User Guide JSPDocPortal

RELEASE 2.0

29. Mai 2008

Heiko Helmbrecht (München)

Anja Helmbrecht-Schaar (Rostock)

Robert Stephan (Rostock)

Vorwort

JSPDocPortal ist eine auf dem MyCoRe-Kern basierende Beispielanwendung für einen Publikations- und Dissertationsserver.

Diese Dokumentation enthält **ausschließlich** Informationen zu **JSPDocPortal**, als eine Alternative zu DocPortal – der MyCoRe-Standardbeispielanwendung.
Es werden Installation, Konfiguration beschrieben und Hinweise zum Aufbau einer eigenen Applikation auf Basis von JSPDocPortal gegeben. Weiterhin werden die Unterschiede in Aufbau und Funktionalität dargestellt.

Dieser Dokumentation liegt der MyCoRe-UserGuide zu Grunde. Bei identischen Sachverhalten, wie der Installation von MyCoRe und der Beschreibung grundlegender Funktionalität (z.B. Command-Line-Interface (CLI) oder MyCoRe-Datenmodell) werden wir auf die entsprechenden Kapitel im MyCoRe User Guide verweisen.

Hinweis:
Diese Dokumentation beschreibt das Release 2.0 der MyCoRe- und JSPDocPortal-Software. Um die genannten Features nutzen zu können, benutzen Sie bitte diesen Codestand inklusive der dazugehörigen Fixes!

Inhaltsverzeichnis

1 Allgemeines zu JSPDocPortal

1.1 Vorbemerkungen

JSPDocPortal ist eine auf dem MyCoRe-Kern basierende Beispielanwendung für einen Dokumenten- und Dissertationsserver.

Zusätzlich zu den in MyCoRe verwendeten Technologien kommen im JSPDocPortal noch die folgende Technologien zum Einsatz:

JSP: JavaServer Pages zur einfachen dynamischen Erzeugung von XML- und HTML-Ausgaben

JSTL: JavaServer Pages Standard Tag Library – Sammlung von Bibliotheken, die grundlegende Funktionalitäten für verschiedene Bereiche anbieten.

Folgende Bibliotheken finden Verwendung:

Funktionsbereich	***URI***	***Präfix***
Kern	http://java.sun.com/jsp/jstl/core	**c**
XML Verarbeitung	http://java.sun.com/jsp/jstl/xml	**x**
Internationalisier-ung / Formate	http://java.sun.com/jsp/jstl/fmt	**fmt**
Funktionen	http://java.sun.com/jsp/jstl/functions	**fn**
eigene TagLibrary		
Mycore Funktionalität	`/WEB-INF/lib/mycore-taglibs.jar`	**mcr**

Jbpm: externe Workflow-Engine zur Modellierung und Intergration bibliothekarischer Geschäftsprozesse.

1.2 Die Applikation JSPDocPortal

Auf den ersten Blick ähneln sich die Applikationen **DocPortal** und **JSPDocPortal**. Beide setzen die Funktionalität von **MyCoRe** als Webapplikation um. Es wurden jedoch andere Technologien verwendet und einige Komponenten stärker ausgebaut. So ist es eine Frage der eigenen Kenntnisse und Vorlieben, welche der beiden Beispielanwendung als Basis für die eigene Anwendung dienen soll.
JSPDocPortal erfordert mehr JSP/JSTL Kenntnisse, erspart aber den extensiven Umgang mit XSL. In JSPDocPortal wurde jedoch besonders der Fokus auf die Modellierung und Integration bibliothekarischer Geschäftsprozesse gelegt. Der Anwender erhält eine voll funktionsfähige Applikation, die als Beispielszenario einen Publikations- und Dissertations-Server, wie er bei vielen potentiellen Nutzern vorkommen könnte, enthält.

Zur Funktionalität der Beispielanwendung gehören u.a.:

- An/Abmelden am System
- Sitemap
- Mehrsprachige Benutzeroberfläche (deutsch/englisch)
- Suchen in Dokumenbeständen (einfach, erweitert)
- Indexbrowser über Personen/Institutionen
- Browsen über Klassifikationen (Dokumentformate, Typen, DDC, BKL)
- Workflows zum Einstellen und Ändern von Autoren, Institutionen, Publikationen und Dissertationen
- Workflow zur Annahme von Nutzerregistrierungen
- Administrationsinterfaces für die jeweiligen Workflowkomponenten
- Administrationsinterface für Nutzer

Diese Beispielanwendung kann auf relativ einfache Weise an die eigenen Bedürfnisse angepasst und erweitert werden. Dazu ist zunächst einmal die Installation nötig.

2 Installationswege

Für die Installation der Beispielanwendung gibt es zwei Möglichkeiten - die automatische oder eine manuelle Installation.

2.1 Installation über das Installationsprogramm

Diese Installationsmethode steht zur Zeit nicht zur Verfügung.

Um einen ersten Eindruck vom System zu erhalten, wird von der Entwicklergruppe eine Binär-Distribution auf den Webseiten[1] bereitgestellt. Dabei handelt es sich um eine mit der Software IZPack generierte jar-Datei. Durch die Ausführung der Datei wird die Installation von JSPDocPortal gestartet. Als Ergebnis erhält man eine leere Anwendung, deren Daten in einer integrierten HSQL Datenbank verwaltet werden.
Als Webanwendungsserver wird Jetty verwendet. Eine Installationsanleitung ist in dem Dokument ***QuickInstallationGuideJSP*** zu finden. Weiterhin sind Beispieldaten in Distributionen thematisch zusammengestellt und können analog zum DocPortal dazu installiert werden.

2.2 Manuelle Installation

Dieser Abschnitt erläutert die manuelle Installation. Es wird erklärt, wie die Beispielanwendung JSPDocPortal schrittweise zu installieren ist und wie der Administrator per Konfiguration diese Anwendung in sein Zielsystem integriert. Hierbei ist zu unterscheiden, ob das Zielsystem unter den Betriebssystemen UNIX (Linux) oder MS Windows arbeitet. Auf die jeweiligen Unterschiede wird in den einzelnen Installationsschritten näher eingegangen. Die **Installation von MyCoRe,** die im MyCoRe User Guide (Kapitel2, Kapitel3)[2] ausführlich beschrieben ist, sollte an dieser Stelle bereits abgeschlossen sein oder jetzt durchgeführt werden.

Es müssen also die erforderlichen externen Softwareprodukte wie Java, Ant, Subversion... installiert sein. Weiterhin sollten die Umgebungsvariable gesetzt sein. Für JSPDocPortal sollten sie die Umgebungsvariable **$DOCPORTAL_HOME** auf das Verzeichnis *$MyCoReInstallDir/**jspdocportal*** setzen (nicht auf *docportal*). Anschließend sollte der MyCoRe-Kern aus dem Subversion Repository geladenen, installiert und compiliert sein, so dass am Ende die Datei *mycore.jar* im Verzeichnis *$MYCORE_HOME/lib* liegt.

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

[1] http://www.mycore.de/content/main/download/docportal_binary.xml
[2] http://www.mycore.de/content/main/documentation.xml

2.2.1 Download der Module vom Subversion-Repository

Derzeit stehen die Quellen von MyCoRe und der Beispielanwendung JSPDocPortal in dem Subversion-Repository server.mycore.de zur Verfügung. Der Download von JSPDpcoPortal erfolgt analog zu dem des MyCoRe-Kerns.

Wechseln Sie nun in das Verzeichnis, in welches Sie MyCoRe installieren haben, dieses Verzeichnis sollte nur mycore beinhalten.

Dieses Verzeichnis wird im folgenden *$MyCoReInstallDir* genannt.

Jetzt laden sie mit dem folgenden Kommandos die aktuelle Version der JSPDocPortal Anwendung herunter:

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

2.2.2 Die Verzeichnisstruktur

Bevor Sie mit der Übersetzung von JSPDocPortal beginnen, sollten Sie in Ihrem *$MyCoReInstallDir* die folgenden Ordner vorfinden:

jspdocportal – enthält die JSPDocPortal-Komponenten

mycore - enthält die MyCoRe-Komponenten

Nach dem Sie, wie in Abschnitt 2.2.3 beschrieben, die Webanwendung erfolgreich übersetzt haben, sollten sie folgende weitere Verzeichnisse vorfinden:

myapplication – dieser Ordner enthält alle *jspdocportal*-Verzeichnisse aber nur ihre eigenen Dateien, die von den Dateien im *jspdocportal* abweichen, wie z. B. Konfigurationsdateien möglicherweise aber auch JSP-Dateien, in denen Sie eigene Inhalte programmieren, die entweder zusätzliche Funktionalität beinhalten oder aber die namensgleichen JSP-Dateien im jspdocportal **überlagern** sollen.

Nach erfolgreicher Übersetzung befindet sich im *$MyCoReInstallDir* eine **WAR Datei**, in der die komplette Anwendung für die Installation auf einem Webserver gepackt ist.

myapplication/working – dieser Ordner enthält am Ende die jspdocportal-Dateien, mycore-Bibliotheken, ihre überlagernden Dateien aus dem *myapplication* Verzeichnis und die Webanwendung, als *webapps* Verzeichnis. Der Ordner stellt dann die komplette übersetzte Beispielanwendung dar.

Diese Verzeichnisstruktur wurde gewählt, um möglichst einfach und zu jedem Zeitpunkt eine sichtbare Trennung zwischen den eigenen Anpassungen und der Basisanwendung zu haben. Dadurch ist, neben einer größeren Übersichtlichkeit, ein Update auf eine neuere jspdocportal- und mycore- Version relativ einfach, da die eigenen Entwicklungen von den Dateien der Beispielanwendung sauber getrennt sind.

Wenn Sie für Ihre Anwendung Dateien anpassen wollen, müssen Sie diese vom Verzeichnis *jspdocportal* nach *myapplication* kopieren. Das heißt z. B., wenn die Navigation bei Ihnen ganz anders aussehen soll, als in dieser Beispielanwendung, was sehr wahrscheinlich ist, dann kopieren Sie die Datei *navigation.xml* von

$MyCoReInstallDir/jspdocportal/config in das Verzeichnis ihrer Applikation, also *$MyCoReInstallDir/myapplication/config*.

Der relative Pfad zu *myapplication* und *jspdocportal* muss gleich sein, da in den Installationsscripten von einem gleichen Basisverzeichnis (*$MyCoReInstallDir*) ausgegangen wird.

Im Verzeichnis *$MyCoReInstallDir/myapplication* können Sie anschließend alle Dateien nach Bedarf verändern oder editieren.

2.2.3 Übersetzen der einzelnen Komponenten

Erzeugen des Arbeitsverzeichnisses für die eigenen Anwendung

Um die eigentlichen Webanwendung zu erzeugen, sind folgende Schritte notwendig:

1. Editieren sie das Property MYAPPLICATION.DIRNAME in der Datei *$MyCoReInstallDir/jspdocportal* wie folgt:
   ```
   <property name="MYAPPLICATION.DIRNAME" value="myapplication" />
   ```
2. Im Verzeichnis *$MyCoReInstallDir/jspdocportal* führen Sie den Befehl
   ```
   ant create.my.own.application
   ```
   aus.

Es wird ein Verzeichnis *$MyCoReInstallDir/myapplication* erzeugt, das zunächst die Struktur des *jspdocportal* Verzeichnisses darstellt und einige Konfigurationsdateien enthält.

Im Anschluss daran können Sie das Verzeichnis *$MyCoReInstallDir/myapplication* in den Namen der eigenen Applikation umbenennen. In dieser Anleitung werden wir den Namen jedoch beibehalten.

Datenbanktreiber

Wenn Sie eine andere SQL-Datenbank (MySQL, DB2, ...) verwenden, kopieren Sie bitte die JAR-Datei mit dem Datenbanktreiber in das Verzeichnis *$MyCoReInstallDir/myapplication/lib*

Anpassung der Builddatei

Kopieren Sie zunächst die Datei *$MyCoReInstallDir/myapplication/build.xml.template* *nach* *$MyCoReInstallDir/myapplication/build.xml .*

Passen Sie nun das Property MYAPPLICATION_NAME in *$MyCoReInstallDir/myapplication/build.xml* wie folgt an:

```
<property name="MYAPPLICATION_NAME" value="myapplication" />
```

Anpassung der Hibernate-Konfigurationsdateien

Für MyCore muss in *$MyCoReInstallDir/myapplication/config/hibernate*

hibernate.cfg.xml.template *in* *hibernate.cfg.xml* und

für die Workflow-Engine muss in *$MyCoReInstallDir/myapplication/config/worflow*
`jbpm_hibernate.cfg.xml.template` in `jbpm_hibernate.cfg.xml`
umbenannt werden.

In beiden Dateien müssen nun die Parameter für die verwendete Datenbank konfiguratiert werden. Voreingestellt ist HSQLDB – ein Datenbanksystem, welches von MyCoRe mitgeliefert wird und für Testzwecke verwendet werden kann.

Jetzt sollten Sie (wenn Sie nicht HSQLDB) verwenden, die spezifizierten Datenbank anlegen (z.B. per SQL mit
CREATE DATABASE mycore und **CREATE DATABASE mycore-workflow**).

Eigene Properties

Viele Anpassungen können durch Modifikation der Properties in der Datei *mycore.properties* erreicht werden. Diese Datei wird während des Build-Prozesses automatisch generiert und aktualisiert. Wenn Sie Properties ändern möchten, tun Sie dies in der Datei *mymycore.properties.* Mittels des ANT-Tasks **merge.my.properties**, der auch vom Buildprozess verwendet wird, wird aus dieser Datei und der Datei *mycore.base.properties* die Datei *mycore.properties* zusammengesetzt. Properties in *mymycore.properties* überschreiben dabei Properties aus der Basisdatei.

Properties, die Sie unbedingt anpassen sollten, finden Sie in der Datei *mycore.properties.template. Diese Datei können Sie als Basis für mymycore.properties nehmen.*

Dieses Vorgehen hat den Vorteil, dass Sie nicht, wenn sich Basisproperties ändern, Ihre Anpassungen wieder neu von Hand einpflegen müssen.

Diese Ant-Targets sollten Sie nach Möglichkeit nicht noch einmal aufrufen müssen.

Folgende Properties müssen unbedingt angepasst werden:

```
MCR.datadir=c:/workspaces/mycoresample/working_content
MCR.basedir=c:/workspaces/mycoresample/projects/jspdocportal
MCR.baseurl=http://localhost:8080/myapplication
MCR.WorkflowEngine.Administrator.Email=admin@mycore.de
MCR.mail.server=your.mailserver.com
MCR.mail.protocol=smtp
MCR.mail.debug=false
```

MCR.basedir ist das Verzeichnis, in das Sie das JSPDocportal ausgecheckt haben.
MCR.datadir ist das Verzeichnis, in welchem die Anwendung eigene Dateien (z.B. Suchindexe, temporäre Kopien von Objekten, ...) speichern kann.

Builden der Anwendung (mit ANT)

Wechseln Sie anschließend in das Verzeichnis ihrer Applikation (*$MyCoReInstallDir/myapplication*) und führen Sie dort folgende ANT-Befehle aus:

ant info
Prüfen der Systemumgebung in myapplication (Pfadangabe, Variable...)

ant merge.my.properties
Erzeugen der aktuellen Konfigurationsdateien

ant create.schema
XML Schema Dateien für die im Konfigurationsverzeichnis liegenden Datenmodelle generieren.

ant jar
Bilden der Java Bibliothek docportal-for-<db-Backend>.jar

ant create.directories
Erzeugen aller notwendigen Arbeitsverzeichnisse für Metadaten- und Volltextindex, Workflow...

ant create.genkeys
Signaturschlüssel für das UploadApplet erzeugen

ant hsqldbstart
Nur wenn Sie HSQLDB als Datenbank (default) gewählt haben, müssen Sie diese explizit starten und auch stoppen bevor sie auf die DB zugreifen.

ant create.metastore
Anlegen der Datenbankstrukturen

ant create.class
Laden aller Klassifikationen aus dem Verzeichnis content/classification

ant create.users
Laden der Zugriffsdefinitionen, Gruppen und Nutzer

ant create.workflowengine.database
Anlegen der Datenbank für die Workflow-Engine

ant deploy.workflow.processdefinitions
Laden der Prozessdefinitionen aus dem config/workflow Verzeichnis

ant hsqldbstop
Stop der HSQL DB (vor dem Start der Webapplikation muss HSQL natürlich erneut gestartet werden)

ant create.scripts
Erstellen der Scripts um im /bin Verzeichnis das MyCoRe Command Line Interface (CLI) nutzen zu können

ant war
Bilden der Webapplikation und daraus das entsprechende WAR-Archiv

Das im Verzeichnis *$MyCoReInstallDir* erzeugte WAR-File kann nun auf einem J2EE-Applikationsserver gestartet werden (Tomcat, Jetty, ...)

