MyCoRe Progammer Guide

Frank Lützenkirchen
Jens Kupferschmidt
Detlev Degenhardt
Johannes Bühler

29. März 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Dieser Teil der Dokumentation ist vor allem für Applikationsprogrammierer gedacht. Er beschreibt die Designkriterien und ihre Umsetzung in der vorliegenden Version 1.0 . Mit Hilfe dieser Dokumentation sollte es Ihnen möglich sein, Details von MyCoRe zu verstehen und eigene Anwendungen konzipieren und implementieren zu können. Die Schrift wird stetig erweiteret.

1.Softwareentwicklung

1.1 Vorabbemerkungen

In diesem Kapitel soll kurz in die Design- und Entwicklungsmechanismen des MyCore-Projektes eingeführt werden.

1.2 CVS-Zugang

Der Zugang zum CVS-Server des MyCoRe Projekts für Entwickler erfolgt nach Freischaltung eines Accounts über SSH. Nach dem Freischalten sind folgende Umgebungsvariablen zu setzten:

```
CVS_RSH=ssh
CVSROOT=:ext:mcr_username@server.mycore.de:/cvs
export CVS_RSH CVSROOT
```

Es empfielt sich zuerst die MyCoRe Quellen herunterzuladen.

```
cvs -d:ext:mcr_username@server.mycore.de:/cvs checkout mycore
```

Danach können sie mit

```
cvs -d:ext:mcr_username@server.mycore.de:/cvs commit -m "Kommentar fürs CVS" file
```

Daten einstellen. Voraussetzung ist ein CVS-Login. Dieses können Sie bei Frank Lützenkirchen beantragen.[1] Soll in ein bestehendes Projekt eine neue Datei integriert werden, so legt man sie zunächst lokal im vorgesehenen (und bereits ausgecheckten!) Verzeichnis an. Dann merkt man sie mittels `cvs add<filename>` vor. Um sie dann global zuregistrieren, erfolgt ein (verkürzt): `cvs commit<filename>` . Neue Unterverzeichnisse werden auf die gleiche Weise angelegt. Weitere und sehr ausführliche Informationen gibt es zu Hauf im Internet[2].

1.3 Entwicklungsumgebungen

1.4 Tools

1luetzenkirchen@bibl.uni-essen.de

2http://www.cvshome.org/docs/

http://cvsbook.red-bean.com/translations/german/

http://panama.informatik.uni-freiburg.de/~oberdiek/documents/OpenSourceDevWithCVS.pdf

http://www.selflinux.org/selflinux/pdf/cvs_buch_kapitel_9.pdf

2.Allgemeines der Implementierungen

2.1 Klassen, Pakete und Verantwortlichkeiten

2.2 Allgemeine Klassen / Exception-Modell / MCRCache

2.3 Das API-Konzept allgemein

2.4 Die Session-Verwaltung

Mehrere verschiedene Benutzer und Benutzerinnen (oder allgemeiner Prinzipale) können gleichzeitig Sitzungen mit dem MyCoRe-Softwaresystem eröffnen. Während einer Sitzung werden in der Regel nicht nur eine sondern mehrere Anfragen bearbeitet. Es ist daher sinnvoll, kontextspezifische Informationen wie die UserID, die gewünschte Sprache usw. Für die Dauer der Sitzung mitzuführen. Da das MyCoRe-System mit mehreren gleichzeitigen Sitzungen konfrontiert werden kann, die zudem überverschiedene Zugangs wege etabliert sein können (z.B. Servlets, Kommandozeilenschnittstelle oder Webservices), muss das System einen allgemein verwendbaren Kontextwechsel ermöglichen.

