```

2.10.1 Start des Klassifikationseditors

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

Beispiel:
`http://localhost:8080/docportal/browse?mode=edit`

Die Handhabung des Klassifikationseditors ist im UserGuide beschrieben.

3 Module

Module sind eine sinnvolle Ergänzung der MyCoRe-Kern-Komponenten. Sie können optional in die Applikationen der Anwender eingebaut werden und sollen diese funktional bereichern.

3.1 Das SimpleWorkflow-Modul

3.1.1 Allgemeines

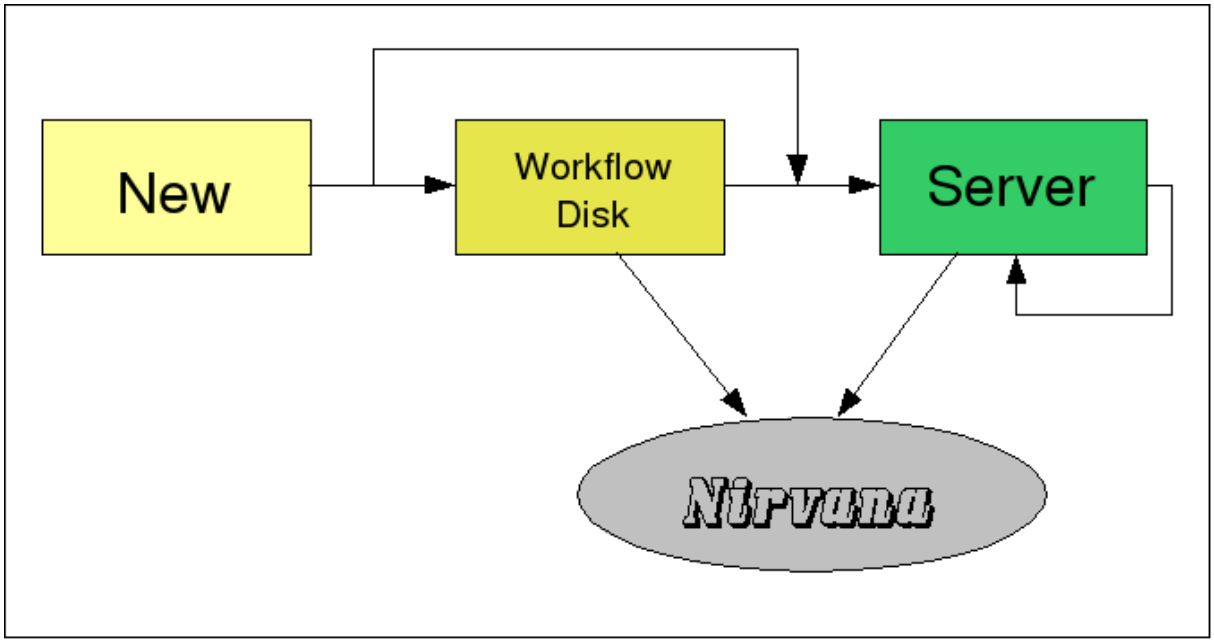


Abbildung 3.1: Grundübersicht des SimpleWorkflow

Für die Erstellung einfacher Anwendungen, welche nur einen relativ primitiven Arbeitsablauf bedingen, war es notwendig ein einfacher Werkzeug zur Gestaltung dieser Abläufe anzubieten. So entstand die Idee des `SimpleWorkflow`. Eigentlich handelt es sich dabei gar nicht um einen Workflow, sondern eher um eine Menge von kleinen Werkzeugen, die über HTTP-Request zu einem interaktiven Arbeitsablauf zusammengegefügt werden können. Der Begriff *Workflow* soll dabei die Bearbeitungsebene zwischen dem ersten Erstellen eines Objektes und dessen Ablage in den Server und die dabei vor sich gehenden Arbeitsschritte beschreiben. Physisch handelt es sich um ein Verzeichnis `workflow`, unter welchem für jeden Objekttypen Unterverzeichnisse angelegt sind und in welchem die Daten zwischengespeichert werden. Konsultieren Sie zum besseren Verständnis auch die Beschreibung im MyCoRe-UserGuide.

Der SimpleWorkflow besteht im wesentlichen aus einer Sammlung von Servlets, welche über HTTP-Request angesprochen, verschiedene Bearbeitungsprozesse initiieren. Dabei wird gleichzeitig eine Berechtigungsprüfung für den Zugriff auf einzelne Aktionen durchgeführt. Für Aktionen das Neuanlegen von Objekten ist dabei die Permission 'create-'ObjectTyp erforderlich. Ist das Objekt schon vorhanden, so entscheiden die ACL's des Systems über die Möglichkeit der Bearbeitung.

ACL-Permission	Bedeutung
`writewf`	Gestattet das Bearbeiten der Objekte im Workflow (d. h. auf dem Plattenbereich).
`deletewf`	Gestattet das Löschen von Objekten im Workflow (d. h. auf dem Plattenbereich).
`writedb`	Gestattet das Bearbeiten von Objekten im Server.
`deletedb`	Gestattet das Löschen der Objekte aus dem Server.

Tabelle 3.1: Permissionliste für den SimpleWorkflow

3.1.2 Komponenten und Funktionen

Der SimpleWorkflow besteht aus einer Reihe von Servlets. Die folgende Tabelle listet die Servlets auf.

Servlet	Funktion
MCRStartEditorServlet	Das Servlet dient als Startpunkt für alle Arbeiten mit dem SimpleWorkflow.
MCRCheckCommitDataServlet	Wird vom Editor über ein <target>-Tag aufgerufen und schreibt die Metadaten nach Bearbeitung in den Server.
MCRCheckEditDataServlet	Wird vom Editor über ein <target>-Tag aufgerufen und schreibt die Metadaten nach deren Bearbeitung auf die Platte.
MCRCheckNewDataServlet	Wird vom Editor über ein <target>-Tag aufgerufen und schreibt die neuen Metadaten auf die Platte.
MCRCheckCommitFileServlet	Wird vom Editor über ein <target>-Tag aufgerufen und schreibt die Derivate-Daten nach Bearbeitung in den Server.