Nach diesen Schritten ist ein weiteres Verzeichnis *$MyCoReInstallDir/myapplication/working* angelegt worden, dass die gesamte Webanwendung enthält. In diesem Verzeichnis sollten Sie keine Änderungen vornehmen, da diese immer wieder überschrieben werden. Das MyCoRe CLI kann hier aus dem *$MyCoReInstallDir/myapplication/working/bin* Verzeichnis heraus gestartet werden.

Alternativ zu **ant war** können Sie auch mit **ant webapps** nur das Webapplikationsverzeichnis in *$MyCoReInstallDir/myapplication/working* erzeugen.

Optional: Laden der Beispiel-Inhalte

Wenn Sie HSQLDB verwenden, starten sie die Datenbank mit **ant hsqldbstart.**

Dann müssen die Klassifikationen eingespielt werden. Dazu kopieren Sie alle Dateien aus *$MyCoReInstallDir/jspdocportal/content/classification/ignore* nach *$MyCoReInstallDir/myapplication/content/classification* und führen danach **ant create.class** im Verzeichnis *$MyCoReInstallDir/myapplication* aus.

Jetzt können Sie mit **ant load_default_content** die Beispiele laden

3 Zugriffsberechtigungen (Access Control System)

Um eigene Dokumente einstellen zu können, müssen Nutzeraccounts mit den entsprechenden Rechten eingerichtet werden. Während des Installationsprozesses sind Standardaccounts angelegt worden.

Die Zugriffsrechte auf die digitalen Objekte sind von den Metadaten getrennt und werden in einer SQL-Datenbank gespeichert. Beim Erstellen der digitalen Objekte werden Defaultregeln angelegt. Innerhalb des Workflows hat der Autor/Editor die Möglichkeit über einen ACL-Editor (ACL = Access Control Level) die Zugriffsrechte auf sein Objekt selbst zu setzen.

3.1 Aktionsgebundene Berechtigungen - Permissions

Aktionsgebundene Berechtigungen gelten nicht für einzelne digitale Objekte sondern für allgemeine Aktionen. Sie werden in der Datei config/user/permissions.xml defaultmäßig definiert und können beliebig erweitert oder geändert werden.

Lediglich die Bezeichnungen bereits vordefinierter Berechtigungen wie z.B. „create-document" sollten nicht geändert werden, da sie im MyCoRe-Code explizit abgefragt werden.

Die globalen Berechtigungen in JSPDocPortal weichen teilweise von denen im DocPortal ab.

Berechtigung	***Beschreibung***
administrate	Beinhaltet alle Administratoren und Editoren mit Teiladminsitrationsrechten
administrate-institution	Verwalten von Prozessen für Institutionen (WF)
administrate-xmetadiss	Verwalten von Prozessen für Dissertationen (WF)
administrate-publication	Verwalten von Prozessen für Publikationen (WF)
administrate-author	Verwalten von Prozessen für Autoren (WF)
administrate-accessrules	Verwalten von Zugriffsrechten für Dokumente (WF)
administrate-registeruser	Verwalten von Nutzer Neuregistrierungen (WF)
administrate-webcontent	Editieren der statischen WebContent Dateien via FCK
delete-user	Löschen von Benutzern
create-group	Anlegen von Gruppen
modify-group	Verändern von Gruppen
delete-group	Löschen von Gruppen
create-classification	Erzeugen einer Klassifikation
create-author	Erzeugen eines Autors
create-document	Erzeugen eines Dokuments

create-disshab	Erzeugen einer Dissertation
admininterface-access	Zugang zum Administrationsinterface
admininterface-user	Zugang zur Benutzerverwaltung
admininterface-accessrule	Zugang zum Regeleditor

Über das ANT-Target create.users werden diese Permissions initial geladen.

Zuweisen aktionsgebundener Zugriffsrechte (permissions) über CLI

Wenn Sie nachträglich Permissions hinzufügen möchten, können Sie dies tun, indem sie eine eigene *permissions.xml* Datei in Ihr *myapplication/config/users* Verzeichnis legen und diese Permissions dann über das CLI laden. Dafür gibt es folgende Kommandos:

```
load permissions data from file {0}
```

Das Kommando lädt alle globalen Zugriffsrechte aus der Datei {0}.

```
update permissions data from file {0}
```

Ein Update der Permissions mit den Daten aus dem File {0} wird ausgeführt.

```
list all permissions
```

Zeigt alle statischen Permissions im CLI an.

```
delete all permissions
```

Das Kommando löscht alle statischen Permissions.

```
save all permissions to file {0}
```

Alle globalen Zugriffsrechte werden in der Datei {0} gespeichert.

3.2 Objektgebundene Zugriffsrechte

Lese- und Schreibrechte werden für jedes Objekt vergeben. JSPDocPortal nutzt dabei folgende Berechtigungstypen:

- Lesen (**read**),
- Schreiben (**writedb**),
- Löschen vom Workflow (**deletewf**)
- Löschen vom Server (**deletedb**)

Für die Workflowkomponenten sind Zugriffsregeln über Properties in der Datei *mycore.properties* vordefiniert. Diese Zugriffsregeln beziehen sich auf die vordefinierten Gruppen und müssen eventuell bei einer Änderung dieser auch angepasst werden. Weitere Informationen dazu finden sie im Kapitel zur Workflowkomponente.

Zuweisen einer Zugriffsregel über das CLI

Zum Ändern einer Zugriffsregel außerhalb des Workflows stehen mehrere Funktionen im CLI zur Verfügung:

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

Damit weisen Sie dem Objekt mit der ID {1} für den Berechtigungstyp {0} eine Berechtigungsregel {2} zu.

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

Damit weisen Sie allen Objekten eines Dokumenttyps {1} für den Berechtigungstyp {0} eine Berechtigunsregel {2} zu

Bsp: `update permission read for id DocPortal_document_00000001 with rulefile C:/temp/rule.xml`

Syntax einer Zugriffsregel

```
<condition format="xml">
 <boolean operator="and">
    <boolean operator="or">
       <condition field="group" operator="=" value="admingroup"  />
       <condition  field="user" operator="=" value="author1A" />
        <condition field="ip" operator="="
                                value="129.187.87.131/255.255.255.0" />
    </boolean>
    <boolean operator="or">
       <condition value="12.04.2007" operator="&lt;=" field="date"/>
    </boolean>
 </boolean>
</condition>
```

Über die booleschen Operatoren **and**, **or** und **not** lässt sich die Zugriffsberechtigung beliebig komplex zusammensetzen. Als mögliche Bedingungstypen stehen **group**, **user**, **ip** und **date** zur Verfügung. Die angegebene Beispielregel gewährt genau dann eine Berechtigung, wenn das aktuelle Datum kleiner ist als der 12. April 2007 und wenn der aktuelle Anwender entweder der User „author1A" ist oder der Gruppe „admingroup" angehört oder seine Anfrage von der ip-addresse/subnetmask 129.187.87.131/255.255.255.0 aus stellt.

Die einfachste Regel, die immer Zugriff gewährt, d.h. **true** zurück liefert, lautet:

```
<condition format="xml">
 <boolean operator="true" />
</condition>
```

4 Benutzerverwaltung

4.1 Gruppen und Benutzer der Beispielanwendung

Die Beispielanwendung bringt zur Demonstration bereits eine Reihe von Benutzern und Gruppen mit. Einer Gruppe können dabei mehrere Benutzer angehören. Aber auch ein Nutzer kann mehreren Gruppen zugeordnet sein.
Die Vergabe von Rechten auf Objekte kann dann sowohl für einen einzelnen Benutzer als auch für eine ganze Gruppe erfolgen. Die Benutzer und Gruppen werden beim Erstellen der Webanwendung mit dem ant-Target **create.users** erzeugt.

Bearbeiten von Gruppen über CLI

Die Anpassung und Änderung der Benutzer und Gruppen können Sie über die Kommandos des CLI realisieren. Eigene Benutzer- und Gruppendefinitionen sollten im Verzeichnis *myapplication/config/users* abgelegt werden.
Eine detaillierte Beschreibung der CLI - Kommandoaufrufe für die Nutzerverwaltung finden Sie im MyCoRe User Guide (Kapitel 5.2.2)

Vordefinierte Gruppen

Im JSPDocportal sind folgende Gruppen vordefiniert: (Diese sind vor allem zur Demonstration der Workflowkomponenten notwendig.)

Gruppe	Beschreibung
root	Administration der gesamten Anwendung
admingroup	Administration der gesamten Anwendung
adminauthor	Administration der Autorendaten.
adminclassification	Administration der Klassifikationen.
admindisshab	Administration der Dissertationen.
admininstitution	Administration der Institutionsdaten.
adminpublication	Administration der Publikationen.
adminuser	Administration der Nutzer, Gruppen und Zugriffsrechte.
adminwebcontent	Administration der statischen Textinhalte.
createauthor	Erstellen von Autorendaten
createdisshab	Erstellen von Dissertationen.
createpublication	Erstellen von Publikationen.
createinstitution	Erstellen von Institutionendaten.
gastgroup	Suchen/Browsen in Dokumentenbestand. (Default)

Tabelle 4.1: Beispielgruppen in JSPDocPortal

Vordefinierte Benutzer

Im JSPDocPortal sind folgende Benutzer vordefiniert: (Für die Workflowkomponente ist damit für jede Rolle ein Beispielnutzer festgelegt. Für die eigene Anwendung sollten Sie die Nutzer zumindest bezüglich des Passwortes manipulieren.)

Benutzer	Passwort	Mitglied in Gruppe
administrator	alleswirdgut	admingroup
root	alleswirdgut	rootgroup
authorA	authorA	createauthor
authorP	authorP	createpublication
authorI	authorI	createinstitution
authorD	authorD	createdisshab
editorA	editorA	adminauthor
editorP	editorP	adminpublication
editorI	editorI	admininstitution
editorD	editorD	admindisshab
editoruser	editoruser	adminuser
editorclass	editorclass	adminclassification
editorweb	editorweb	adminwebcontent
gast	gastgroup	-- (default user)

Tabelle 4.2: Beispielbenutzer in JSPDocPortal

4.2 Anbindung einer externen Nutzerverwaltung

Zusätzlich zur Nutzerverwaltung durch MyCoRe kann die Überprüfung der Zugangsdaten der Nutzer auch durch ein externes System durchgeführt werden. Dies könnte beispielsweise eine Datenbank oder ein ActiveDirectory sein. Die Anbindung ist in Java selbständig zu programmieren. Die Verwaltung der Zugriffsrechte sowie Gruppenzugehörigkeiten wird aber weiterhin durch MyCoRe realisiert.

Die neue Klasse muss das Interface **org.mycore.user2.MCRExternalUserLogin** implementieren. Es enthält die Methoden:

- *public boolean loginUser(String userID, String password)*
  NutzerID und Passwort werden gegen das externe System geprüft und es wird „true" zurückgegeben, wenn das Login erfolgreich ist.

- *public String retrieveMyCoReUserID(String userID, String password)*
  gibt für eine externe NutzerID die MyCoReUserID zurück

- *public String retrieveMyCoRePassword(String userID, String passwd)*
  gibt für einen externe NutzerID das MyCoRE Nutzerpasswort zurück

- *public void updateUserData(String uid, String pwd, MCRUser mcrUser)*
  liest die Nutzerdaten aus dem externen System und schreibt sie in das MyCoReUser Objekt.

- *public String checkUserID(String userID)*
  überprüft, ob die übergebene ID existiert und gibt eine Statusmeldung zurück. (z.B. Nutzer XY existiert, Nutzer XY wurde gesperrt, ...)

Anschließend wird mit dem Property *MCR.Application.ExternalUserLogin.Class* der Name der Klasse registriert in *mycore.properties* registriert. Sie wird in der Anwendung dynamisch geladen.
(z.B.: `MCR.Application.ExternalUserLogin.Class =de.hsnb.digibib.user.HSNBLBSUserLogin`)

5 Konfiguration Ihrer eigenen Anwendung

5.1 Internationalisierung

Die Internationalisierung der Anwendung erfolgt nach dem Java Standard **i18n.**
Für jede Sprache gibt es zwei Ressource-Dateien, die wie gewöhnliche Java Property-Dateien aus Key-Value-Paaren aufgebaut sind. Alle Sprachdateien finden Sie im Verzeichnis *jspdocportal/languages* bzw. *myapplication/languages*.
Die mehrsprachigen Ausdrücke werden durch Angabe der Key-Attribute in JSTL-Tags oder über Java-RessourceBundles integriert.

Die Sprachdateien **messages_[de|en|*].properties** enthalten die Definitionen, die in der Beispielanwendung verwendet werden.

In jspdocportal/languages/messages_de.properties werden deutsche Übersetzungen gepflegt: **Bsp**: `contributors=Beteiligte`

In jspdocportal/languages/messages_en.properties werden englische Übersetzungen gepflegt: **Bsp**: `contributors=Contributors`

Über das JSTL-Tag: `<fmt:message key="contributors" />`

in der JSP-Datei, wird zur Laufzeit die aktuelle Sprache ermittelt und der entsprechende Wert ausgegeben.

In **mymessages_[de|en|*].properties** können Sie eigene Werte definieren oder Werte der Standardanwendung überschreiben. Im Build-Prozess wird diese Datei automatisch in die zugehörige **messages_[de|en|*].properties** integriert.
Dadurch ist es nicht notwendig alle Sprach-Properties in die eigene Applikation (*myapplication/languages*) zu übernehmen. In *mymessages.properties* nehmen sie nur die eigenen und die Sprachproperties auf, die sie anders setzen wollen. Bei Weiterentwicklungen am JSPDocPortal bleiben Sie dadurch unabhängig und können neuere Sprachdateien übernehmen ohne ihre eigenen Einstellungen zu verlieren.

5.2 Definition der Navigation

Die gesamte Navigation der Webapplikation wird in der Datei *navigation.xml* im Konfigurationsverzeichnis definiert.

Alle Seiten, denen über das *Navigationservlet* ein Template zugeordnet wird, müssen in der Navigationsdatei definiert werden. Als Beispiel für eine Definitionsdatei geben wir hier einen kleinen Ausschnitt an:

```
<?xml version="1.0" encoding="UTF-8"?>
 <navigations>
  <navigation name="left" label="Nav.MainmenueLeft">
   <navitem name="left" label="Nav.MyCoRe" href="content/index.jsp">
    <navitem name="search" label="Nav.Search" href="content/search.jsp">
```

```
    <navitem name="allmeta" label="Nav.SearchAllMetadataFields"
            href="content/searchmask-editor.jsp?editor=
                  editor/searchmasks/SearchMask_AllMetadataFields.xml"
            hidden="true" >
     <refitem name="~searchstart-allmeta" label="" />
     <navitem name="searchresult-allmeta" label="Nav.Searchresults "
             href="content/searchresult.jsp" hidden="true">
       <refitem name="~searchresult-allmeta" label="" />
     </navitem>
    </navitem>
    <navitem name="simple" label="Nav.SimpleDocumentSearch"
            href="content/searchmask-editor.jsp?editor=
                  editor/searchmasks/SearchMask_SimpleDocument.xml">
     <refitem name="~searchstart-simple" label="" />
     <navitem name="searchresult-simple" label="Nav.Searchresults"
             href="content/searchresult.jsp" hidden="true">
      <refitem name="~searchresult-simple" label="" />
     </navitem>
    </navitem>
   ...
  </navitem>
 </navigation>
</navigations>
```

<navigations> ist das Wurzelelement.

Mit einem <navigation>-Element wird eine Navigationsleiste/Menü durch Setzen des Namens- und des Label-Attributs definiert. In der Beispielanwendung sind die Navigationsleisten **left**, **top** und **admin** definiert.