Bei der Bearbeitung einer Anfrage oder Transaktion muss nicht jede einzelne Methode oder Klasse Kenntnis über die Kontextinformationen besitzen. Daher ist es sinnvoll, die Übergabe des Kontextes als Parameter von Methode zu Methode bzw. Von Klasse zu Klasse zu vermeiden. Eine Möglichkeit, dies zu bewerkstelligen ist der Einsatz von sog. Thread Singletons oder thread-local Variablen. Die Idee dabei ist, den Thread der den Request bearbeitet als Repräentation des Request selbstan zusehen. Dazu müssen die Kontextinformationen aller dings an den Thread angebunden werden, was seit Java1.2 mit Hilfe der Klassen java.lang.ThreadLocal bzw. java.lang.InheritableThreadLocal möglich ist. Jeder Thread hat dabei seine eigene unabhängig initialisierte Kopie der Variable. Die set() und get() Methoden der Klasse ThreadLocal setzen bzw. Geben die Variable zuruck, die zum gerade ausgeführten Thread gehört. Die Klassen der Sessionverwaltung von MyCoRe sind auf Basis dieser Technologie implementiert (siehe Abbildung).

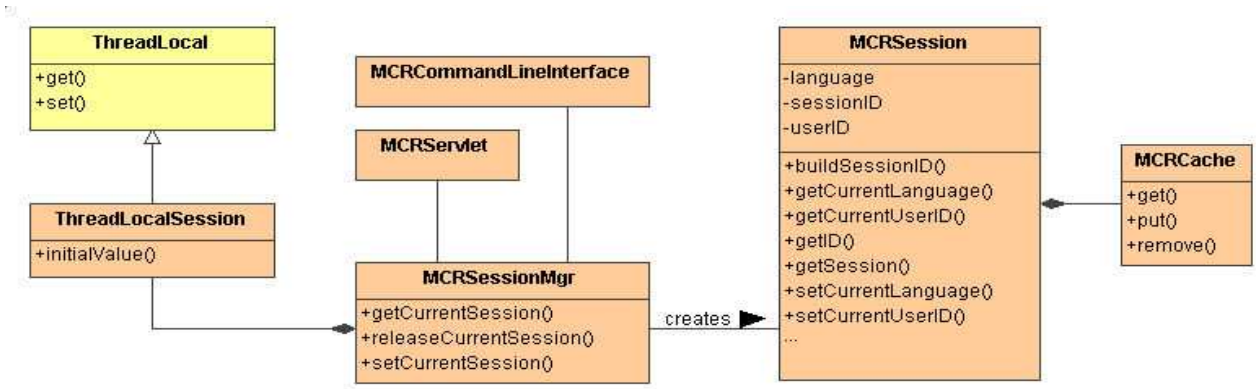


Abbildung 2.4.1: Die Klassen der Sessionverwaltung

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

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

Damit die Instanzen von MCRSession als thread-local Variablen an den aktuellen Thread angebunden werden können, werden sie nicht direkt sondern über die statische Methode getCurrentSession() der Klasse MCRSessionMgr erzeugt und später gelesen. Beim ersten Aufruf von getCurrentSession() in einem Thread wird über die von java.lang.Threadlocal erbende, statische innere KlasseThreadLocalSession gewährleistet, dass eine eindeutige Instanz von MCRSession erzeugt und als thread-local Variable abgelegt wird. Der Zugriff auf die thread-local Variablen eines Threads kann nur über die Klasse ThreadLocal (bzw. InheritableThreadLocal) erfolgen. Auf diese Weise ist sichergestellt, dass bei nachfolgenden Aufrufen von getCurrentSession() genau die zum aktuellen Thread gehörende Referenz auf die Instanz vonMCR Session zuruckgegeben wird.

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

Die Methode MCRSessionMgr.releaseCurrentSession() sorgt dafür, dass das thread-local Session-Objekt eines Threads durch ein neues, leeres Objekt ersetzt wird. Dies ist in Thread-Pool-Umgebungen wichtig, weil es sonst moglich bzw. sogar wahrscheinlich ist, dass Kontextinformationen an einem Thread angebunden sind, dieser Thread aber bei seiner

Wiederverwendung in einem ganz anderen Kontextarbeitet. Code-Beispiele zur Verwendung der Session-Verwaltung finden sich in org.mycore.frontend.servlets.MCRServlet.doGetPost().

2.5 Das Vererbungsmodell

In MyCoRe ist innerhalb des Datenmodells für die Metadaten die Möglichkeit einer Vererbung vorgesehen. Dies ist fest in den Kern implementiert und wird ausschließlich durch die steuernden Metadaten des jeweiligen Datensatzes festgelegt. Das heißt, für eine Datenmodell-Definition (z. B. document) können Datensätze mit (Buch mit Kapiteln) und ohne (Dokument) nebeneinander eingestellt werden. Wichtig ist nur, dass die Vererbung nur innerhalb eines Datenmodells oder eines, welches die gleiche Struktur aufweist, jedoch andere Pflichtfelder hat, funktioniert. Vererbung zwischen Datenmodellen mit verschieden Metadaten ist ausgeschlossen.