MCRCheckNewFileServlet	Wird vom Editor über ein <target>-Tag aufgerufen und schreibt die neuen Derivate-Daten auf die Platte.
MCRFileListWorkflowServlet	Listet die auf der Platte befindlichen Dateien in den Derivaten auf.
MCRFileViewWorkflowServlet	Gestattet den Zugriff auf eine Derivate-Datei auf der Platte.
MCRListDerivateServlet	Listet alle auf der Platte befindlichen Derivate auf.
MCRListWorkflowServlet	Erzeugt einen XML-Baum, welcher zur Darstellung des Workflow (Platteninhaltes) benötigt wird.

Tabelle 3.2: Übersicht der SimpleWorkflow-Servlets

Die folgende Abbildung soll noch einmal die Beziehungen der einzelnen Komponenten verdeutlichen. Hier gibt es zwei Komplexe. Der erste arbeitet mit dem Plattenzwischenspeicher und stellt einen simplen *Workbasket* dar. In diesen Korb können Objekte neu eingestellt, bearbeitet, ergänzt, geprüft oder wieder gelöscht werden. Ist dieser Arbeitsschritt fertig, so kann das Objekt in den Server hoch geladen werden. Hier gibt es wieder die Möglichkeit, so der Nutzer die Berechtigung dazu hat, Objekte zu bearbeiten, zu verändern oder zu löschen. Alle diese Schritte arbeiten direkt gegen den Server. Ausgangspunkt aller Aktivitäten ist dabei das `MCRStartEditorServlet`. Hier wird beim Aufruf eine Aktion mit gegeben, welche den weiteren Ablauf bestimmt. Entweder werden jetzt die ToDo's direkt ausgeführt (Löschen) oder es wird z. B. eine Web-Seite mit einer Editor-Maske aufgerufen. Diese wiederum beinhaltet im <target>-Tag das zu benutzende Verarbeitungs-Servlet, welches dann je nach Aufgabe wieder zu einer Web-Seite oder der Workflow-Ansicht verzweigt.

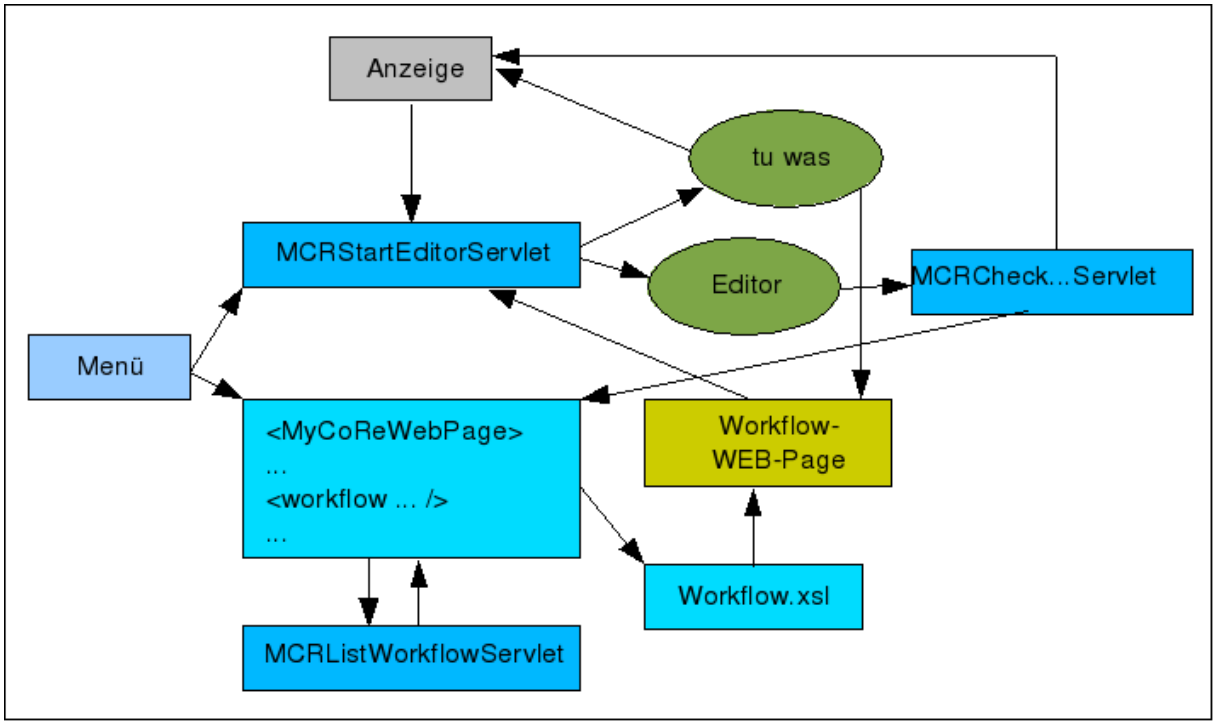


Abbildung 3.2: Ablaufschema im SimpleWorkflow

3.1.3 Installation

In DocPortal ist ein großer Teil der Funktionen bereits standardmäßig integriert. Grundsätzlich muss analog zu DocPortal im Verzeichnis `modules` der Anwendung das Verzeichnis `module-swf` mit dem darin befindlichen `build.xml` File angelegt werden. Dies kopiert alle Daten aus dem Kern in die Anwendung. Nun muss der Modul noch integriert werden.

Bei der Verwendung in anderen Applikationen sind wieder die zwei Funktionskomplexe zu unterscheiden – für die einfache Workflow-Gestaltung auf der Platte müssen sie:

1. die folgende Zeile in das XSL-Stylesheet kopieren, welches die Auswertung Ihrer XML-Web-Seiten realisiert (z. B. `MyCoReWebPage.xsl`).
   `<xsl:include href="workflow.xsl" />`

2. Weiterhin finden Sie im Kern das Stylesheet `mycoreobject-document-to-workflow.xsl`. Dieses ist eine Transformationsvorlage für die Transformation von den XML-Objekt-Metadaten in eine Workflow-interne XML Struktur. Für jeden Ihrer Metadaten-Typen muss eine solche Konverter-Datei mit Namen `mycoreobject-<type>-to-workflow.xsl` in Ihrer Anwendung vorhanden sein.

3. Als letztes muss der Workflow in eine XML-Webseite integriert und diese entsprechend über Menüpunkte aufgerufen werden.

Die Integration des SimpleWorkflow in die Präsentationsseiten erfolgt unter Einbeziehung der bereitgestellten Icons und eines dahinter liegenden Linke oder Formulars. Dieses kann an beliebiger Stelle in der Präsentation platziert werden.