Über <navitem> werden die einzelnen Navigationselemente innerhalb eines Navigationsmenüs definiert. Jeder Endknoten ist einer Seite zugeordnet. Diese hat eine strukturelle ID, die sich aus den ID's der <navitem>-Elemente entlang des Pfades im XML-Baumes ergibt. Daneben kann man jeder Seite auch zusätzlich einen Referenznamen (<refitem>) zuordnen. Als Konvention müssen Referenznamen in jspdocportal mit der Tilde '~' beginnen.

Die Seite mit dem Label „Nav.SearchAllMetadataFields" in obigem Beispiel hat den Referenznamen „~searchstart-allmeta" und die strukturelle ID „left.search.allmeta".

Über strukturelle ID oder Referenznamen kann eine Seite vom Navigationsservlet *(jspdocportal/sources/org/mycore/frontend/jsp/NavServlet.java)* aufgerufen werden. Wir können die entsprechende Seite in unserem Beispiel also auf zwei Arten erhalten.

- *http://$WebApplicationBaseURL/nav?path=left.search.allmeta*
- *http://$WebApplicationBaseURL/nav?path=~searchstart-allmeta*

Die Referenznamen sind dabei bis auf die Referenzen innerhalb des Klassifikations-Browsers frei wählbar, der Anwender sollte aber auf Eindeutigkeit achten.

Referenzen auf Suchmasken

Referenznamen für JSP-Seiten, die den Aufruf einer Suchmaske beinhalten, müssen einen Referenznamen besitzen, der sich aus

searchstart+[Bezeichner] zusammensetzt

und das zugehörige Resultlist-Navitem muss den entsprechenden Referenznamen *searchresult*+[Bezeichner] besitzen.

Bsp: Aufruf der Seite die eine Suche über alle Metadatenfelder beinhaltet.

Navitem: <navitem name="allmeta" label="Nav.SearchAllMetadataFields" ....

Referenzname der Suchmaske: ~searchstart-allmeta

Referenzname der Ergebnisliste: ~searchresult-allmeta

5.3 Das Layout der Applikation

Wenn Seiten, die in der Navigationsdatei *navigation.xml* definiert sind, über das *Navigationsservlet* eingebunden werden, das erkennt man in der URL anhand des Ausdrucks „nav?path=", dann wird um die entsprechende Seite ein HTML-Rahmen gelegt, der von der Datei *frame.jsp* (in *jspdocportal/webpages*) definiert wird. Im Allgemeinen werden alle Seiten der Webapplikation in diesen Frame eingebettet.

Wenn Sie also das Layout der eigenen Anwendung anpassen möchten, müssen Sie die Datei *frame.jsp* in ihr Applikationsverzeichnis (*myapplication/webpages*) übernehmen und dort entsprechend ändern. Für Farben und rein optische Layout-Anpassungen genügt es eventuell, die CSS-Stylesheets zu ändern, die Sie im Verzeichnis *jspdocportal/webpages/css* finden.

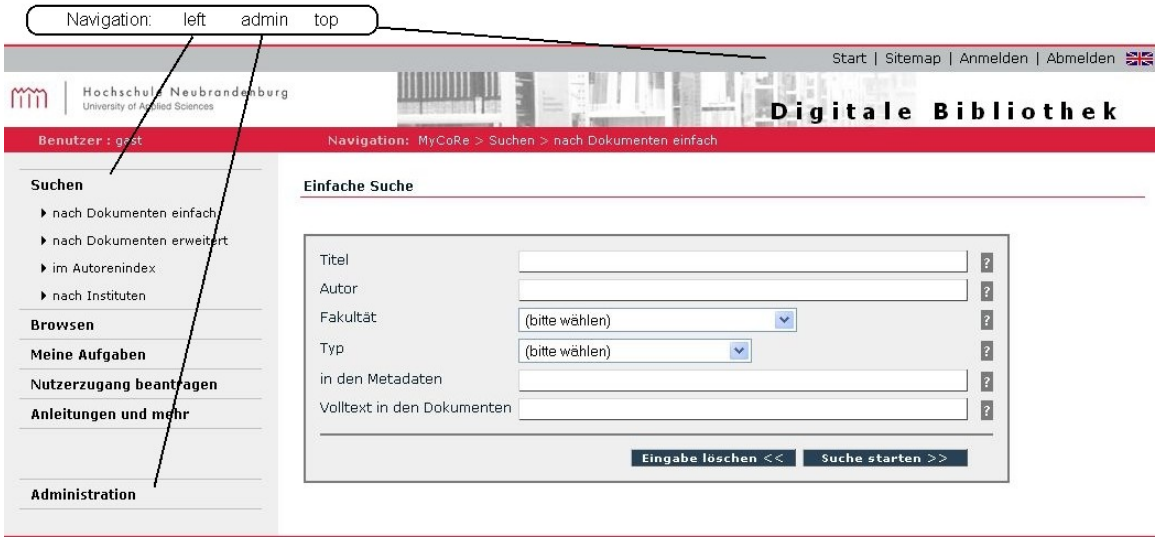


Abbildung 5.1: Anwendung mit eigenem Layout - mit eigener frame.jsp und css

5.4 Einzeltrefferanzeige

Den Teil der Metadaten und die Reihenfolge, die Sie in den Trefferseiten anzeigen wollen, können Sie für jeden Datentyp per Konfiguration bestimmen. Die Konfigurationsdateien befinden sich in *jspdocportal/webpages/content/results-config*.

Es existiert für jeden Datentyp eine XML-basierte Beschreibung der auszugebenden Metadaten, sowohl für die Einzeltrefferanzeige als auch für die Darstellung in der Ergebnisliste einer Suche.

Die Namen der Konfigurationsdateien habe das Format: *docdetail-<datentyp>.xml*

Die Daten werden in der definierten Reihenfolge ausgegeben. Es werden natürlich nur die Felder angezeigt, die auch Werte besitzen.

Bsp. einer Einzeltrefferanzeige: *docdetails-document.xml* (Auszug)

```
<MCRDocDetails name="field definition for documentdetails" >
 <MCRDocDetail rowtype="standard" labelkey="OMD.identifiers" >
   <MCRDocDetailContent
       xpath="/mycoreobject/metadata/identifiers/identifier"
       templatetype="tpl-text-values" separator=", "
       languageRequired="no" />
 </MCRDocDetail>
 <MCRDocDetail rowtype="standard" labelkey="OMD.author" >
   <MCRDocDetailContent
       xpath="/mycoreobject/metadata/creatorlinks/creatorlink"
       templatetype="tpl-author_links" separator="; " terminator="; "
       languageRequired="no" />
   <MCRDocDetailContent
       xpath="/mycoreobject/metadata/creators/creator"
       templatetype="tpl-text-values" separator="; "
       languageRequired="no" />
 </MCRDocDetail>
 <MCRDocDetail rowtype="standard" labelkey="OMD.class-origins" >
 <MCRDocDetailContent
       xpath="/mycoreobject/metadata/origins/origin"
       templatetype="tpl-classification" separator=", "
       languageRequired="yes" />
 </MCRDocDetail>
 <MCRDocDetail rowtype="standard" labelkey="OMD.date" >
   <MCRDocDetailContent
       xpath="/mycoreobject/metadata/dates/date[@type='create']"
       templatetype="tpl-date-valuesMinimal" separator=", "
       languageRequired="yes" introkey="datecreated" />
 </MCRDocDetail>
 <MCRDocDetail rowtype="space" labelkey="" />
 <MCRDocDetail rowtype="standard" labelkey="OMD.documents" >
   <MCRDocDetailContent
       xpath="/mycoreobject/structure/derobjects/derobject"
       templatetype="tpl-alldocument"
       separator="br" languageRequired="no" />
  </MCRDocDetail>
       ...
</MCRDocDetails>
```

5.4.1 <MCRDocDetail>

Für **MCRDocDetail –** sind folgende Attribute möglich:

rowtype = {"standard"|"image"|"children"|"table"|"space"|"line"}

Der Row-Type wird auch im XML-Output als Attribut 'type' zur Verfügung gestellt. Über Ihn kann die Ausgabe (Layout) gesteuert werden.Der Row-Type **table** dient z. B. dazu die Ausgaben in Tabellenform darzustellen. Wenn Sie den Rowtype **image** für ein Ausgabeelement wählen, sollten sich dann in der Ausgabe auch eine Bilddatei

befinden, die dann angezeigt wird. Die Row-Types **line** und **space** dienen zur Darstellung einer Trennlinie bzw. einer leeren Zeile.

labelkey = {Sprachpropertie}

Über den Labelkey wird die Spaltenbezeichnung in der Ausgabe festgelegt, es enhält den Key für ein Property aus den messages_[de|en|*] Dateien, das zur Laufzeit sprachabhängig eingebunden wird.

Für `labelkey="OMD.author"` finden wir also in

jspdocportal/languages/ messages_de.properties: *OMD.author=Autor*

5.4.2 <MCRDocDetailContent>

Ein XPATH-Ausdruck weist auf das auszuwertende Element im XML-Metadatensatz. Je nach Typ wird der Text, ein Link auf Kindobjekte, Klassifikationsdaten (Label, Kategorie und Klassifikations-ID) oder Metadaten zu den zugehörigen Daten des Dokuments ausgegeben. Mit dem separator-Attribut gibt man ein Trennzeichen an, falls man mehrere Treffer erhält. Über das Attribut languageRequired lässt sich die Sprachabhängigkeit der Anzeige aktivieren. Das Attribut „templatetype" beschreibt den Typ des Metadatenfeldes.

Für **MCRDocDetailContent** sind folgende Attribute möglich:

xpath = {Xpath-Ausdruck des MyCoRe Metadatentags}

Das Attribut „xpath" gibt die Stelle im Metadatenobjekt an, auf welche das Template angewendet wird.

[introkey = {Sprachproperty}]

Optional: - Über den Introkey kann eine zusätzliche Bezeichnung für den Inhalt des Metadatenfeldes festgelegt werden. Dies ist z. B. sinnvoll, wenn die Xpath-Ausdrücke attributiert sind und Sie Werte mit verschiedenen Bedeutungen anzeigen wollen.

Ein Anwendungsfall wäre z. B. die Darstellung von Lebensdaten (labelkey) mit Geburtstag und Todestag als introkey vor den anzuzeigenden Metadatenfeldern.

Bsp:

```
<MCRDocDetail rowtype="standard" labelkey="OMD.lifedates" >
  <MCRDocDetailContent introkey="birthday"
    xpath="/mycoreobject/metadata/dates/date[@type='birth']".../>
  <MCRDocDetailContent introkey="dayofdeath"
    xpath="/mycoreobject/metadata/dates/date[@type='death']".. />
</MCRDocDetail>
```

templatetype = {"tpl-author_links"|"tpl-text-messagekey"
|"tpl-text-values"|"tpl-classification"|"tpl-date-values"

|"tpl-document"|"tpl-alldocument"|"tpl-boolean"

|"tpl-address"|"tpl-email"|"tpl-metalink"

|"tpl-authorjoin"|"tpl-urn"|"tpl-image"|"tpl-child"}

Über den Template-Type wird ausgedrückt, auf welche Art und Weise der Inhalt des Metadatenfeldes behandelt werden soll und was als Ausgabe erzeugt werden soll.

Als Anwender sollten Sie mit den vorgegebenen, im Folgenden aufgeführten, Template-Typen auskommen. Benötigen Sie einen weiteren Typ, stellen Sie einfach die Anforderung für den neuen Template-Type an die MyCoRe-Community.

tpl-document oder tpl-alldocument

Zeigt die Derivat-Daten des Objekts an. Bei „tpl-alldocument" werden alle digitalen Dateien angezeigt, bei „tpl-document" nur das jeweilige Hauptdokument (maindoc) eines Derivats.

Bsp:
```
<MCRDocDetail rowtype="standard" labelkey="documents" >
  <MCRDocDetailContent
        xpath="/mycoreobject/structure/derobjects/derobject"
        type="document" templatetype="tpl-document"
        separator="br" languageRequired="no" />
</MCRDocDetail>
```

tpl-classification erzeugt zum zugehörigen Metadatenfeld (ein MCRMetaClassification Tag) die Ausgabe von Label, Kategorie und Klassifikations-ID.

Bsp:
```
<MCRDocDetail rowtype="standard" labelkey="class-origins" >
  <MCRDocDetailContent
        xpath="/mycoreobject/metadata/origins/origin"
        templatetype="tpl-classification" separator=", "
    languageRequired="yes" />
</MCRDocDetail>
```

tpl-text-values interpretiert das Metadatenfeld als Text und liefert gezielt den mit Xpath addressierten Ausdruck

Bsp:
```
<MCRDocDetail rowtype="standard" labelkey="descriptions" >
    <MCRDocDetailContent
         xpath="/mycoreobject/metadata/descriptions/description"
        templatetype="tpl-text-values" separator=", "
        languageRequired="yes" escapeXml="false" />
</MCRDocDetail>
```

tpl-date-values(Minimal|Standard) formatiert ein Datum. Bei Angabe von Minimal oder Standard, erfolgt die Ausgabe so, wie sie durch die Properties *MCR.Dateformat.Minimal* oder *MCR.Dateformat.Standard* in *jspdocportal/languages/messages_de.properties* definiert wurde.

Bsp:
```
<MCRDocDetail rowtype="standard" labelkey="date" >
    <MCRDocDetailContent
        xpath="/mycoreobject/metadata/dates/date[@type='create']"
        templatetype="tpl-date-values" separator=","
        languageRequired="no"
        introkey="datecreated" />
```

```
</MCRDocDetail>

<MCRDocDetail rowtype="standard" labelkey="date" >
    <MCRDocDetailContent
         xpath="/mycoreobject/metadata/dates/date[@type='create']"
         templatetype="tpl-date-valuesMinimal" separator=", "
         languageRequired="yes" introkey="datecreated" />
</MCRDocDetail>
```

tpl-author_links holt Vorname und Zuname eines mit MCRMetaLinkID angegebenen Autoren

Bsp:
```
<MCRDocDetail rowtype="standard" labelkey="author" >
  <MCRDocDetailContent
       xpath="/mycoreobject/metadata/creatorlinks/creatorlink"
       templatetype="tpl-author_links" separator=", "
       languageRequired="no" />
   <MCRDocDetailContent
       xpath="/mycoreobject/metadata/creators/creator"
       templatetype="tpl-text-values" separator=", "
       languageRequired="no" />
</MCRDocDetail>
```

tpl-authorjoin Erzeugt einen Link auf eine Trefferseite mit den Dokumenten eines Autors. **Achtung!** Hier verstecken sich in Xpath zwei Parameter. Vor dem Komma wird die ID des Autors mit übergeben.

Bsp:
```
<MCRDocDetail rowtype="standard"
   labelkey="OMD.documents-by-author" >
  <MCRDocDetailContent
       xpath="/mycoreobject/@ID,
              /mycoreobject/metadata/names/name/surname"
       templatetype="tpl-authorjoin" separator="" terminator=""
       languageRequired="no" />
</MCRDocDetail>
```

tpl-boolean Für die Anzeige von Metadaten des Formats MCRMetaBoolean.

Bsp:
```
<MCRDocDetail rowtype="standard" labelkey="gender" >
  <MCRDocDetailContent
       xpath="/mycoreobject/metadata/females/female"
       templatetype="tpl-boolean" separator=""
       languageRequired="no" />
</MCRDocDetail>
```

tpl-address Für die Anzeige von Metadaten des Typs MCRMetaAddress.