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

```
<mycoreobject ... >
...
<metadata xml:lang=“de“>
<titles class=“MCRMetaLangText“ heritable=“true“ notinherit="false"..>
<title xml:lang=“de“>Buchtitel</title>
</titles>
<authors class=“MCRMetaLangText“ heritable=“true“ notinherit="false"..>
<author xml:lang=“de“>Erwin der Angler</author>
</authors>
<sizes class=“MCRMetaLangText“ heritable=“false“ notinherit="false"..>
<size xml:lang=“de“>100 Seiten</size>
</sizes>
...
</metadata>
....
</mycoreobject>
```

Abbildung 2.5.1: Auszug aus dem Metadaten-Objektes des Elternsatzes

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

Abbildung 2.5.2: Auszug aus dem Metadaten-Objektes des Kindsatzes

Die Beiden Datensätze sollen folgendes Szenario widerspiegeln:

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

Entscheidend für die Umsetzung sind folgende Dinge:

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

Die Ausgabe des Eltern-Datensatzes nach einer Query entspricht dem der Dateneingabe, lediglich im XML-structure-Teil wurde ein Verweis auf die Kinddaten eingetragen.[3]

3 Siehe XML-Syntax im User Guide.

Das unten stehende Listing zeigt den Kind-Datensatz, wie er von System nach einer erfolgreichen Anfrage im Resultats-Container zurückgegeben wird. Dabei ist deutlich die Funktionalität der MyCoRe-Vererbungsmechanismen zu erkennen.

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

Abbildung 2.5.3: XML-Syntax eines Kind-Datensatzes als Query-Resultat

3.Funktionsprinzipien und Implementierungen von Kernkomponenten

3.1 Das QueryModell von MyCoRe

MyCoRe bemüht sich, das Syntax-Modell der Suchabfragen an die existierenden Standards des W3C für XML Path Language (XPath) Version 1.0 (W3C Recommendation 16. November 1999) anzugleichen, da die derzeit häufigste Abfrage-Syntax im XML-Bereich ist. Dabei muss aber auch Rücksicht auf die Spezifik des MyCoRe-Systems zur Suche von Objekten und Metadaten genommen werden. Daher ist MyCoRe nicht 100% XPath/XQuery -konform, es wird sich jedoch bemüht, bestmöglich die Spezifkationen einzuhalten. Grund für diese Abweichungen sind die Praxisorientierung des MyCoRe-Systems, vor allem im Bereich digitaler Bibliotheken. So sollen verschiedene Persistence-Systeme zum Einsatz kommen, welche sehr unterschiedliche Abfrage-Syntax, auch außerhalb der XML-Welt, implementieren. Daher stellt der in diesem Abschnitt beschriebene Syntax nur einen recht keinen Teil der Möglichkeiten der W3C Spezifikationen dar. Er genügt aber in der Praxis, um recht komplexe Projekte zu realisieren. Eine Annäherung an Xpath2.0 und Xquery 1.0 wird erster folgen, wenn die Standardisierungsphase im wesentlichen abgeschlossen ist.

MyCoRe muss neben der eigentlichen Query noch erfahren, welche Datenmodell-Typen abzufragen sind. Auch dafür ist die Ursache in der Persistence-Unabhängigkeit von MyCoRe zu suchen. Es ist notwendig, die Typen auf die entsprechenden Stores zu mappen, damit die Suche erfolgreich ist. Als Beispiel soll hier der Content Manager 8 ItemType oder die Umsetzung unter eXist stehen. Möglich ist sowohl einzelne Typen, wie auch eine Liste davon anzugeben. Die Liste ist ein String mit folgendem Syntax:

```
Stringtype_list="type1[,...]"
```

Wichtig ist nur, dass die Elemente, nach denen gefragt wird, in allen Datenmodellen der Typen vorkommen (sonst könnte das Ergebnis der Suche eine leere Resultatsliste sein). Normalerweise wird die Type-Liste in der Applikation via Konfiguration festgelegt[4] und an die entsprechenden Query-Schnittstellen im API (MCRQueryBase) übergeben, z. B. durch das SearchMaskServlet.