```
<MyCoReWebPage>
    <section title="Bearbeiten von Objekten aus dem Workflow" xml:lang="de">
        <table id="contentArea" width="100%" cellspacing="20" cellpadding="0">
            <tr>
                <td>
                    <center>
                        <span class="textboldnormal">
                            <u>Bearbeiten und Verwalten von Dokumenten<br />
                            (nur für berechtigte Editoren)</u>
                        </span>
                        <p></p>
                        <workflow type="document" step="editor" />
                    </center>
                </td>
            </tr>
        </table>
    </section>
</MyCoReWebPage>
```

Abbildung 3.3: Beispiel-Web-Seite für den Aufruf eines Workflow

```
<a href="{$ServletsBaseURL}MCRStartEditorServlet{$HttpSession}?
     tf_mcrid={$id}&amp;
     se_mcrid={$id}&amp;
     type={$type}&amp;
     step=commit&amp;
     todo=seditobj"
  ><img
     src="{$WebApplicationBaseURL}images/static/workflow_objedit.gif"
     title="{$OMD.EditorEdit}"/>
</a>
```

3.1.4 Konfiguration

Die Konfiguration des SimpleWorkflow beschränkt sich auf einige wenige Dinge. Anzugeben sind:

- MCRObjectID-Projektnamen für die einzelnen Metadatentypen,
- die Verzeichnisnamen des Plattenspeichers,
- die Mail-Verteilung und
- die Lokation für das Nachladen von Informationen.