Bsp:
```
<MCRDocDetail rowtype="standard" labelkey="OMD.addressofficial">
   <MCRDocDetailContent
      xpath="/mycoreobject/metadata/addresses/
             address[@type='office']"
      templatetype="tpl-address"
       separator="&lt;br&gt;   "
       languageRequired="yes" introkey="OMD.spaces"
       escapeXml="true"/>
</MCRDocDetail>
```

tpl-email Für die Anzeige einer E-Mail-Addresse

Bsp:
```
<MCRDocDetail rowtype="standard" labelkey="email" >
```

```
        <MCRDocDetailContent
            xpath="/mycoreobject/metadata/emails/email"
            templatetype="tpl-email" separator=", "
            languageRequired="yes" />
      </MCRDocDetail>
```

tpl-urn Für die Anzeige einer URN.

```
Bsp: <MCRDocDetail rowtype="standard" labelkey="urns" >
       <MCRDocDetailContent
            xpath="/mycoreobject/metadata/urns/urn"
            templatetype="tpl-urn" separator=", "
            languageRequired="no" />
   </MCRDocDetail>
```

tpl-image Bindet ein Bild in die Metadaten-Anzeige ein, soweit es vorhanden ist.

tpl-text-messagekey kennzeichnet für die Ausgabe, das dieses Feld als MessageKey interpretiert werden soll.

tpl-child erzeugt eine Liste der Kindobjekte des MyCoRe-Objekts.

separator = {Strukturierender Text}

mit dem Separator wird, falls mehr als nur ein Ergebniswert vorliegt, festgelegt, mit welchem Trennzeichen die einzelnen Werte voneinander abgegrenzt werden sollen.

languageRequired={"yes"|"no"}

ist das Attribut auf **"yes"** gesetzt, wird das im Metadatenfeld vorhandene lang-Attribut bei der Auswahl berücksichtigt.

Hinweise für den Entwickler:

Die Konfiguration wird auf das auszugebende MyCoReObjekt angewendet und erzeugt ein relativ einfach strukturiertes JDOM Objekt. Dies geschieht über den Aufruf des JSTL Tags aus der mycore-Tag Library in der Datei *docdetails.jsp* im Verzeichnis *jspdocportal/webpages/content*.

```
<mcr:docDetails mcrObj="${mycoreobject}"
     var="docDetails" lang="${requestScope.lang}" style="${style}" />
```

Sie können sich mit Hilfe eines debug-Parameters in der URL zur Einzeltrefferanzeige,

(*.../nav?path=~docdetail&debug=true&id=DocPortal_document_00000001*)

den XML-Output auch anzeigen lassen. (Das ist z. B. sehr nützlich, wenn Sie für eigene Datentypen auch eine eigene Ausgabekonfiguration haben wollen. Sie müssen für den neuen Datentyp die entsprechende Konfigurationsdatei **docdetail_<datentyp>.xml** im Verzeichnis *myapplication/webpages/result-config* ablegen.

Erzeugt wird folgender XML-Output (Auszug):

```
<?xml version="1.0" encoding="UTF-8"?>
<all-metavalues ID="DocPortal_document_00000001" docType="document">
  <metaname name="OMD.author" type="standard">
    <metavalues type="linkedCategory" separator="; "
                terminator="; " introkey="" escapeXml="true">
      <metavalue href="" text="Anton Alpha" />
      <metavalue href="" text="Bertha Bravo" />
    </metavalues>
    <metavalues type="linkedCategory" separator="; "
                terminator=", " introkey="" escapeXml="true" start="1">
      <metavalue href="" text="Stephan, Robert" />
    </metavalues>
  </metaname>
  <metaname name="OMD.class-origins" type="standard">
    <metavalues     type="linkedCategory" separator=", "
                    terminator=", " escapeXml="true">
      <metavalue href="http://www.uni-duisburg-essen.de/" target="new"
             text="Universität Duisburg-Essen"
             classid="DocPortal_class_00000002" categid="Unis.Essen" />
    </metavalues>
  </metaname>
  <metaname type="space" />
  <metaname name="OMD.documents" type="standard">
    <digitalobjects type="digitalObject" separator="br"
                    terminator=", " introkey="" escapeXml="true">
      <digitalobject derivid="DocPortal_derivate_00000001"
             derivlabel="Dataobject from DocPortal_derivate_00000001"
             derivmain="EMX-Anleitung.pdf" size="176426"
             lastModified="17.08.2006 13:58:10" contentType="pdf"
             md5="46a83a324f760445f49a97d27c943521" pos="first" />
    </digitalobjects>
  </metaname>
  <metaname type="space" />
 ...
</all-metavalues>
```

Das erzeugte JDOM Objekt wird über die Standard JSTL Tags aus den Java Core und XML Librarys in die gewünschte Ausgabe transformiert.

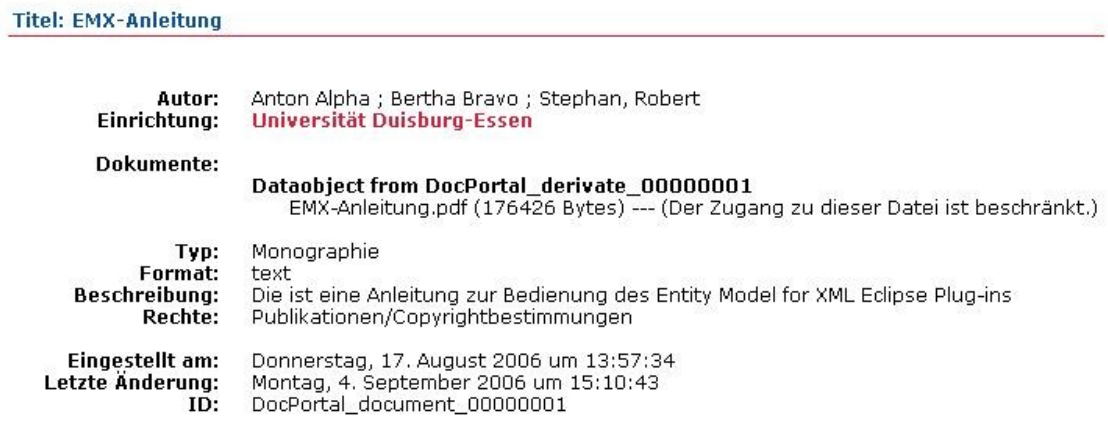


Abbildung 5.2: Detailanzeige eines MyCoRe Objekts

5.5 Trefferlisten

Analog zur Darstellung der Einzeltrefferanzeige existieren auch für die Trefferlisten, für jeden Datentyp ein Konfigurationsfile, das im Verzeichnis *jspdocportal/webpages/content/results-config* liegt.

Die Namen für die Konfigurationsdatei lauten: *resultlist-<datentyp>.xml.*

Die Verarbeitung der Metadatenelemente und deren Darstellung erfolgt nach den gleichen Prinzipien, wie in der Einzeltrefferanzeige.

Hinweise für den Entwickler:

Die Konfiguration wird auf das auszugebende MyCoReObjekt angewandt und erzeugt ein relativ einfach strukturiertes JDOM Objekt. Dies geschieht über den Aufruf des JSTL Tags aus der mycore-Tag Library in der Datei *searchresult.jsp* im Verzeichnis *jspdocportal/webpages/content*.

```
<mcr:setResultList var="resultList"  results="${mcrresult}" from="0"
      until="${numPerPage}" lang="${lang}" />
```

Mit dem Tag werden die Treffer von 0 bis zur Anzahl der darzustellenden Treffer für die aktuelle Seite in das JDOM-Objekt *$resultlist* geschrieben.
Im Anschluss daran kann der Inhalt dieses Objektes dann in der JSP-Datei mit den JSTL Standard Tags verarbeitet werden.

Auch hier können Sie sich durch Hinzufügen eines debug-Parameters in der URL (*&debug=true*), den XML-Output zusätzlich zu den formatierten Inhalten anzeigen lassen.

5.6 Suchmasken

Suchmasken werden mit der gleichen XML-Syntax definiert, die auch für die Erstellung von Editoren verwendet wird. Für Details sei auf die Entsprechende MyCoRe-Dokumentation sowie die vorhandenen Beispiele verwiesen.

Hinweise für den Entwickler:

In den Aufruf einer Suchmaske sind verschiedene Servlets und jsp-Dateien involviert. Deshalb soll er im folgenden kurz beschrieben werden.

Header einer Searchmask_*.xml

In den Header werden die folgenden Elemente aufgenommen

```
<editor>
   <source uri="request:servlets/MCRJSPSearchServlet?mode=load
             &amp;id={XSL.editor.source.id}&amp;MCRSessionID={MCRSessionID}" />
   <target type="servlet" url="/servlets/MCRJSPSearchServlet"
           method="post" format="xml" />
...
```

Im source-Element wird angegeben, wie bei einem erneuten Aufruf der Suchmaske, die Suchmaske mit den zuvor definierten Felder „vorausgefüllt" werden kann.
Die in geschw. Klammern ({}) gesetzten Parameter werden zur Laufzeit mit aktuellen Werten der entsprechenden Requestparametern ersetzt.

Integration in eine Webseite:

```
<c:import url="${applicationScope.WebApplicationBaseURL}editor
                     /searchmasks/SearchMask_AllMetadataFields.xml">
    <c:param name="XSL.editor.source.new" value="true" />
    <c:param name="lang" value="${requestScope.lang}" />
    <c:param name="XSL.editor.cancel.url"
             value="${applicationScope.WebApplicationBaseURL}" />
</c:import>
```

„Suche verfeinern"

- *searchresult.jsp:*

```
<a href="${WebApplicationBaseURL}servlets/MCRJSPSearchServlet
  ?mode=refine &mask=${mask}&id=${resultid}">Suche verfeinern</a>
```

- *MCRJSPSearchServlet*

ruft (via **mask**) einen Eintrag aus **navigation.xml** auf (z.B. `~searchstart-extended`) dieser verweist dann auf searchmask-editor.jsp

- *searchmask-editor.jsp*

als Parameter wird hier die XML-Beschreibung des Editors übergeben (z.B. `?editor=editor/searchmasks/SearchMask_ExpertDocument.xml`)

in dieser Datei wir der Editor dann wie oben beschrieben eingefügt.

5.7 Editieren von Webseiten

Es ist möglich, den Inhalt einzelner Webseiten zu editieren. Das gilt unter anderem für die Startseite und die Hinweisseiten zu den einzelnen Workflows.

Um Webseiten editieren zu können, muss der Nutzer das Recht „administrate-webcontent" besitzen (z. B. dadurch, dass er der Gruppe „adminwebcontent" angehört). Besitzt er das Recht, wird ihm an den editierbaren Stellen ein Icon angezeigt, welches einen HTML-Editor öffnet:

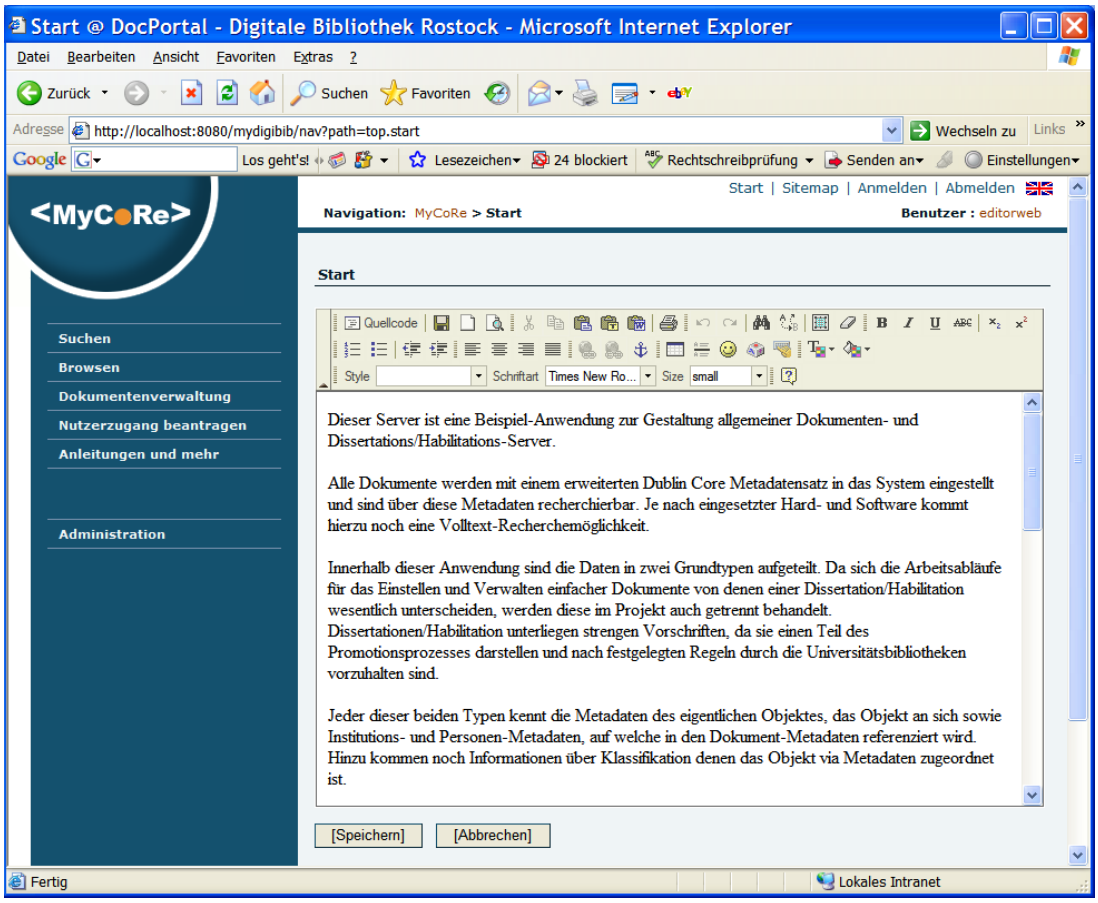
Abbildung 5.3: Edit-Button für Webseiten

5.7.1 FCK-Editor

Zum Editieren wird der FCK-Editor (http://www.fckeditor.net/) verwendet. Das ist ein OpenSource HTML Text Editor, der sich einfach in Webseiten integrieren lässt. Er ist gut konfigurierbar. So lässt sich beispielsweise das Aussehen der Symbolleisten an die eigenen Bedürfnisse anpassen.

Nachdem Sie den Button anklicken, wird der Editor geöffnet:

Abbildung 5.4: Edit-Button für Webseiten



Wie in der Abbildung zu sehen ist, stehen Ihnen, die aus Textprogrammen bekannten Funktionen zur Verfügung. Unter „Style" können Sie den einzelnen Elementen der Website CSS-Stile zuweisen. So werden beispielsweise Überschriften konform zu ihrer Anwendung dargestellt, da auf Ihre CSS-Definition zurückgegriffen wird.

5.7.2 Ablage der Dateien

Die Dateien werden auf dem Webserver in das durch das Property *MCR.WebContent.Folder* festgelegte, Verzeichnis gespeichert. An den Dateinamen wird als Suffix die aktuelle Sprache angehängt (z.B. *introtext_en.html*). Dadurch können mehrsprachige Anwendungen betreut werden.

5.7.3 Sicherung der Dateien

Insbesondere vor einer Neuinstallation der Webapplikation müssen die geänderten Webseiten gesichert werden, da sie sonst mit den vordefinierten Standardtexten aus dem Verzeichnis *myapplication/languages/webcontent* überschrieben werden.

Eine einfache Sicherungsmöglichkeit wurde im Administrationsmenü unter dem Punkt „Webseiten bearbeiten" geschaffen. Dort haben Sie die Möglichkeit alle anpassbaren Webseiten als Zip-Datei herunterzuladen und dann lokal zu sichern.

5.8 ImageViewer

Zum Betrachten von Bilddateien innerhalb eines Derivates wurde der ImageViewer aus dem DocPortal integriert. Es wird empfohlen, zunächst das entsprechende Kapitel im Programmer-Guide des Docportals zu lesen.

5.8.1 Vorausgegangene Integrationsschritte

Die folgenden Schritte sind bereits vom Entwickler durchgeführt worden. Sie sind werden hier dokumentiert, um neuere Versionen des ImageViewers in das JSPDocPortal zu integrieren.

- Kopieren des Verzeichnisses *\modules\UNINSTALLED_module-iview* vom DocPortal in das JSPDocPortal als *\modules\module-iview*.
- Erstellen eines Ant-Scripts, welches das Modul in ein neues Jar-File compiliert, diverse Dateien an die entsprechenden Stellen im JSPDocPortal kopiert, *mycore.properties* erweitert, neue Servlets in der *web.xml* registriert, ...
- Ergänzen der JAI*.jar's in den classpath der *build.xml* des JSPDocPortals
- Erweiterung der Tag-Library um neue Tags für den Aufruf des Imageviewers.

5.8.2 Installation

- Für den ImageViewer wird eine zusätzliche Java Bibliothek – die Java Advanced Images API (JAI) benötigt. Diese muss zunächst installiert werden. Dazu kann die API (aktuell in der Version 1.1.3) für das entsprechende Betriebssystem und Java-Version von https://jai.dev.java.net/binary-builds.html heruntergeladen und anschließend installiert werden (Installationsanleitung auch auf dieser Seite).
- Im MyCore-Kern muss das Modul *imaging* installiert werden.
- Die Integration des Imageviewers erfolgt automatisch durch den Build-Prozess.