3.1.1 Operatoren

MyCoRe verwendet nur die folgenden Operatoren innerhalb eines Test. Dies begründet sich mit der Kompatibilität und Umsetzung gegenüber den einzelnen Persitence-Layern. Wenn weitere Operatoren benötigt werden, so müssen diese in **ALLEN** Persitence-Implementierungen eingebaut werden.

```
SingleQuery        :: OutPath[ Tests ]
                      Tests
Tests              :: Test{ AND | OR Test ... }
```

4 mycore.properties: MCR.type_alldocs=document

```
Test              :: InPath Operator Value
```

- Operator = - ist für alle Values zulässig
- Operator != - ist für alle Values zulässig
- Operator < - ist nur für Datums- und Zahlenangaben zulässig
- Operator <= - ist nur für Datums- und Zahlenangaben zulässig
- Operator > - ist nur für Datums- und Zahlenangaben zulässig
- Operator >= - ist nur für Datums- und Zahlenangaben zulässig
- Operator like – versucht im Value den angegebenen String zu finden, als Wildcard ist * zulässig
- Operator contains – arbeitet wie like, ist eien TextSearch-Engine verfügbar, so erfolgt die linguistische Suche darin

Die Operatoren **like** und **contains** sind eine Ergänzung von MyCoRe um mit Textsuch-Mechanismen arbeiten zu können. Sie sind in der von MyCoRe benötigten Syntax-Form so nicht Bestandteil der W3C Spezifikationen, haben sich aber in der Praxis bewährt.

3.1.2 Pfadangaben

Die Pfadangaben für eine Single Query können einfach oder einmal geschachtelt sein, jenachdem, wie die einzelnen Text- bzw. Attributknoten des XML-Datenmodells logisch zusammengehören. Alle hier aufgeführten Möglichkeiten sind relativ zu XML-Dokument-Wurzeln. Andere Pfadangaben wie beispielsweise attribute:: für @ sind aus Gründen der Kompatibilität zu den einzelnen Persitence-Layern nicht erlaubt (und in der Praxis auch nicht erforderlich).

```
OutPath   :: a
             a/b
InPath    :: SpecialNodeFuncition
             @d
             text()
             c
             c/text()
             c/@d
             c/d/@e
             c/d/text()
SpecialNodeFuncition::       ts()
                             text()
                             doctext()
```

Hier noch einige Hinweise:

- ***** als SpecialNodeFuncition darf allein nur in einer Single Query ohne OutPath angegeben werden. Es erfolgt dann die Suche über alle für TextSearch markierte Metadaten.
- **text()** als SpecialNodeFuncition darf allein nur in einer Single Query ohne OutPath angegeben werden. Es wird nach 'ts()' überführt.
- **doctext()** als SpecialNodeFuncition ist für die Abfrage des TextSearch des Dokument-Objektes

vorgesehen.

- In MyCoRe kann bei Bedarf der der Applikation um weitere SpecialNodeFuncition ergänzt werden.

Nun einige gültige Beispiele:

```
text()contains"..."
*like"..."
@ID="..."
metadata/rights/right="..."
metadata/rights/right/text()="..."
metadata/masse/mass[text()="..."and@type="..."]
doctext()contains"..."
```

3.1.3 Abfragen von Objekt-Metadaten

Unter Objekt-Metadaten sind alle Datenmodell-Typen zu verstehen, welche NICHT **class** oder **derivate** sind[5]. Alle XML-Files der Objekt-Metadaten Typen haben als Master-Tag /mycoreobject und genau auf diesen Knoten als Return-Value zielen auch alle Queries unter MyCoRe. Der allgemeine Syntax ist also:

```
Query::             OneQuery{ AND | OR OneQuery { AND | OR ... }}
OneQuery::          /mycoreobject[SpecialNodeFuncition]
```

Dies bedeutet, es können mehrere Abfragen hintereinander mit AND oder OR verknüpft werden. Für jedes Metadatum des Datenmodells ist eine OneQuery zu formulieren. Das diese Anfragen alle auf denselben Datenraum laufen,dafür sorgt die oben beschriebene Festlegung des Type-Elementes. Somit erhalten Sie ein korrektes Gesamtergebnis.