```
########################################################################
# SimpleWorkflow                                                       #
########################################################################
# The project ID
MCR.default_project_id=DocPortal
MCR.default_project_type=document
MCR.document_project_id=DocPortal
...
# Editor path
MCR.editor_document_directory=/home/mcradmin/docportal/workflow/docume
nt
...
# Editor Mail adresses for Messages add1@serv1,add2,@serv2,...
# Editor flags for todo and type
MCR.editor_document_wnewobj_mail=user@domain.xyz
MCR.editor_document_weditobj_mail=
MCR.editor_document_wcommit_mail=user@domain.xyz
MCR.editor_document_wdelobj_mail=
MCR.editor_document_seditobj_mail=user@domain.xyz
...
MCR.editor_mail_sender=user@domain.xyz
MCR.editor_mail_application_id=MyCoReSample
# Generic mail configuration for MCRMailer
MCR.mail.server=myhost.de
MCR.mail.protocol=smtp
MCR.mail.debug=false
# Editor HostAlias for Classification
MCR.editor_baseurl=local
```

Hier noch einige Hinweise:
Kopieren Sie den Abschnitt in Ihre Datei `mycore.properties.private` und ergänzen Sie die Einträge mit den Daten Ihrer Anwendung. Ein Teil der vom `MCRStartEditorServlet` veranlassten Aktionen ist so implementiert, dass sie auf Wunsch eine Mail an eine oder mehrere Adressen schicken können. Wenn Sie für den Konfigurationswert, welcher durch das Paar `type_todo` beschrieben wird, nichts angeben, so wird die Mail unterdrückt. Alle Angaben in diesem Konfigurationsabschnitt sollten selbsterklärend sei.

3.1.5 Ergänzung eigener ToDo's

Das MCRStartEditorServlet gestattet eine Erweiterung mit eigenen Funktionen durch einfache Vererbung. Erstellen Sie eine Klasse MCRStartEditorServletMyToDo als Ableitung des MCRStartEditorServlet. Hierin können Sie nun Methoden definieren, welche als ToDo direkt aufgerufen werden können[11]. Eine Beispieldatei befindet sich im sources-Zweig von DocPortal. Um das Erweiterungs-Servlet nutzen zu können, muss diese noch in der Datei web.xml der Servlet-Engine bekannt gemacht werden.

[11] siehe Installation des SWF-Modules

3.2 Das Webservice-Modul

3.2.1 Allgemeines

Für das Webservice-Module wird das Axis-Framework verwendet (http://ws.apache.org/axis/). Über den Webservice können MyCoRe-Objekte geholt und Queries in der neuen Abfragesprache ausgeführt werden. In dem Modul ist auch ein Beispiel eines Clients enthalten, der den installierten Webservice von MyCoRe nutzt.

3.2.2 Installation des Webservices

Hierzu müssen in der mycore.properties.private die Properties für den Axis-Administrator gesetzt werden:

```
MCR.ws_admin=Kennung des Axis-Administrators

MCR.ws_adminpasswd=und zugehöriges Passwort
```

Von `ant webapps` werden Kennung und Passwort in die Datei webapps/WEB-INF/users.lst eingetragen. Docportal wird wie gewohnt gestartet und durch Eingabe der Url

```
http://localhost:8080/servlets/AxisServlet
```

wird geprüft, ob Axis richtig konfiguriert ist. Danach wird das Deployment des Webservice mit `ant webservice.deploy` (docportal/modules/module-webservices/build.xml) durchgeführt. Ein erneuter Aufruf des AxisServlets zeigt den MyCoRe-Webservice mit dem Namen `MCRWebService` und den Methoden `MCRDoRetrieveObject` und `MCRDoQuery` an. Ein Klick auf die `WSDL` (Web Service Description Language) von `MCRWebService` zeigt die Parameter und Datentypen der Rückgabewerte an.

Mit `ant webservice.undeploy` wird das Undeployment des Webservice `MCRWebService` durchgeführt.

Mit