5.8.3 Anwendung (Tags)

Die Integration des ImageViewers auf eine Website erfolgt durch Tags. Die folgenden Tags stehen dazu zur Verfügung. Ihre Beschreibung entnehme man den Kapitel „Tag Library"

- **imageViewer**
  Integration des ImageViewers in die Webseite
- **imageViewerGetSupport**
  prüft ob ein Derivate Daten enthält, die der ImageViewer anzeigen kann.

6 Integrierte Workflow-Engine zur interaktiven Autorenarbeit

Für eine einfache Definition der Workflows wurde die externe Workflow-Engine **jbpm** in JSPDocPortal integriert. Jbpm ist OpenSource, wird von Jboss gewartet und ist im Internet sehr gut dokumentiert. (http://www.jboss.com/products/jbpm) Es existiert auch ein Eclipse-Plugin zum graphischen Erstellen und Anzeigen der Prozessdefinitionen.

Workflow-Prozesse werden in xml-Konfigurationsdateien definiert, die dem jbpm-Standard **jpdl** entsprechen müssen. Diese Konfigurationsdateien liegen unter *config/workflow/<workflowprocesstyp>.par* und heißen jeweils *processdefinition.xml*.
(z.B. für den Workflow-Typ „xmetadiss": *config/workflow/xmetadiss.par/processdefinition.xml*)
In der Prozessdefinitionsdatei werden alle im Publikationsprozess anfallenden Arbeitsabläufe definiert. Außerdem ist ein Konzept für Nutzerrollen enthalten.

In der JSPDocPortal-Beispielanwendung sind Workflows für das Erstellen von Dissertationen, Publikationen, Autoren, Institutionen und für die Behandlung der Nutzerneuanmeldungen (Registration) definiert und implementiert.

Da für die Workflowprozesse in JSPDocPortal nicht alle in jbpm möglichen Features benötigt werden, wurde (bisher) nur eine Teilmenge von jpdl implementiert. Die genutzten Elemente werden im folgenden am Beispiel des xmteadiss Workflows kurz vorgestellt.

6.1 Modellierung eines Workflowprozesses

6.1.1 Prozess-Rollen

In jedem Workflow werden in der Regel menschliche Akteure Aufgaben zu erfüllen haben. Je nach Aufgabe kann ein Akteur auch in verschiedenen Rollen agieren. Die Rollen werden über das Element <swimlane> definiert.

```
<swimlane name="initiator" />
```

Mit „initiator" bekommt der Autor/Initiator eines Workflows seine eigene Rolle zugewiesen. Dies ist in jeder JSPDocPortal-Prozeßdefinition erforderlich.

```
<swimlane name="disseditor">
 <assignment class=
   "org.mycore.frontend.workflowengine.jbpm.MCRPooledActorAssignmentHandler">
  <groupName>admindisshab</groupName>
 </assignment>
</swimlane>
```

Hier wird eine Rolle „disseditor" definiert. Durch den angegebenen AssignmentHandler werden nur Mitglieder der MyCoRe-Benutzergruppe „admindisshab" berechtigt, Aufgaben für die Rolle „disseditor" im weiteren Workflow zu übernehmen.

6.1.2 Start- und Endzustand

In jedem Workflow ist ein Anfangs- und ein Endzustand zu definieren. (start-state, end-state)

```
<start-state name="start">
  <task name="initialization" swimlane="initiator" />
  <transition name="go2processInitialized"
              to="processInitialized" />
  <transition name="go2processEditInitialized"
              to="processEditInitialized" />
</start-state>
```

Dieser Startknoten beinhaltet die Task zum Initialisieren eines Workflows mit 2 möglichen Zustandsübergängen – die Initialisierung des Workflow für eine neues Dokument oder die Initialisierung eines Workflows zur Bearbeitung eines bereits auf dem Server vorhandenem Dokument.

```
<end-state name="cleanUpWorkingDirectory">
  <event type='node-enter'>
   <action class=
   "org.mycore.frontend.workflowengine.jbpm.MCRCleanUpWorkflowAction" >
     <varnameOBJID>createdDocID</varnameOBJID>
     <varnameERROR>CLEANUPERROR</varnameERROR>
   </action>
  </event>
</end-state>
```

Mit Erreichen des Endzustandes ist hier noch die Aktion des 'Aufräumens' verbunden, die in der Klasse genau definiert ist.

6.1.3 Aufgabenknoten

Aufgabenknoten sind Zwischenzustände in der Prozessdefinition.

Ein Task-Knoten enthält immer einen Task, der eine bestimme Aufgabe beschreibt und sie einer Nutzerrolle zuordnet. Der Nutzer muss zum Beispiel notwendige Informationen durch Ausfüllen eines Formulars oder Anklicken eines Knopfes / Links bereitstellen.

Der Übergang von einem Zustand in einen nächsten Zustand wird durch Transitionen definiert.

```
<task-node name="disshabCreated">
 <task name="taskCompleteDisshabAndSendToLibrary"
       swimlane="initiator" />
     <transition name="go2canDisshabBeSubmitted"
             to="canDisshabBeSubmitted" />
</task-node>
```

Im Beispiel haben wir zunächst den Aufgabenknoten „disshabCreated. In diesem ist eine Aufgabe für die Rolle „initiator" definiert. Die Aufgabe hat den Namen „taskCompleteDisshabAndSendToLibrary". Beim Ausführen der Aufgabe muss der Autor die Metadaten vervollständigen und anschließend auf „Absenden" drücken. Dann wechselt der Zustand über die Transition „go2canDisshabBeSubmitted" zum Entscheidungsknoten „canDisshabBeSubmitted".

6.1.4 Entscheidungsknoten

Wenn der Workflow ohne Nutzereingabe automatisch entscheiden kann, ob und mit welcher Transition der Workflow seinen Zustand verändert, wählen wir einen Entscheidungsknoten (decision-node).

```
<decision name="canDisshabBeSubmitted">
   <handler class=
"org.mycore.frontend.workflowengine.jbpm.xmetadiss.MCRDecisionHandlerXmetadiss"
      />
   <transition name="disshabCantBeSubmitted" to="disshabCreated" />
   <transition name="disshabCanBeSubmitted" to="disshabSubmitted" />
</decision>
```

Im Entscheidungsknoten muss nun automatisch ermittelt werden können, mit welcher Transition im Workflow weitergemacht wird. Um die Bedingungen zu überprüfen kann man das Interface org.jbpm.graph.node.DecisionHandler implementieren oder wie im Beispiel oder auf die jpdl Expression Language zurückgreifen, die syntaktisch auf JSP EL basiert (siehe unten). In der Regel sollte das Verhalten der vordefinierten „decision-nodes" nur von Java-Programmierern angepasst werden.

```
<decision name="wasCommitmentSuccessful">
  <transition name="go2adminCheck" to="adminCheck">
   <condition expression=
            "#{!empty(contextInstance.variables['COMMITERROR'])}" />
  </transition>
  <transition name="go2disshabCommitted" to="disshabCommitted">
    <condition expression=
            "#{empty(contextInstance.variables['COMMITERROR'])}" />
  </transition>
</decision>
```

6.1.5 Actions

Innerhalb eines Workflow-Prozesses werden auch Dinge erledigt, die für den Geschäftsablauf nicht relevant sind. Hierfür steht das Konzept der Actions zur Verfügung. Actions sind Klassen, die das Interface org.jbpm.graph.def.ActionHandler implementieren und bieten sich zum Beispiel für das Versenden von Benachrichtigungen über den aktuellen Workflow-Zustand oder für das Aufrufen reiner MyCoRe-Funktionalitäten an (siehe die beiden Beispiele unten).

Actions lassen sich auch mit Ereignissen für Zustandsknoten wie node-enter und node-leave verknüpfen. Sie können auch für Transitionen definiert werden.

```
<task-node name="disshabCommitted">
 <event type='node-enter'>
  <action class=
"org.mycore.frontend.workflowengine.jbpm.xmetadiss.MCRSendmailActionXmetadiss"
>
    <from>MCR.WorkflowEngine.xmetadiss.replyTo</from>
    <to>initiator</to>
    <subject>Ihre elektronische Dissertation wurde erfolgreich
      veröffentlicht
```

```
    </subject>
    <!-- <body>dynamic</body> -->
    <mode>success</mode>
  </action>
 </event>
 <transition name="go2cleanupWorkflow" to="cleanupWorkflow" />
</task-node>
```

Dem Initiator wird eine Erfolgsbestätigungsmail zugesendet, die Adressangaben können über ein Property in *mymycore.properties* applikationsspezifisch gesetzt werden.

Wenn der Entscheidungsknoten als nächste Transition „`go2cleanupWorkflow`" auswählt, wird der „end-state" (wie schon oben dargestellt) erreicht und der Prozess ist beendet.

6.1.6 Beispieldefinitionsdatei für Publicationen

File: *config/workflow/publication.par/processdefinition.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition  xmlns="urn:jbpm.org:jpdl-3.1"  name="publication">

 <!-- SWIMLANES (= process roles) -->
 <swimlane name="initiator" />

 <swimlane name="publicationeditor">
       <assignment class=
       "org.mycore.frontend.workflowengine.jbpm.MCRPooledActorAssignmentHandler>
        <groupName>adminpublication</groupName>
       </assignment>
  </swimlane>

  <swimlane name="technicalAdministration">
    <assignment class=
       "org.mycore.frontend.workflowengine.jbpm.MCRPooledActorAssignmentHandler>
       <groupName>admingroup</groupName>
     </assignment>
  </swimlane>

  <start-state name="start">
     <task name="initialization" swimlane="initiator" />
     <transition name="go2getPublicationType" to="getPublicationType" />
     <transition name="go2processEditInitialized"
              to="processEditInitialized" />
  </start-state>

  <task-node name="getPublicationType">
     <task name="taskGetPublicationType" swimlane="initiator" />
     <transition name="go2processInitialized" to="processInitialized" />
  </task-node>

  <task-node name="processInitialized">
     <task name="taskprocessInitialized" swimlane="initiator" />
     <event type='node-enter'>
        <action class=
     "org.mycore.frontend.workflowengine.jbpm.publication.MCRCreateURNAction"/>
        <action class=
"org.mycore.frontend.workflowengine.jbpm.publication.MCRCreateDocumentAction"/>
       <action class=
     "org.mycore.frontend.workflowengine.jbpm.publication.MCRAssignURNAction"/>
     </event>
     <transition name="go2isInitiatorsEmailAddressAvailable"
                  to="isInitiatorsEmailAddressAvailable" />
  </task-node>

  <decision name="isInitiatorsEmailAddressAvailable">
     <transition name="no" to="getInitiatorsEmailAddress">
        <condition expression=
              "#{empty(contextInstance.variables['initiatorEmail'])}"/>
     </transition>
     <transition name="yes" to="documentCreated">
        <condition expression=
              "#{!empty(contextInstance.variables['initiatorEmail'])}"/>
     </transition>
  </decision>

  <task-node name="getInitiatorsEmailAddress">
     <task name="taskGetInitiatorsEmailAddress" swimlane="initiator" />
     <transition name="go2IsInitiatorsEmailAddressAvailable"
```

```
                    to="isInitiatorsEmailAddressAvailable" />
    </task-node>

    <task-node name="documentCreated">
       <task name="taskCompleteDocumentAndSendToLibrary" swimlane="initiator" />
       <transition name="go2canDocumentBeSubmitted" to="canDocumentBeSubmitted"/>
    </task-node>

    <task-node name="processEditInitialized">
      <task name="taskprocessEditInitialized" swimlane="initiator" />
         <transition name="go2canDocumentBeSubmitted" to="canDocumentBeSubmitted" />
    </task-node>

    <decision name="canDocumentBeSubmitted">
       <handler class=
"org.mycore.frontend.workflowengine.jbpm.publication.MCRDecisionHandlerPublication"/>
       <transition name="documentCantBeSubmitted" to="documentCreated" />
       <transition name="documentCanBeSubmitted" to="documentSubmitted"/>
    </decision>

    <task-node name="documentSubmitted">
       <event type='node-enter'>
          <action class=
"org.mycore.frontend.workflowengine.jbpm.publication.MCRDocumentSubmittedAction">
           <lockedVariables>initiator,createdDocID,reservatedURN
            </lockedVariables>
          </action>
       </event>
       <task name="taskCheckCompleteness" swimlane="publicationeditor">
       </task>
       <transition name="go2sendBackToDocumentCreated"
                    to="sendBackToDocumentCreated" >
          <action class=
       "org.mycore.frontend.workflowengine.jbpm.MCRSetDefaultAclsAction">
             <varmcrid>createdDocID</varmcrid>
             <varuserid>initiator</varuserid>
          </action>
       </transition>
       <transition name="go2canDocumentBeCommitted" to="canDocumentBeCommitted"/>
    </task-node>

    <task-node name="sendBackToDocumentCreated">
       <task name="taskEnterMessageData" swimlane="publicationeditor"/>
       <transition name="go2documentCreated2" to="documentCreated">
        <action class="org.mycore.frontend.workflowengine.jbpm.MCRSendmailAction">
            <from>MCR.WorkflowEngine.publication.from</from>
            <to>initiator</to>
            <replyTo>MCR.WorkflowEngine.publication.replyto</replyTo>
            <subject>Sie müssen bei Ihrer Publikation Daten ergänzen</subject>
            <!-- <body>Mustertext</body> -->
            <jbpmVariableName>tmpTaskMessage</jbpmVariableName>
          </action>
       </transition>
    </task-node>


    <decision name="canDocumentBeCommitted">
       <handler class=
"org.mycore.frontend.workflowengine.jbpm.publication.MCRDecisionHandlerPublication"/>
       <transition name="go2sendBackToDocumentCreated"
                    to="sendBackToDocumentCreated" />
      <transition name="go2wasCommitmentSuccessfull"
                    to="wasCommitmentSuccessful">
       <action class="org.mycore.frontend.workflowengine.jbpm.MCRCommitObjectAction">
             <varnameOBJID>createdDocID</varnameOBJID>
```

```
            <varnameERROR>COMMITERROR</varnameERROR>
        </action>
      </transition>
    </decision>

    <decision name="wasCommitmentSuccessful">
       <transition name="go2adminCheck" to="adminCheck">
        <condition expression="#{!empty(contextInstance.variables['COMMITERROR'])}"/>
       </transition>
       <transition name="go2documentCommitted" to="documentCommitted">
        <condition expression="#{empty(contextInstance.variables['COMMITERROR'])}"/>
       </transition>
    </decision>

    <task-node name="adminCheck">
       <event type='node-enter'>
         <action class="org.mycore.frontend.workflowengine.jbpm.MCRSendmailAction">
            <from>MCR.WorkflowEngine.publication.from</from>
            <to>MCR.WorkflowEngine.publication.admin</to>
            <replyTo>MCR.WorkflowEngine.publication.from</replyTo>
            <subject>Probleme beim Veröffentlichen einer Publikation</subject>
            <body>Die Publikation konnte nicht veröffentlicht werden.</body>
         </action>
       </event>
       <task name="taskAdminCheckCommitmentNotSuccessFul"
               swimlane="technicalAdministration" />
       <transition name="go2documentCommitted" to="documentCommitted" />
    </task-node>

    <task-node name="documentCommitted">
       <event type='node-enter'>
          <action class=
"org.mycore.frontend.workflowengine.jbpm.publication.MCRSendmailActionPublication">
            <from>MCR.WorkflowEngine.publication.from</from>
            <to>initiator</to>
            <replyTo>MCR.WorkflowEngine.publication.from</replyTo>
            <subject>Ihre Publikation wurde erfolgreich veröffentlicht</subject>
            <!-- <body>dynamic</body> -->
            <mode>success</mode>
          </action>
       </event>
       <transition name="go2cleanUpWorkingDirectory" to="cleanUpWorkingDirectory"/>
    </task-node>

    <end-state name="cleanUpWorkingDirectory">
       <event type='node-enter'>
          <action class=
                "org.mycore.frontend.workflowengine.jbpm.MCRCleanUpWorkflowAction" >
             <varnameOBJID>createdDocID</varnameOBJID>
             <varnameERROR>CLEANUPERROR</varnameERROR>
          </action>
       </event>
    </end-state>
  </process-definition>
```

6.1.7 Visualisierung einesWorkflow-Prozesses

Für die Visualisierung eines Workflowprozesses steht das Eclipse-Plugin

„jbpm-Graphic-Designer" zur Verfügung, welches die Prozessdefinition als Graphen darstellt und die Möglichkeit der grafischen Bearbeitung bietet.

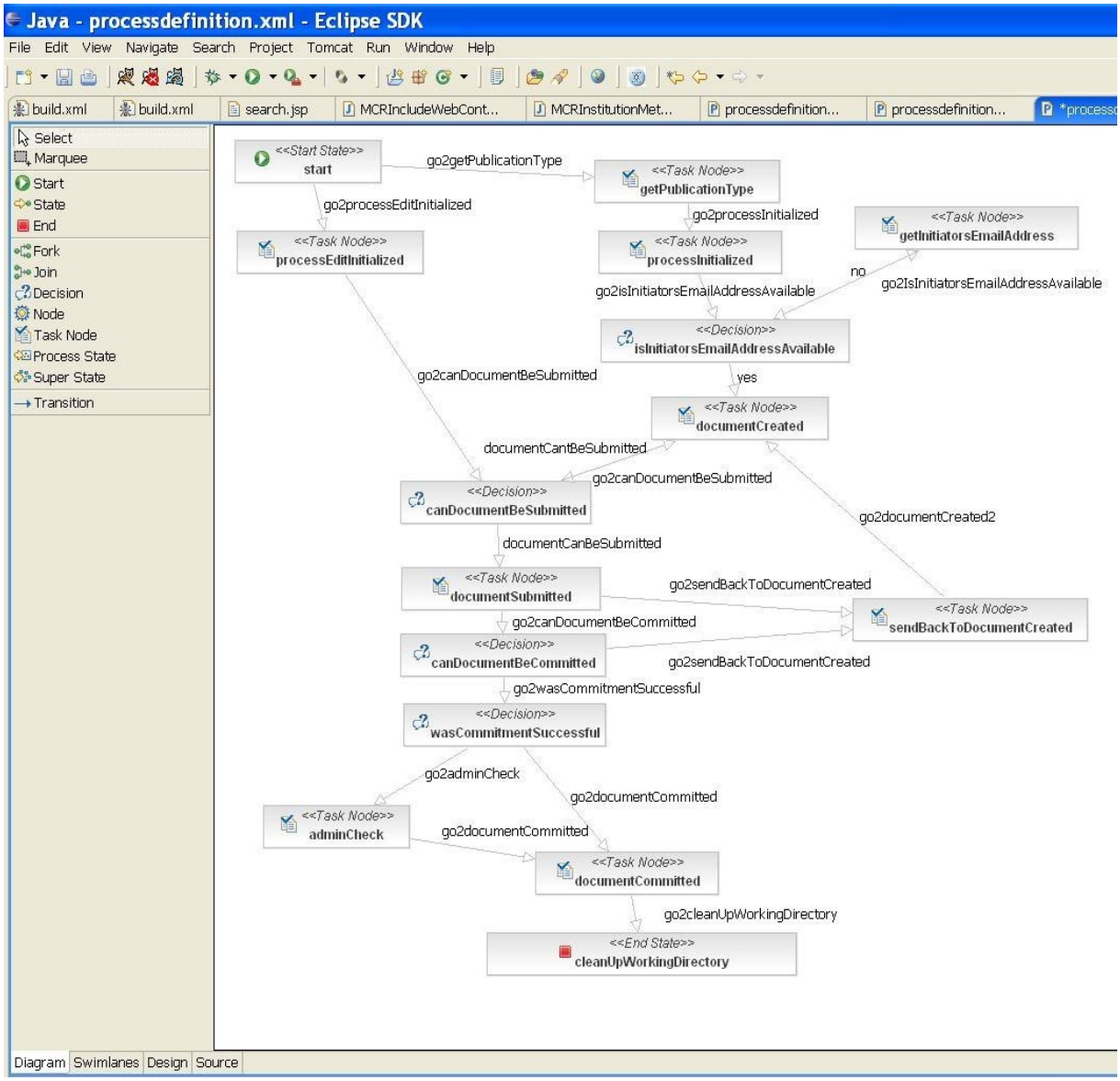


Abbildung 6.1: jbpm-Graphic-Designer

6.2 Installation der Workflowengine

Um die aktuellen Zustände zu speichern (Persistenz) werden diese über Hibernate in eine SQL-Datenbank geschrieben. Die Verbindungsparameter werden in *config/workflow/jbpm_hibernate.cfg.xml* konfiguriert.

Der Name der Datenbank wird über das folgende Property festgelegt:

```
MCR.persistence_workflow_sql_database_name=mycore-workflow
```

Initialisiert wird die Datenbank durch das Ausführen eines Ant-Scripts (aus build.xml) im Ordner *myapplication*:

```
ant create.workflowengine.database
```

Zum Einspielen der Definitionen der Workflow-Prozesse ist auszuführen:

```
ant deploy.workflow.processdefinitions
```

Damit werden alle Prozess-Definitionen in die Datenbank übertragen und aktiviert.

Man kann eine Prozess-Definition auch einzeln installieren,

```
z.B. ant deploy.processdefinition.xmetadiss
```

Dies muss bei jeder Änderung der Prozessdefinition ausgeführt werden!!

6.3 Die zu einem Workflow gehörenden Java-Klassen und JSP-Dateien

Zu jedem Workflow gehören eine Reihe von JSP-Dateien, die im Verzeichnis webpages/content/workflow/<workflowtype> abgelegt werden.

Das Kernstück ist die Datei *workflow.jsp.* Über diese Datei werden alle aktuellen Prozesse eines Workflow aufgerufen. Die einzelnen Aufgaben werden in der Datei *getTask.jsp* definiert und die für die jeweilige Aktion benötigten UI-Elemene erzeugt.

Der Start eines neuen Workflowprozesses wird durch die JSP-Datei *begin.jsp* initiiert. Je nach Rolle des Anwenders (Editor/Autor/Administrator) wird eine Information zur Vorgehensweise dargestellt. Die Informationstexte sind als externe WebContent-Dateien eingebunden und können entsprechend angepasst werden. (Siehe Kapitel „Editieren von Webseiten" und Kapitel „MyCoRe Tag Library" JSP-TAG includeWebContent)

Alle zum Workflow gehörenden Java Klassen liegen im Verzeichnis *source/org/mycore/frontend/workflowengine/**. Dabei gibt es die Ordner *jbpm* und *strategies*. Klassen, die alle Workflowtypen benötigen, liegen im Basis-Verzeichnis *jbpm*. Strategien, die alle Workflowtypen benutzen liegen im *strategies* Verzeichnis.

Für jeden Workflowtypen gibt es unterhalb des Ordners *jbpm* ein entsprechendes Verzeichnis, in dem sich alle Klassen befinden, die für diesen Workflowtypen angepasst wurden.

6.4 Einen eigenen Workflow hinzufügen

Wenn sie die Anforderung haben, einen eigenen Workflow in ihre Anwendung zu integrieren, sollten Sie sich zunächst genau die Arbeitsweise der vorhandenen Workflows anschauen und sich einen der Ihren Anforderungen am nächsten kommenden als Vorlage nehmen. Legen Sie analog zu den anderen Prozessdefinitionen Ihre Prozessdefinition in ein Verzeichnis, das sie sinnvollerweise

mit dem Namen des Workflowtyps bezeichnen. Dieses Verzeichnis sollte sich in *myapplication/config/workflow* befinden.

TODO!!

7 Die MyCoRe Tag Library

7.1 Allgemeines

Die MyCoRe-Tag-Library ist eine Sammlung von MyCoRe-Methoden, die bei Erzeugen der JSP-Seiten hilfreich sind.

Die Tag-Libraray wird wie eine Standard JSTL im Kopf der JSP-Seite eingebunden.

```
<%@ taglib uri="/WEB-INF/lib/mycore-taglibs.jar" prefix="mcr"%>
```

Die Meta-Beschreibungsdatei *mycore-tablib.tld*, zur Definition der MyCoRe-eigenen Markup-Elemente (Tags) befindet sich im Konfigurationsverzeichnis.

Über die TLD-Datei werden die eigene Elemente, samt fakultativer Attribute, XML-konform definiert und mit der entsprechenden serverseitig ausgeführten Klassenbibliotheken assoziiert. Damit ist eine konsequente Trennung von Code und Darstellungslogik (gekapselt durch die Taglibs) realisiert.

Die Ausführung der Tags ist immer von den Privilegien des aktuellen Nutzers abhängig. Hat dieser nicht die entsprechenden Rechte, erfolgt eine dementsprechende Fehlermeldung.

Im Folgenden sollen die wichtigsten Tags und ihre Verwendung erklärt werden.

7.2 Such- und Treffer spezifische Tags

Tagname		***verwendet z. B. in searchresult.jsp***
setResultList		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRSetResultListTag		
Parameter:		
var	mandatory	PageContext-Variable für das Ausgabeelement (DOM-Objekt)
results	mandatory	übergibt das Suchergebnis aus der vorangegangenen Suche (JDOM Objekt)
from	mandatory	position ab dem Treffer ausgegeben werden
until	mandatory	position bis zu der Treffer ausgegeben werden
lang	mandatory	Sprache der Ausgabeelemente
Beispielaufruf: `<mcr:setResultList var="resultList" results="${mcrresult}" from="0" until="${numPerPage}" lang="${lang}" />`		

Beschreibung: Das in der Session zur Verfügung stehende mcrresult wird an das Tag übergeben. Es werden die Treffer von 0 bis 10 für die XML-Basierte Ausgabe vorformatiert und in einem DOM-Objekt zurückgeben, sind nur weniger Treffer vorhanden, werden entsprechend weniger zurückgegeben. Die Beschreibung der Ausgabeelemente ist in Kap. 5 (Hinweise für den Entwickler) dargestellt.

Tagname		***verwendet z. B. in docdetails.jsp***
receiveMcrObjAsJdom		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRReceiveMcrObjAsJdomTag		
Parameter:		
varDom	optional	setzen der PageContext-Variable für das Ausgabeelement (DOM-Objekt)
var	optional	oder setzen der PageContext-Variable für das Ausgabeelement (JDOM Objekt)
mcrid	mandatory	MCRID des auszugebenen Objekts
fromWForDB	optional	das MyCoRe-Objekt kann wahlweise aus dem Workflow oder aus der Datenbank geholt werden. {**db**|workflow}

Beispielaufruf:
```
<mcr:receiveMcrObjAsJdom var="mycoreobject" mcrid="${mcrid}" fromWForDB="${from}" />
```

Beschreibung: Das über die MCRID adressierte MyCoRe-Objekt wird, wenn nicht anders angegeben, aus der Datenbank geholt, für die XML-basierte Ausgabe vorformatiert und in einem JDOM- oder DOM-Objekt zurückgeben. Die Parameter *var* oder *varDom* sollten alternativ gesetzt sein. Die Beschreibung der Ausgabeelemente ist in Kap. 5 (Hinweise für den Entwickler) dargestellt.

Tagname		***verwendet z. B. in docdetails.jsp***
simpleXpath		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRSimpleXpathTag		
Parameter:		
jdom	mandatory	das MyCoRe-Objekt (JDOM-Objekt)
xpath	mandatory	der Pfadausdruck des auszugebenen Elements

Beispielaufruf:
```
<mcr:simpleXpath jdom="${mycoreobject}" xpath="/mycoreobject/metadata/titles/title[@xml:lang='${requestScope.lang}']" />
```

Beschreibung: Das z. B. vorher durch das TAG receiveMcrObjAsJdom geholte MyCoRe-Objekt wird als JDOM-Objekt übergeben und es erfolgt die Ausgabe des in xpath referenzierten Wertes – soweit dieser vorhanden ist.

Tagname	***verwendet z. B. in docdetails.jsp***	
docDetails		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRDocDetailsTag		
Parameter:		
var	mandatory	setzen der PageContext-Variable für das Ausgabeelement (DOM-Objekt)
style	mandatory	Datentyp des MyCoRe-Objekts {user|disshab|document|author...}
mcrObj	mandatory	das MyCoRe-Objekt (JDOM-Objekt)
lang	mandatory	Sprache der Ausgabeelemente

Beispielaufruf:
```
<mcr:docDetails mcrObj="${mycoreobject}" var="docDetails"
lang="${requestScope.lang}" style="${style}" />
```

Beschreibung: Das z. B. vorher durch das TAG receiveMcrObjAsJdom geholte MyCoRe-Objekt wird als JDOM-Objekt übergeben. Aus dem Objekt wird wird ein XML-Baum erzeugt, der je nach Style mit dem entsprechenden Konfigurationsfile aus dem Vereichnis *webpages/content/results-config* aufgebaut wird.

Das erzeugte DOM Objekt wird in die Ausgabevariable *var* geschrieben und kann per Standard JSTL weiterverarbeitet werden. Die Beschreibung der Ausgabeelemente ist in Kap. 5 (Hinweise für den Entwickler) dargestellt.

Tagname	***verwendet z. B. in docdetails.jsp***	
checkAccess		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRCheckAccessTag		
Parameter:		
var	mandatory	setzen der PageContext-Variable für das Ausgabeelement (String Objekt)
permission	mandatory	zu prüfendes Recht
key	optional	die ID des zu prüfenden MyCoRe-Privilegs

Beispielaufruf:
```
<mcr:checkAccess var="modifyAllowed" permission="writedb"
key="${mcrid}" />
```

Beschreibung: Für das Objekt mit der ID *key* wird die angegebene Permission für

den aktuellen Nutzer überprüft uns das Ergebnis in *var* zurückgegeben (typische objektgebunden Privilegien sind z. B. read, writedb, comitdb, ...).

Für aktionsgebundene Zugriffsrechte wird keine Objekt-ID benötigt. Die Permission wird allgemein für den aktuellen Nutzer geprüft (z. B. Pool-Privilegien aus der Datei permissions.xml).

Tagname		***verwendet z. B. in docdetails.jsp***
session		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRSessionTag		
Parameter:		
var	mandatory	die PageContext-Variable, aus der ein Wert gelesen, bzw. in die geschrieben wird
method	mandatory	„set" - Setzen von Sessionvariablen „get" - Auslesen von Sessionvariablen
type	optional	Typ des Wertes der gelesen/geschrieben werden soll:„userID", „userName", „IP", „language", „ID"
key	optional	der Schlüssel zu einer Variable, die in der session-internen Hashtabelle abgelegt / gelesen werden kann

Beispielaufruf:
```
<mcr:session var="lang" method="get"
                          type="language"/>
```

Beschreibung: Mit diesem Tag lassen sich Parameter und Variablen der MCRSession lesen und schreiben.

7.3 Webseiten editieren

Um größere Textpassagen innerhalb einer JSP-Datei editierbar zu machen, wurde folgendes Tag eingeführt:

Tagname		***verwendet z. B. in documentmanagement.jsp***
includeWebContent		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRIncludeWebContentTag		
Parameter:		
file	mandatory	die Datei, deren Inhalt ein die JSP-Seite einzubetten ist.

Beispielaufruf:
```
<mcr:includeWebContent file="documentmanagement_introtext.jsp" />
```

Beschreibung: Besitzt der aktuelle Nutzer das Recht „administrate-webcontent" wird zusätzlich zu dem Inhalt der Datei ein Button angezeigt, der einen HTML Editor öffnet.

Sämtliche Dateien befinden sich im durch das Property *MCR.WebContent.Folder* definierten Verzeichnis.

An den Dateinamen wird zuvor ein Suffix für die aktuelle Sprache angehängt, dadurch wird die Datei eingebunden, die die aktuelle Sprache enthält. Existiert so eine Datei nicht, wird die Defaultdatei (ohne Suffix) angezeigt.

Beispiel: Standardmäßig ist Deutsch im System gesetzt, dann wird *documentmanagement_introtext.jsp* includiert, wird die Anzeige der Website auf Englisch umgeschaltet, wird entsprechend *documentmanagement_introtext_en.jsp* eingebunden.

(Ausführlichere Informationen entnehmen Sie dem Kapitel „Webseiten editieren".)

7.4 Editoren einbetten

Ein Editor für die Metadaten eines Objektes kann über den Import der Seite *editor-include.jsp* eingebettet werden. Editoren können Suchmasken oder Masken für MyCoRe-Objekte darstellen. Eine ausführliche Beschreibung für Editoren finden Sie im User- und Programmer Guide von **MyCoRe**, da diese in beiden Applikationen **DocPortal** und **JSPDocPortal** gleich verwendet werden.
An dieser Stelle wird lediglich der Aufruf und die Einbindung in JSPDocPortal beschrieben.

Tagname		***verwendet in editor-include.jsp***
includeEditor		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRIncludeEditorTag		
Parameter:		
editorSessionID	optional	falls vorhanden, wird die SessionID des Editors (siehe XSL.editor.session.id in der MyCoRe Programmers Guide zum Editor)
isNewEditorSource	mandatory	leeren Editor laden {true|false} (XSL.editor.source.new)
editorSource	optional	die Url des Editors (XSL.editor.source.url)
editorPath	optional	ev. abweichende Pfadangabe zum Editor ($BaseUrl/editor/workflow)
mcrid	mandatory	die ID des MyCoRe-Objekts das in den Editor geladen werden soll.
nextPath	mandatory	Pfad, der nach dem Editiervorgang aufgerufen werden soll

cancelurl	optional	extra Abbruch Pfad, Defaultmäßig wird wieder der aktuelle Workflow angezeigt
target	mandatory	Servlet, das die Eingabe verarbeitet
processid	mandatory	ProzessID des aktuellen Workflows
workflowType	mandatory	Workflowtyp {xmetadiss|publication|... }
step	mandatory	Editorstep {author|editor}
type	mandatory	Typ des Objekts {document|disshab|...}
publicationType	optional	Untertyp einer Publikation
mcrid2	optional	2. ID für die Angabe der Derivate ID
uploadID	optional	nur bei Laden des Upload Editors

Beispielaufruf:

```
<mcr:includeEditor
    editorSessionID="${editorSessionID}"          isNewEditorSource="$
    {isNewEditorSource}"
    mcrid="${mcrid}" type="${type}"
    processid="${processid}" workflowType="${workflowType}"
    publicationType="${publicationType}"  step="author"
    target="${target}" nextPath="${nextPath}"
    editorPath="${editorPath}"      editorSource="${editorSource}"
    mcrid2="${mcrid2}" uploadID="${uploadID}"
    />
```

Beschreibung: Die Parameter werden in unterschiedlicher Kombination verwendet. Im Allgemeinen wird ein leerer oder ein mit Daten aus einem MyCoRe-Objektes befüllter Editor innerhalb eines Workflows aufgerufen. Um den richtigen Editor zu laden, müssen die Variable *step, type* und eventuell *publicationtype* übergeben werden. Aus ihnen wird die URL des Ediors zusammengestellt.

Bsp.: *$BaseUrl/editor/workflow/editor_form_author_disshab.xml*

Da wir im Publikationenworkflow zwischen selbständigen und unselbständigen Publikation unterscheiden, muss hierfür auch der Publikationstyp mit übergeben werden. Der Publikationstyp für selbständige Publikationen ist TYPE0001, für unselbständigen Publikationen TYPE0002 (siehe Klassifikation der Dokumentypen DocPortal_class_00000005).

Für die Variable-Belegung *step=editor* sind derzeit keine Editoren vorhanden, da die Rollenkonzepte des Editierenden über den Workflow geregelt wird.

Es können natürlich nur Editoren aufgerufen werden (Zusammensetzen der URL), die auch vorhanden sind.

Für das wiederholte Laden von Suchmasken, die ebenfalls als Editorformulare abgelegt sind, wird z. B. die SessionID der Maske benötigt um die bereits eingegebenen Suchbegriffe, die in der Session gespeichert sind, erneut anzuzeigen.

Der Aufruf dieses Tags ist innerhalb der Workflowkomponenten in Servlets eingebettet, indem die JSP-Seite *editor-include.jsp* verwendet wird. Hier sind standardmäßig alle Parameter des Tags gesetzt, auch wenn sie nicht alle immer gleichzeitig benötigt werden.

7.5 An-/Abmelden

Tagname		***verwendet z. B. in login.jsp***
login		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRLoginTag		
Parameter:		
uid	mandatory	die Nutzer-ID
pwd	mandatory	das Passwort
var	mandatory	setzen der PageContext-Variable für das Ausgabeelement (DOM-Objekt)

Beispielaufruf:

```
<mcr:login uid="${param.uid}" pwd="${param.pwd}" var="loginresult" />
```

Beschreibung: Die im Formular eingegebenen Werte werden im Login-Tag ausgewertet. Die Ausgabe erfolgt über das DOM-Objekt. Der Erfolg der Anmeldung wird in einem Status-Element abgelegt. Ist die Anmeldung erfolgreich, werden der aktuelle Benutzername, der Name und seine Gruppenrechte dargestellt, damit der Nutzer weiß, welche Aktionen er im System vornehmen kann.

Tagname		***verwendet z. B. in login.jsp***
login_startlink		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRLogin_StartLinkTag		
Parameter:		
group_id	mandatory	die Gruppen-ID
group_description	mandatory	die Gruppen-Beschreibung

Beispielaufruf:

```
<mcr:login_startlink group_id="${gid}" group_description="${gdescr}" />
```

Beschreibung: Dieses Tag erzeugt einen Link ... mit einer Linkadresse, die anhand der group_id in den Properties als `MCR.Application.Login.StartLink.${group_id}` nachgeschlagen wird. Und der group_description als Linktext.

Dadurch kann der eingeloggte Nutzer für jedes seiner gültigen Rechte auf eine passende Startseite navigieren.

7.6 Workflowspezifische Tags

Einige Tags werden bei der Abarbeitung einer Workflowprozessdefinition benötigt.

Tagname	***verwendet z. B. in workflow/publication/begin.jsp***	
initWorkflowProcess		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRInitWorkflowProcessTag		
Parameter:		
status	mandatory	PageContext-Variable mit dem Ergebnis der Initialisierung
workflowType	mandatory	typ {xmetadiss|publication|author|...}
processidVar	mandatory	PageContext-Variable mit dem des ProzessID des erzeugten Workflowprozesses.
transition	mandatory	Angabe der Transition, mit der der aktuelle Zustand des Workflows nach Abarbeitung verlassen werden soll.
scope	mandatory	Definitionsraum

Beispielaufruf:

```
<mcr:initWorkflowProcess userid="${username}"
     status="status" workflowProcessType="${workflowType}"
     processidVar="pid" scope="request"
     transition="go2getPublicationType"
     />
```

Beschreibung: Es wird für den aktuellen Nutzer ein Workflow eines bestimmten Typen gestartet. Dabei wird nach dem Erzeugen des Workflows, je nach Prozessdefinition in die nächst mögliche Transition verzweigt.

Beispiel: Für den Publikationenworkflow sind innerhalb der Prozessdefinition für das Initialisieren 2 Transitionen möglich, je nachdem ob ein neue Publikation angelegt werden soll, oder eine vorhandene bearbeitet werden muss.

```
<start-state name="start">
  <task name="initialization" swimlane="initiator"> </task>
  <transition name="go2getPublicationType"  to="getPublicationType" />
  <transition name="go2processEditInitialized" to="processEditInitialized" />
  </start-state>
```

Tagname	***verwendet z. B. in workflow/publication/workflow.jsp***
listWorkflowProcess	
Klasse	
org.mycore.frontend.jsp.taglibs.MCRListWorkflowProcessTag	

Parameter:		
var	mandatory	PageContext-Variable in der das Ergebnis (DOM-Objekt) abgelegt wird.
mode	mandatory	Modus {activeTasks|initiatedProcesses}
workflowTypes	mandatory	Komma separierte Liste der zu testenden Workflowtypen {publication,xmetadiss,..}
varTotalSize	mandatory	Maximum an Einträgen

Beispielaufruf:

```
<mcr:getWorkflowTaskBeanList var="myTaskList" mode="activeTasks"
workflowTypes="publication" varTotalSize="total1" />
<mcr:getWorkflowTaskBeanList var="myProcessList" mode="initiatedProcesses"
workflowTypes="publication"    varTotalSize="total2" />
```

Beschreibung: Im Modus *activeTasks* werden für die angegebenen Workflowtypen die ausstehenden Aufgaben und die aktuellen Belegungen der Prozessvariablen in ein DOM Objekt geschrieben, das dann mit JSTL-TAGS ausgegeben werden kann.

Im Modus *initiatedProcesses* werden für die angegebenen Workflowtypen die aktiven Prozesse und die aktuellen Belegungen der Prozessvariable in ein DOM Objekt geschrieben, das dann mit JSTL-Tags ausgegeben werden kann.

Der Output kann mit dem debug-Parameter (&debug=true) in der URL im XML-Format ausgegeben werden.

Tagname	***verwendet z. B. in workflow/publication/workflow.jsp***
endTask	
Klasse	
org.mycore.frontend.jsp.taglibs. MCREndTaskTag	
Parameter:	
success	mandatory	PageContext-Variable, in die das Ergebnis (String Objekt) geschrieben werden soll.
processID	mandatory	die ID des aktiven Prozesses
taskName	mandatory	der zu beendende Task
transition	mandatory	die gewünschte Folgetransition

Beispielaufruf:

```
<mcr:endTask success="success" processID="${param.processID}"     taskName="$
{param.endTask}" transition="${param.transition}"/>
```

Beschreibung: Eine Task soll beendet werden und anschließend soll der Zustandsknoten über die angegebene Transition verlassen werden.

7.7 Tags für ImageViewer

Tagname		***verwendet z. B. in docdetails-*.jsp***
imageViewer		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRImageViewerTag		
Parameter:		
derivID	mandatory	Die MCR-ID des Derivates
pathOfImage	mandatory	absoluter Pfad zum Bild oder Ordner, der angezeigt werden soll
height	mandatory	Höhe des eingebetteten Fensters
width	mandatory	Breite des eingebetteten Fensters
scaleFactor	mandatory	Zoom: {0.1 .. 1.0, *fitToWidth*, *fitToScreen*}
display	mandatory	Ansicht des Viewers: *minimal* - nur Navigationsleiste *normal* - obere Menüleiste *extended* - obere + erweiterte Menüleiste
style	mandatory	Modus in dem das Bild angezeigt wird: {*thumbnail, image, text* (Metadaten)}

Beispielaufruf:

```
<mcr:imageViewer derivID="atlibri_derivate_000000000003"
  pathOfImage="${iviewLink}" display="normal" height="500" width="800"
  scaleFactor="fitToWidth" style="image"/>
```

Beschreibung: Mit diesem Tag lässt sich der ImageViewer in eine Webseite integrieren.

Tagname		***verwendet z. B. in docdetails-*.jsp***
imageViewerGetSupport		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRImageViewerGetSupportTag		
Parameter:		
derivID	mandatory	Die MCR-ID des Derivates
var	mandatory	Name der Variablen, in die das Resultat gespeichert werden soll

Beispielaufruf:

```
<mcr:imageViewerGetSupport derivID="atlibri_derivate_000000000003"
var="iviewlink" />
```

Beschreibung: Mit diesem Tag wird geprüft, ob ein Derivat darstellbare Objekte enthält. In *var* wird, wenn vorhanden, der Pfad zur Hauptdatei gespeichert. Wird eine Hauptdatei nicht unterstützt, enhält *var* einen leeren String.

Tagname		***verwendet z. B. in docdetails-*.jsp***
imageViewerGetEmbeddedThumbnail		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRImageViewerGetEmbeddedThumbnailTag		
Parameter:		
derivID	mandatory	Die MCR-ID des Derivates
pathOfImage	mandatory	absoluter Pfad zum Bild oder Ordner, der angezeigt werden soll

Beispielaufruf:

```
<mcr:imageViewerGetEmbeddedThumbnail
    derivID="atlibri_derivate_000000000003" pathOfImage="\bild1.jpg"/>
```

Beschreibung: Mit diesem Tag lässt sich mit dem ImageViewer ein Thumbnail generieren und in eine Webseite einbetten. Die Größe wir durch Properties in *\modules\module-iview\config\mycore.properties.iview* bestimmt.

Tagname		***verwendet z. B. in docdetails-*.jsp***
imageViewer		
Klasse		
org.mycore.frontend.jsp.taglibs.MCRImageViewerGetAddressTag		
Parameter:		
derivID	mandatory	Die MCR-ID des Derivates
pathOfImage	mandatory	absoluter Pfad zum Bild oder Ordner, der angezeigt werden soll
height	mandatory	Höhe des eingebetteten Fensters *(zur Zeit ungenutzt)*
width	mandatory	Breite des eingebetteten Fensters *(zur Zeit ungenutzt)*
scaleFactor	mandatory	Zoom: {0.1 .. 1.0, *fitToWidth*, *fitToScreen*}
display	mandatory	Ansicht des Viewers: *minimal* - nur Navigationsleiste *normal* - obere Menüleiste *extended* - obere + erweiterte Menüleiste
style	mandatory	Modus in dem das Bild angezeigt wird: {*thumbnail, image, text* (Metadaten)}

Beispielaufruf:

```
<mcr:imageViewerGetAddress derivID="atlibri_derivate_000000000003"
  pathOfImage="${iviewLink}" display="normal" height="500" width="800"
  scaleFactor="fitToWidth" style="image"/>
```

Beschreibung: Mit diesem Tag lässt sich ein Link erzeugen, der den ImageViewer im Vollbildmodus öffnet.

8 Tipps und Tricks

8.1 Nutzung einer anderen Datenbank

Im JSPDocPortal wird für die Speicherung der Daten das freie Produkt HSQLDB verwendet. Es hat den Vorteil, dass es direkt mit dem MyCoRe-Kern ausgeliefert wird und nicht gesondert installiert werden muss.
Leider ist diese Datenbank im Verhältnis zu anderen Produkten für den Produktionsbetrieb relativ leistungsschwach und kann nur relativ geringe Datenmengen aufnehmen. Sie eignet sich vorrangig für schlanke Desktop-Installationen, bzw. um die grundlegende Funktionalität in einer ersten Beispielanwendung darzustellen und zu testen.

Alternativ dazu können freie oder kommerzielle Datenbanken genutzt werden. Deren Installation soll im folgenden Abschnitt beschrieben werden. Dabei wird davon ausgegangen, dass weiterhin Hibernate als Zwischenschicht benutzt werden soll. Also erfolgt das Anlegen der Tabellen auch mit