```
http://localhost:8080/services/MCRWebService?
      method=MCRDoRetrieveObject&id=<mcrid>
```

kann überprüft werden, ob der `MCRWebService` Ergebnisse liefert.

Sollten Sie beim Deploy/Undeploy die Meldung „Exception in axis-admin" oder „axis-admin failed with {http://xml.apache.org/axis/}HTTP (401)Unauthorized" erhalten, setzen Sie wie am Anfang dieses Abschnitts beschrieben, Kennung und Passwort des Axis-Administrators. Nach einem erneuten `ant webapps` wiederholen Sie das `ant webservice.deploy`. Die Konfigurationsdatei für Axis isti docportal/config/server-config.wsdd.

3.2.3 Client für den Webservice erzeugen

Mit '`ant client.cmd`' wird ein Webservice-Client erstellt, der den Webservice `MCRWebService` nutzt. Hierzu werden WSDL-Informationen vom Server geholt und mittels Axis die Stubs generiert und im Verzeichnis `module-webservices/build/src` gespeichert. Anschließend werden alle Daten kompiliert und `module-webservices/build/bin/wsclient.cmd` (Windows) gebaut.

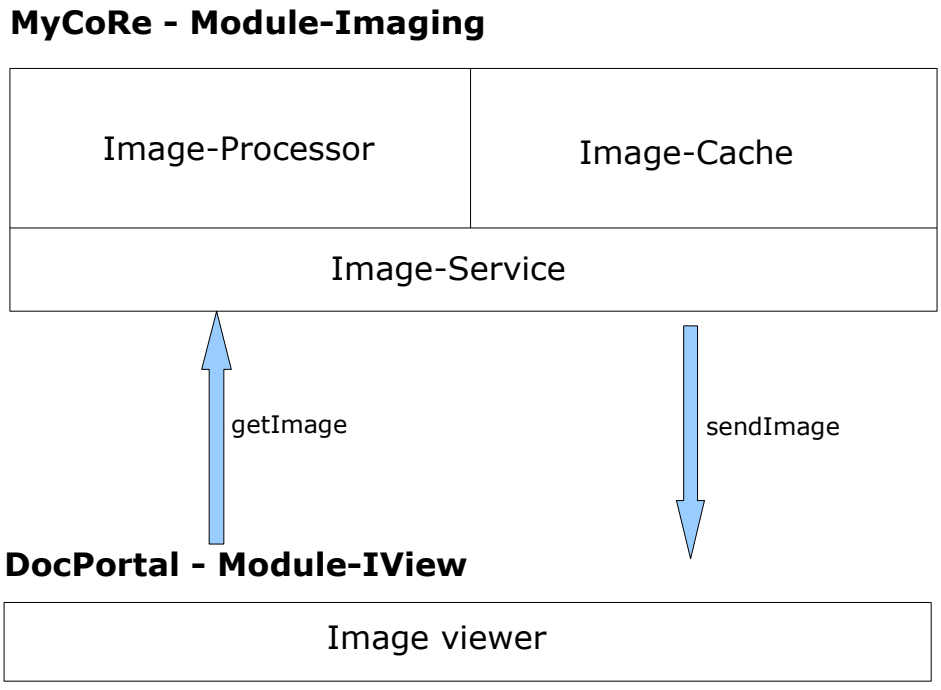
`[ToDo]: sh für Linux bauen]`

3.3 Bildbetrachter

3.3.1 Allgemeines

DocPortal bietet einen sogenannten „Image-Viewer" für das komfortable Betrachten von Bilddateien an. Dieser eignet sich sehr gut für Bildarchive, oder jegliche Content-Repositories in denen Bilddaten verwaltet und angezeigt werden sollen.

Der Bildbetrachter basiert, wie MyCoRe, auf Java und XML/XSL. Grundsätzlich wird die Funktionalität in zwei getrennten Modulen, „Module-Imaging" im MyCoRe-Kern und „Module-IView" in DocPortal realisiert. Die folgende Abbildung zeigt die allgemeine Systemarchitektur.

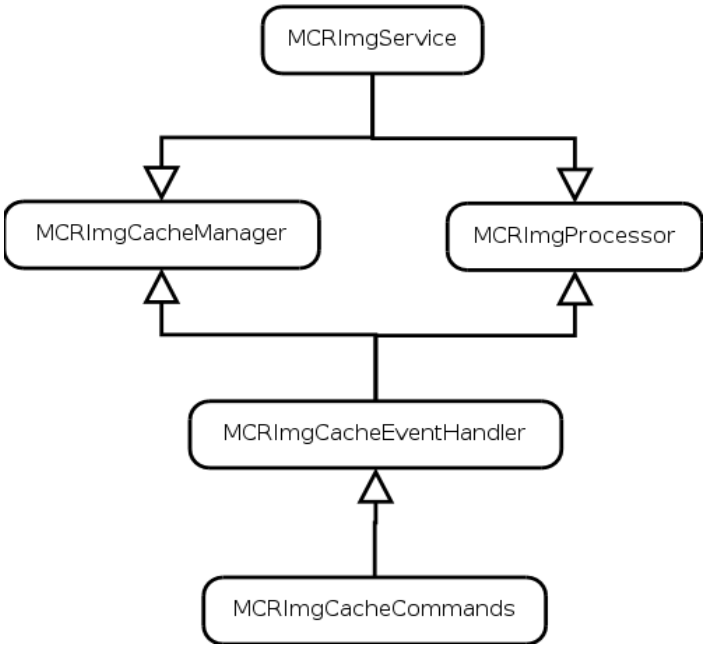


3.3.2 Module-Imaging – API zur Bildbearbeitung

Das „Module-Imaging" befindet sich im MyCoRe-Kern und bietet eine API um Bilddaten, in MyCoRe speziell abzuspeichern, performant zu laden und gegebenenfalls zu verändern (Skalierung, Bereichsextraktion).

Das Modul benutzt die Bildbearbeitungsbibliothek „JAI" von Sun als Grundlage für Bildberechnungen.

Die folgende Abbildung zeigt gibt Überblick, zu den verschiedenen Klassen und deren Zusammenhang.



Bilder lesen

Bilder können über den Image-Service geladen werden. Das Modul organisiert dabei automatisch den für das Bild optimalen Ladevorgang. Entspechend der Ausgabegröße wird entschieden, ob der Cache benutzt werden kann oder eine Live-Skalierung durchgeführt werden muss.

Hinweis: *Beachten sie, dass performante Leseoperation nur garantiert werden können, wenn der Cache eingeschaltet ist. Lesen sie dazu den entsprechenden Abschnitt „Cache".*

Bilder schreiben

Bei eingeschaltetem Cache, werden die Bilder redundant abgelegt. Die gecachten Bilder werden im selben Content-Store gespeichert, wie der restliche Content. Es ist nicht nötig, dafür weitere Konfigurationen vorzunehmen.

Cache

Der Cache speichert grundsätzlich Bilder redundant im Repository. Dabei werden die am häufigsten angeforderten Bildgrößen bereits fertig berechnet vorgehalten. Dadurch müssen bei Leseoperationen keinerlei Berechnungen mehr durchgeführt werden und die Ladezeit wird sehr gering.

Es können grundsätzlich drei verschiedene Caches angelegt werden:

1. Thumbnail-Cache

   Dieser Cachteil speichert eine Thumbnailversion des Bildes ab. Die Größe des Thumbnails wird in der Properties-Datei $DocPortal/modules/module-iview/mycore.properties.iview im Bildbetrachter („MCR-IView") als Pixel angegeben.

   ```
   MCR.Module-iview.thumbnail.size.width=100

   MCR.Module-iview.thumbnail.size.height=75
   ```

2. Übersichtsbilder

   Dieser Cacheteil speichert Bilder in einer gebräuchlichen Bildschirmauflösung um Übersichten („Bildgröße an Seite-, Breite angepasst") vorzuhalten.

   Weiterhin wird er dazu benutzt, Ausgabegrößen, die kleiner sind als diese Cachegröße, beschleunigt zu berechnen.

   Zur Konfiguration sollte die am meisten benutzte maximale Monitorauflösung der Nutzer angegeben werden. In der Properties-Datei $DocPortal/modules/module-iview/mycore.properties.iview im Bildbetrachter („MCR-IView") sind die beiden Werte als Pixel anzugegeben:

   ```
   MCR.Module-iview.cache.size.width=1280

   MCR.Module-iview.cache.size.height=1024
   ```

3. Originalgröße

   Dieser Cacheintrag speichert vom Originalbild ein gekacheltes TIFF in der Originalauflösung ab. Das ist nötig, wenn Nutzer innerhalb von Bildern , wie zum Beispiel bei Landkarten, navigieren müssen.

   Durch diesen Cacheeintrag müssen beim Lesen von Ausschnitten aus Bildern, nur die Informationen der Ausschnitte geladen werden, und nicht mehr das ganze Bild.

Bilder die bereits im Original im TIFF-Format vorliegen, wird kein eigener Cache erzeugt, sondern das Originalbild gekachelt und überschrieben. Für alle Formate außer TIFF, werden separate Cache-Einträge angelegt.

In der Properties-Datei $DocPortal/modules/module-iview/mycore.properties.iview im Bildbetrachter („MCR-IView") wird dieser Wert angegeben.

```
MCR.Module-iview.cacheOrig=false|true
```

Hinweis:
Beachten sie, dass dieser Cache-Eintrag das Repository stark vergrößern kann, wenn sie Bilder nicht als TIFF abspeichern. Nur wenn sie wirklich innerhalb der Bilder navigieren möchten, sollte dieser Cache eingeschaltet werden.

3.3.3 Module-IView - Bildbetrachter

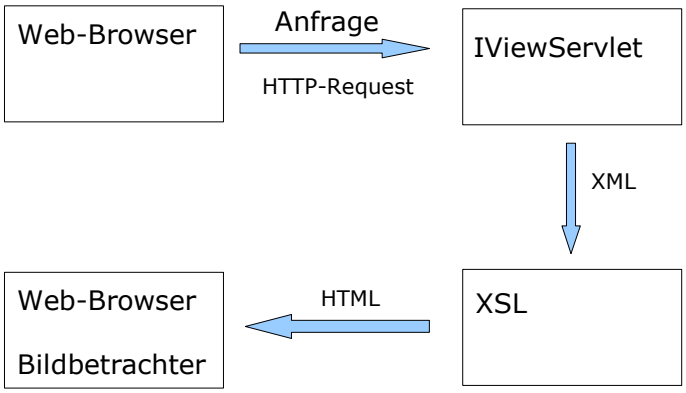
Allgemeines

Das Modul-IView ist ein Anwendungsmodul von DocPortal und realisiert einen Bildbetrachter. Es basiert auf dem Modul „Module-Imaging" aus dem MyCoRe-Kern, nutzt dessen API.

Es besteht im Wesentlichen aus einem Servlet, zwei XSL-Stylesheets und einer JavaScript-Bibliothek. Um den Bildbetrachter in der Anwendung zu nutzen, muss der Web-Browser JavaScript interpretieren können.

Funktionsweise

Der Bildbetrachter wird über folgenden Mechanismus erzeugt.



Der Bildbetrachter wird über eine HTTP-Anfrage an das Iview-Servlet initialisiert. Das Iview-Servlet erstellt die aktuelle Konfiguration für die Session und leitet diese als XML-Document an das Layout-Servlet weiter. Das Layout-Servlet seinerseits

generiert mithilfe XSL („mcr-module-iview.xsl“) HTML. Dieses HTML wird dann als Response an den Web-Browser zurückgesendet und als Bildbetrachter dargestellt.

Integration

Der Bildbetrachter-IView kann in XSL oder von einer MyCoRe-WebPage aufgerufen werden .

XSL

Iview kann in jedes XSL-Stylesheet eingebunden werden. Dazu muss lediglich im Kopf des entsprechenden Stylesheets mcr-module-startIview.xsl eingebunden werden.