```
ant create.metastore
```

8.1.1 MySQL

MySQL ist ein derzeit frei verfügbares relationale Datenbanksystem, welche zur Speicherung von Daten innerhalb des MyCoRe-Projektes eingesetzt wird.
Es besitzt eine JDBC Schnittstelle und ist SQL konform. MySQL ist Bestandteil der meisten Linux-Distributionen. Eine Windows-Version kann aus dem Internet bezogen werden und lässt sich einfach installieren.

Installation unter Linux

Der Test erfolgte mit einem MySQL 4.1.x System, höhere Versionen sollten keine Probleme bereiten.

1. Installieren Sie aus Ihrer Distribution die folgenden Pakete und danach ggf. noch vom Hersteller der Distribution per Netz angebotene Updates. Die angegebenen Versionsnummern sind nur exemplarisch.
   - **mysql-4.1...**
   - **mysql-shared-4.1...**
   - **mysql-client-4.1...**
   - **mysql-devel-4.1...**
   - **mysql-connector-java-3.1.6-...**
   - **mysql-administrator-1.0.19-... (optional)**
   - **mysqlcc-... (optional)**
2. Die Dokumentation steht nun unter */usr/share/doc/packages/mysql*.
3. Starten Sie als **root** den Datenbankserver mit dem Kommando `rcmysql start` und/oder tragen Sie den Start des MySQL-Servers für den Systemstart ein.

4. Setzen Sie als **root** das mysql-root-Passwort wie folgt:
   ```
   /usr/bin/mysqladmin -u root password rootpassword
   /usr/bin/mysqladmin -u root -h <full_host_name> password
      rootpassword
   ```
5. Die folgende Sequenz sorgt dafür, dass der mysql-**mcradmin**-Benutzer alle Rechte auf der Datenbank hat. Dabei werden bei der Ausführung von Kommandos von **localhost** aus keine Passwörter abgefragt. Von anderen Hosts aus muss *newpassword* eingegeben werden. (Hinweis: Dies ist die schnellste, aber keine sichere Methode. Dazu bitte die MySQL-Dokumentation lesen!)
   ```
   mysql –uroot -prootpassword mysql
   GRANT ALL PRIVILEGES ON *.* TO mcradmin@localhost WITH GRANT OPTION;
   quit
   ```
6. Möchten Sie, dass auch externe Hosts auf Ihr System zugreifen, so nutzen Sie das folgende Kommando. Dabei muss von anderen Hosts aus *newpassword* eingegeben werden.
   ```
   mysql –uroot -prootpassword mysql
     GRANT ALL PRIVILEGES ON *.* TO mcradmin@'%' IDENTIFIED BY
             'newpasswort' WITH GRANT OPTION;
   quit
   ```
7. Ist das Passwort einmal gesetzt, müssen Sie zusätzlich die Option -p verwenden.
8. Zum Verifizieren, ob der Server läuft, nutzen Sie folgende Kommandos
   ```
   mysqladmin -u mcradmin version
   mysqladmin -u mcradmin variables
   ```
9. Jetzt können Sie die Datenbasis für MyCoRe mit nachstehendem Kommando anlegen.
   ```
   mysqladmin -u mcradmin create mycore
   ```
10. Falls weitere Benutzer noch das Recht auf Selects von allen Hosts aus haben sollen, verwenden Sie die Kommandos
    ```
    mysql –u mcradmin mycore
    GRANT SELECT ON mycore.* TO mcradmin@'%';
    quit
    ```

Falls sie keine Verbindung mit ihrem Rechnernamen (nicht localhost) aufbauen können, kann das an den Einstellungen Ihrer Firewall oder TCPWrapper liegen.
Bei einer Firewall sollte der Port 3306 für das lokale System freigegeben werden und bei einem TCPWrapper der entsprechende Dienst (**mysqld**) in die Datei */etc/hosts.allow* geschrieben werden.

Integration in MyCoRe

Im MyCoRe-Projekt werden ein Teil der Organisations- und Metadaten in klassischen relationalen Datenbanken gespeichert. Aus Kompatibiltätsgründen zu den verschiedenen SQL-Dialekten wird das Persistenz-Framework hibernate verwendet.

Achtung, bei der Nutzung von MySQL unter Linux dürfen Sie nicht den System-Benutzer mit dem su-Kommando wechseln, MySQL wird dann im falschen User-Kontext ausgeführt und bringt ggf. Fehler.

In der Konfigurationsdatei myapplication/config/*mycore.properties* legen Sie im Parameter **MCR.hibernate.connection.driver_class** fest, welcher JDBC-Treiber verwendet werden soll.

Weiterhin müssen Sie die Variable **MCR.hibernate.connection.url** anpassen, die die JDBC URL für Verbindungen zu Ihrer Datenbank festlegt. Achten Sie bei Verwendung von MySQL darauf, dass der richtige Datenbank-User (per Default mcradmin) angegeben ist. Sollte es zu Problemen beim Zugriff auf MySQL kommen, versuchen Sie die Adresse **127.0.0.1** durch **localhost** zu ersetzen.

Für die Integration von MySQL steht schon ein vordefinierter Konfigurationsblock in der Konfigurationsdatei *mycore.properties* zur Verfügung. Die gleichen Parameter setzen Sie auch für die WorkflowEngine in der Datei *myapplication/config/ workflow/ jbpm_hibernate.cfg.xml*. Hier sollten Sie darauf achten, dass Sie einen anderen Datenbankname wählen als den der Datenbank (in mycore.properties), die die MyCoRe-Daten enhält.

Bsp. MySql Konfiguration in mycore.properties:

```
MCR.hibernate.dialect=org.mycore.backend.hibernate.dialects.MCRMySQLMyISAMDiale
ct
MCR.hibernate.connection.driver_class=org.gjt.mm.mysql.Driver
MCR.hibernate.connection.url=jdbc:mysql://127.0.0.1/mycore?
user=mcradmin&autoReconnect=true
MCR.hibernate.connection.username=mcradmin
MCR.hibernate.connection.password=********
```

TIPP: Diese Einträge sind sinnvollerweise in der Datei *mymycore.properties* in *myapplication/config* unterzubringen, so werden sie bei einem Update von JSPDocportal aus dem CVS nicht überschrieben.

Bsp. MySql Konfiguration in jbpm_hibernate.cfg.xml:

```
<property name="hibernate.dialect">
      org.mycore.backend.hibernate.dialects.MCRMySQLMyISAMDialect
</property>
<property name="hibernate.connection.driver_class">
      org.gjt.mm.mysql.Driver
</property>
<property name="hibernate.connection.url">
      jdbc:mysql://127.0.0.1/mycore_workflow
</property>
<property name="hibernate.connection.username">mcradmin</property>
<property name="hibernate.connection.password">********</property>
```

8.2 Einbindung virtueller Host-Namen für Apache-Web-Server

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

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

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

Installation von httpd-2.0.54 mit Einbindung von mod_proxy

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

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

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

Zur Kontrolle der Übersetzung können Sie mittels des Kommandos */usr/local/apache2/bin/httpd -l* die Einbindung der Proxy-Module testen. Die Auflistung muss die beiden Module apr und apr-util anzeigen.

Die Verbindung von Tomcat5 und Apache2

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

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

Abbildung 8.1: Ausschnitt der httpd.conf

Die folgenden Änderungen basieren auf den im Laufe der Installation benutzten Tomcat5 Konfiguration, wie Sie im User Guide beschrieben ist.

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

Abbildung 8.2: Änderungen in der server.xml

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

8.3 URL-Rewriting im Tomcat

Für eine einfachere Verwaltung von URLs und leicht konfigurierbares URL-Rewriting wurde der Java Web Filter **URLRewriteFilter** von Paul Tuckey integriert (http://tuckey.org/urlrewrite/).

Installation

Um das Tool zu installieren muss das JAR in das jspdocportal/lib Verzeichnis kopiert werden.

Registrierung

Der Filter wird in der web.xml wie folgt registriert ...

```
<filter>
   <filter-name>UrlRewriteFilter</filter-name>
   <filter-class>
      org.tuckey.web.filters.urlrewrite.UrlRewriteFilter
   </filter-class>
</filter>

<filter-mapping>
   <filter-name>UrlRewriteFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Abbildung 8.3: Anpassungen in web.xml für URLRewriteFilter

Für die Registrierung können zusätzlich Parameter definiert werden. Eine Beschreibung entnehme man dem UrlRewriteFilter-Handbuch (http://tuckey.org/urlrewrite/manual/2.6/).

Konfiguration

Die Konfiguration erfolgt über die Datei **urlrewrite.xml**. Sie sollte im config-Verzeichnis der Anwendung angelegt werden. Während des Build-Prozesses wird sie in das WEB-INF Verzeichnis kopiert.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE urlrewrite PUBLIC "-//tuckey.org//DTD UrlRewrite 2.6//EN"
      "http://tuckey.org/res/dtds/urlrewrite2.6.dtd">
<urlrewrite>
   <rule>
      <from>^/cpr$</from>
      <to type="redirect">nav?path=left.collections.cpr</to>
   </rule>
</urlrewrite>
```

Abbildung 8.4: urlrewrite.xml

Für eine genauere Beschreibung sei wiederum auf das UrlRewriteFilter Handbuch (http://tuckey.org/urlrewrite/manual/2.6/) verwiesen.

8.4 Eclipse IDE für JSPDocPortal mit MyCoreSample einrichten

Die folgende „Kurz-Anleitung" dient als Hilfe um JSPDocPortal in die Entwicklungsumgebung Eclipse zu laden. Als Datenbank wird in dieser Anleitung MySQL gewählt. Die Metadaten- und die Volltextindizierung erfolgt wie auch in der Beispielanwendung mit Lucene.

8.4.1 Download und Installation der Tools

Titel	*Package*
Java	Java SE JDK 5.0 oder besser (http://java.sun.com/javase/downloads/index.jsp)
Eclipse	Eclipse IDE for Java EE Developer (http://www.eclipse.org/downloads/download.php ?file=/technology/epp/downloads/release/europa/winter/ eclipse-jee-europa-winter-win32.zip)
SysDeo TomcatPlugin	Tomcat Plugin v. 3.2.1 (http://www.eclipsetotale.com/tomcatPlugin.html)
MySql	MySQL Community Server 5.0 (http://dev.mysql.com/downloads/) MySQL GUI Tools (Adminitrator, QueryBrowser) (http://dev.mysql.com/downloads/gui-tools/5.0.html)
Tomcat	Tomcat 6.0 zip-Datei (braucht nicht installiert werden) (http://tomcat.apache.org/download-60.cgi)
Tomcat Native Lib	Apache Portable Runtime *(Win: DLL ins Systemverzeichnis kopieren)* (http://tomcat.apache.org/tomcat-6.0-doc/apr.html)
JBoss jBPM	*folgt ...*

8.4.2 MySQL vorbereiten

MySQL (als root): neuen Benutzer „mcradmin" anlegen

```
GRANT ALL PRIVILEGES ON *.* TO 'mcradmin'@'%'
IDENTIFIED BY '*password*' WITH GRANT OPTION;
```

8.4.3 Eclipse-Workspace einrichten

Windows: Verzeichnis C:\workspaces\mycoresample anlegen

Windows: Link auf Desktop oder Batch-Datei: (Anlegen + Ausführen)

```
C:\eclipse\eclipse.exe
      -data "c:\mycore-workspace\myapplication\projects"
      -vm "c:\Programme\Java\jdk1.5.0_06\bin\javaw.exe"
      -vmargs -Xmx512m
```

Eclipse: Preference-Dialog (Menü: Window – Preferences) öffnen
- Tomcat: Tomcat-Version: 5.x + Tomcat Home setzen
- Java/Installed JREs: **JDK** 1.5.x hinzufügen und auswählen

Eclipse: File -> New -> Project
Java/Tomcat Project „myapplication" anlegen
Anwendungs-URI: /myapplication
Webapp-root: /working/webapps/myapplication

8.4.4 Code einfügen per SVN

Eclipse: File -> New -> Other -> CVS -> Checkout Projects from CVS
Server: server.mycore.de
User: anonymous PW: <leer>
root-directory: /cvs
protocoll: pserver

-> „Use an exisiting module" auswählen
im CVS die Zweige jspdocportal und mycore auswählen

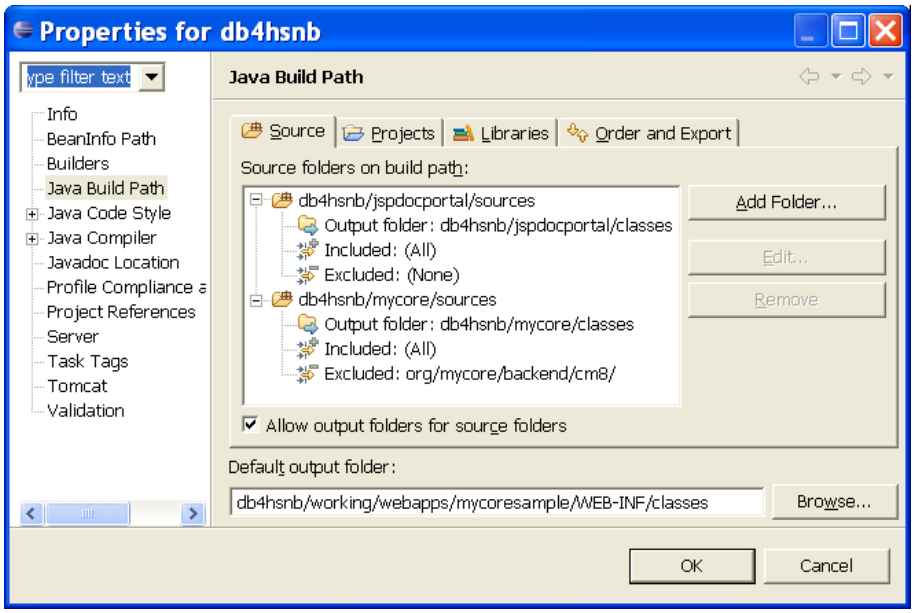
Checkout into an existing project : „mydigibib"

from: HEAD -> Finish Project -> Refresh

8.4.5 Pfade einrichten

Eclipse: Properties: Java Build Path einstellen
Libraries: Add External Jars
- mydigibib\mycore\lib*.*
- mydigibib\jspdocportal\lib*.*

Source:



8.4.6 Build MyCoRe

Eclipse: Kopieren von *mycore\config\build.properties.template*
nach *build.properties*

ANT (mycore): info-Target ausführen, jar-Target ausführen

MySQL (als mcradmin): CREATE DATABASE mycore

8.4.7 Build JSPDocPortal

Eclipse: *jspdocportal\build.xml* Property setzen

```
<property name="MYAPPLICATION.DIRNAME" value="myapplication" />
```

ANT (jspdocportal): create.my.own.application

Eclipse: *myapplication\build.xml.template* nach *build.xml* umbenennen

Eclipse: *myapplication\config\mymycore.properties* nach *mymycore.properties* umbenennen
Properties anpassen in *myapplication\config\mymycore.properties*:

```
MCR.datadir=c:/workspaces/mycoresample/working_content
MCR.basedir=c:/workspaces/mycoresample/projects/jspdocportal
MCR.baseurl=http://localhost:8080/myapplication
```

Eclipse: Klassifikationen kopieren (ggf. anpassen)
von */jspdocportal/content/classification/ignore*
nach */myapplication/content/classification*

Eclipse: *myapplication\config\hibernate\hibernate.cfg.xml.template*
nach *hibernate.cfg.xml* umbennen
hibernate.cfg.xml: Absatz „hsqldb" auskommentieren
Absatz „MySQL" entkommentieren + anpassen

Eclipse: Property in *myapplication\build.xml* anpassen

```
<property name="MYAPPLICATION_NAME" value="myapplication" />
```

8.4.8 Einrichten der WorkflowEngine

MySQL (als mcradmin): CREATE DATABASE mycore_workflow

Eclipse: COPY *myapplication\config\workflow\jbpm_hibernate.cfg.xml.template*
-> *myapplication\config\workflow\jbpm_hibernate.cfg.xml*

j*bpm_hibernate.cfg.xml* für MySQL anpassen

Achtung:
der MySQL JDBC Treiber ist nicht Bestandteil der Beispielanwendung. Bitte besorgen Sie sich den Ihrer MySQL-Version entsprechenden Treiber von der MySQL Website.

z.B. mysql-connector-java-3.0.14-production-bin.jar

und kopieren Sie ihn in das Verzeichnis *myapplication/lib*.

8.4.9 Initialisierung (Ausführen von ANT Targets in Eclipse)

In Eclipse die folgenden Ant Targets der build.xml (in myapplication) ausführen

- Initial nur beim erstmaligen Installieren notwendig bzw. bei Änderung an einzelnen Modulen (neue Nutzer, Ändern der Prozessdefinition, ...).

merge.my.properties

create.schema (Datenmodell Schema generieren)

jar (Builden der Applikation)

create.directories (Speicherort für Derivate)

create.metastore (MyCore Tabellen in MySQL erzeugen)

create.users (MyCore Benutzer in die Datenbank schreiben)

create.class (Laden der Klassifikationen)

create.scripts (.cmd Dateien für MyCoRe CLI)

create.workflowengine.database (DB anlegen)

deploy.workflow.processdefinition (Prozessdefinitionen in DB schreiben)

create.genkeys (Keystore zum Signieren von jars)

8.4.10 Webapplikation installieren und testen

Zum Erzeugen der Anwendung (auch wenn lediglich Java-Klassen oder JSP-Dateien geändert wurden das ANT-Target (aus build.xml in mycoresample_app)

war oder webapps

ausführen.

Tomcat-Debug-Dialog:	JSP-Verzeichnisse zur Source hinzufügen (wenn JSPs debugged werden sollen)
Tomcat Server starten	(Icon in Eclipse) oder Debug-Dialog: Java Application: „Tomcat 5.x“
Im Webbrowser:	http://localhost:8080/myapplication aufrufen.

9 Anhang

9.1 Abbildungsverzeichnis

9.2 Tabellenverzeichnis