```
<include href="mcr-module-startIview.xsl" />
```

Anschließend stehen die folgenden Templates zur Verfügung

1. Test auf unterstützte Hauptdatei

   Das folgende Template überprüft, ob die Hauptdatei eines Derivates ein Bild ist, das mit Iview dargestellt werden kann.

   ```
   <xsl:call-template name="iview.getSupport">
     <xsl:with-param name="derivID" />
   </xsl:call-template>
   ```

   Antwort:

   - leerer String => Hauptdatei wird nicht unterstützt
   - String mit Wert => Hauptdatei wird unterstützt, Wert ist der absolute Pfad der Hauptdatei (wird zum Aufruf des Viewers benötigt)

2. Aufruf des Bildbetrachters im eingebetteten Modus (wird als IFrame in eine HTML-Seite integriert)

```
<xsl:call-template name="iview">
                <xsl:with-param name="derivID" />
                <xsl:with-param name="pathOfImage" />
                <xsl:with-param name="height" />
                <xsl:with-param name="width" />
                <xsl:with-param name="scaleFactor" />

                <xsl:with-param name="display" />
                <xsl:with-param name="style" />
   </xsl:call-tenmplate>
```

Folgende Parameter sind zu übergeben:

- derivID = Derivate-ID
- pathOfImage = absoluter Pfad des Bildes oder Ordners
- height = Höhe des eingebetteten Fensters
- width = Breite des eingebetteten Fensters
- scaleFactor = Zoom
  - 0.1 ... 1.0
  - „fitToWidth" - an Breite angepasst
  - „fitToScreen" - an Seite angepasst
- display = Ansicht des Viewers
  - „minimal" = nur die Navigationsleiste (vor, zurück) wird eingeblendet
  - „normal" = die obere Menüleiste wird angezeigt
  - „extended" = obere und erweiterte Menüleiste werden eingeblendet
- style = Modus, in der Bild angezeigt wird
  - „thumbnail" = Thumbnailübersicht
  - „image" = Bild ansich
  - „text" = technische Metadatenansicht

3. Ermitteln der Bildbetrachter-Adresse für den Vollbildmodus

   Der Bildbetrachter kann auch im Vollbildmodus aufgerufen werden. Das folgende Template ermittelt die Adresse dafür.

```
<xsl:call-template name="iview.getAddress">
        <xsl:width-param name="derivID" />
        <xsl:width-param name="pathOfImage"/>
        <xsl:width-param name="width" />
        <xsl:width-param name="height" />
        <xsl:width-param name="scaleFactor" />
        <xsl:width-param name="display" />
        <xsl:width-param name="style" />
</xsl:call-template>
```

   Die zu übergebenden Parameter sind identisch mit Punkt 2 – Aufruf im eingebetteten Modus.

Antwort: Adresse des Bildbetrachters

4. Erzeugen eines Thumbnail

Der Image-Viewer kann auch einfach dazu benutzt werden, Thumbnails von Bildern anzuzeigen. Durch den Aufruf des folgenden Templates wird ein HTML-Image (<img...>) erzeugt, ohne den Viewer ansich.

```
<xsl:call-template name="iview.getEmbedded.thumbnail" >
  <xsl:with-param name="derivID" />
  <xsl:with-param name="pathOfImage" />
</xsl:call-template>
```

Zu übergebende Parameter:

- derivID – Derivate-ID
- pathOfImage – absoluter Pfad des Bildes

Antwort:

```
<img src="Adresse des Thumbnails" />
```

XML – MyCoRe-WebPage

Statische Webseiten für DocPortal werden über XML-Dateien, sogenannte MyCoRe-WebPages generiert. Jede Webseite hat eine XML-Seite, aus der sie generiert wird. Diese Webseiten können auch im WYSIWYG-Modus über das Module-WCMS gepflegt werden.

In diese Webseiten, kann auch der Bildbetrachter Iview eingebunden werden. Folgender Aufruf ruft den Bildbetrachter auf:

```
<iview
    @derivid
    @pathofimage
    @height
    @width
    @scalefactor
    @display
    @style
</iview>
```

Die zu übergebenden Parameter sind identisch zum vorigen Punkt XSL, 2 – Aufruf im eingebetteten Modus.

Konfiguration

Die Konfiguration des Bildbetrachters wird über die Datei $DOCPORTAL_HOME/modules/module-iview/config/mycore.properties.iview vorgenommen.

Was soll angezeigt werden?

Der Bildbetrachter kann grundsätzlich Dateien mit folgenden MIME-Types anzeigen:

```
jpeg,gif,tiff,tif,bmp,png,FlashPix,flashpix
```

Über das Property

```
MCR.Module-iview.SupportedContentTypes
```

kann gesteuert werden, welche Dateien Iview anzeigen soll. Es sind hier die MIME-Types, kommasepariert, anzugeben.

Sortierung von Bildsammlungen

Exisitieren mehrere Bilder in einem Derivat, kann die Sortierung voreingestellt werden. Beim ersten Aufruf wird automatisch in der eingestellten Reihenfolge sortiert.

Das dafür zuständige Property ist

```
MCR.Module-iview.defaultSort
```

Mögliche Werte:

- `name` – nach Dateiname
- `size` – nach Dateigröße
- `lastModified` – nach Datum der letzten Änderung

Die Reihenfolge wird über das zweite Property eingestellt:

```
MCR.Module-iview.defaultSort.order
```

Werte: `ascending`, `descending`

Zoomstärke

Die Veränderung des Zoomwertes beim Vergrößern und Verkleinern kann angepasst werden. Standardmäßig ist +-20% eingestellt (das bedeutet am Beispiel: aktuell Zoomstufe=40% -> Vergrößerung -> Zoomstufe=60%).

Das folgende Property steuert die Zoomstärke

```
MCR.Module-iview.zoomDistance
```

Werte: `0.1F` bis `1.0F`

Qualität des Bildes

Bilder werden von Iview immer als JPEG-Datei ausgegeben. Die Qualität der Bilder kann über den Kompressionsfaktor geändert werden.

```
MCR.Module-iview.jpegQuality
```

Werte: `0.1` bis `1.0`

Cache

Für die Nutzung des Bildbetrachters kann ein Cache zu Hilfe genommen werden. Leseoperationen auf Bilder werden dadurch stark beschleunigt.

Die Nutzung des Cache kann ein- oder ausgeschaltet werden. Wenn der Cache ausgeschaltet ist, werden Lese- und Schreiboperationen ohne Cache durchgeführt.

Das Property dafür ist

```
MCR.Module-iview.useCache=true|false
```

Weitere Informationen zur Konfiguration des Cache sind im Abschnitt Module-Imaging -> Cache nachzulesen.

4 Anmerkungen und Hinweise

4.1 Ergänzung der DocPortal-Beispieldaten

Mit Version 1.1 wurden die Beispieldaten für das DocPortal aus der Distribution des selbigen herausgelöst und in eine separaten CVS-Baum untergebracht. Dies hat den Vorteil, dass

- die Installation des DocPortals nicht mehr von Beispieldaten abhängig ist,
- man nach der Installation ein leeres, betriebsbereites System hat,
- die Distribution des Samples schlanker und der Download damit schneller ist,
- mehr Beispiele in einer extra-CVS-Distribution angeboten werden können und
- die Beispiele gezielt geladen und auch wieder entfernt werden können.

Die Beispieldaten stehen auf dem CVS-Server in Essen[12] in einem extra CVS-Baum mit dem Namen `content` bereit. Dieser Enthält eine Sammlung einzelner Beispieldatengruppen. Nach dem checkout können die Gruppen je nach Wunsch einzeln installiert werden.[13] Dabei spielt das jeweils mitgelieferte `build.xml`-Skript eine wichtige Rolle, hier sind alle Funktionen zur Arbeit mit dem Beispiel definiert.

Um neue Beispieldaten bereitzustellen gibt es zwei Wege: es wird eine Beispielgruppe mit Daten ergänzt oder es wird einen neue Beispielgruppe aufgebaut.

4.1.1 Ergänzungen in einer Beispielgruppe

Folgende Arbeiten sind erforderlich:

- Erzeugen der Metadaten für das Dokument (ggf. mit Daten für den Autor und/oder die Institution).
- Erzeugen des/der Derivate.
- Integration des Ladens und Entfernens im `build.xml`-Skript in den `target`-Abschnitten `load` und `remove`.

4.1.2 Hinzufügen einer neuen Beispielgruppe

Hier sind mehr Schritte erforderlich. Dabei ist immer darauf zu achten, dass die Beispielgruppe in sich vollständig ist, d. h. alle Autoren- und Institutionsdaten mitgeliefert werden. Da zum Laden der Daten das Update-Kommando verwendet wird, ist sichergestellt, dass es keine Doppelung im System gibt. Verwenden Sie möglichst die schon vorhandenen Autoren und Institutionen erneut.

[12] server.mycore.de
[13] siehe UserGuide

Abbildung 4.1: Mindeststruktur einer Beispielgruppe

- Kopieren Sie das build.xml-Skript von einem bestehenden Beispiel und adaptieren Sie es. Es muss mindestens die targets `info`, `load` und `remove` beinhalten. Ggf. sind noch weitere targets zum Kopieren von Stylesheets usw.
- Schreiben Sie ein kurzes ReadMe-File im ASCII-Format mit Installationshinweisen.
- Testen Sie das fertige Beispiel.
- Commiten Sie alles in den CVS-Server.

5 Anhang

5.1 Abbildungsverzeichnis

5.2 Tabellenverzeichnis