3.1.4 Das Resultat der Query

Alle Antworten als Resultat der Anfrage werden in einem XML-Conatiner zusammengefasst und der Anwendung zurückgegeben. Der Aufbau des Containers ist dabei Persitence-unabhängig. Nachfolgend die XML-Struktur des Resultates einer Query:

5 Die Typen class und derivate sind reservierte Typen.

```
<mcrresults parsedByLayoutServlet="...">
<mcrresult host="..." id="..." rank="0" hasPred="..." hasSucc="...">
<mycoreobject ...>
</mycoreobject>
</mcrresult>
...
</mcrresults>
```

Abbildung 3.1.1: XML Syntax des Query-Resultates

- Das Attribut **parsedByLayoutServlet** ist ein Flag, welches eine etwaige Vorverarbeitung des Ergebnisses durch das LayoutServlet anzeigt.
- Das Attribut **host** beinhaltet entwerden '**local**' oder den Hostalias des Servers, von dem das Resultat stammt.
- Das Attribut **id** ist die entsprechende MCRObjectID des Resultates.
- Das Attribut **hasPred** gibt mit **true** an, dass dieses Resultat einen Vorgänger hat. Dies ist für die Präsentation der Einzeldaten von Interesse.
- Das Attribut **hasSucc** gibt mit **true** an, dass dieses Resultat einen Nachfolger hat. Dies ist für die Präsentation der Einzeldaten von Interesse.

3.1.5 Abfragen von Derivaten

Für die Derivate gelten die selben Regeln für die Queries. Bachten Sie jedoch, dass das Datenmodel fest vorgeschieben ist und das Master-Tag anders heisst. Das Resultat auf eine Anfrage wird auch in einen **mcrresults**-Container verpackt.

```
Query     :: OneQuery{ AND |OR OneQuery { AND | OR ... }}
OneQuery  :: /mycorederivate[SpecialNodeFuncition]
```

3.1.6 Abfragen von Klassifkationen

Klassifkationen werden in MyCoRe-Anwendungen erfahrungsgemäß am häufigsten abgefragt. DieKlassifkationen können nur nach einem sehr festen Schema abgefragt werden, wobei entwerder die gesamte Klassifkation oder nuer eine einzelne Kategorie zurückgegeben wird. DieXML-Struktur entspricht dabei immer der einer Klassifkation.

```
Query::   /mycoreclass[@ID="..." { and @category like "..."}]
```

3.2 Die Benutzerverwaltung

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

3.2.1 Die Geschäftsprozesse der MyCoRe Benutzerverwaltung

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

Ein Use-Case Diagramm (sieheAbbildung) soll eine Reihe typischer Geschäftsprozesse des Systems zeigen (ohne dabei den Anspruch zu haben, alle Akteure zu benennen oder alle Assoziationen der Akteure mit den Geschäftsprozessen zu defnieren).

Offensichtlich dürfen nicht alle Akteure des Systems die Berechtigung haben, alle Geschäftsprozesse durchführer zu können. Daher musse in System von Privilegien und Regeln implementiert werden: Benutzer/innen haben Privilegien (z.B. die Berechtigung, neue Benutzer/innen zu erstellen) Die Vergabe der Privilegien wird durch die Mitgliedschaft der Benutzer/innen in Gruppen geregelt. Darüber hinaus muss das System definierten Regel gehorchen. So genügt z.B. Das Privileg '**add user to group'** allein nicht, um genau das Hinzufügen eines benutzers zu einer Gruppe definieren zu können. Die Regel ist in diesem Fall, jeder andere Benutzer mit diesem Privileg aber maximal Zugehörigkeit zu Gruppen vergeben kann, in denen er oder sie selbst Mitglied ist. Auf diese Weise wird verhindert, dass sich ein Benutzer oder eine Benutzerin selbst höhere Privilegien zuweisen kann. Die Privilegien und Regeln der MyCoRe-Benutzerverwaltung werden weiter unten ausgeführt.

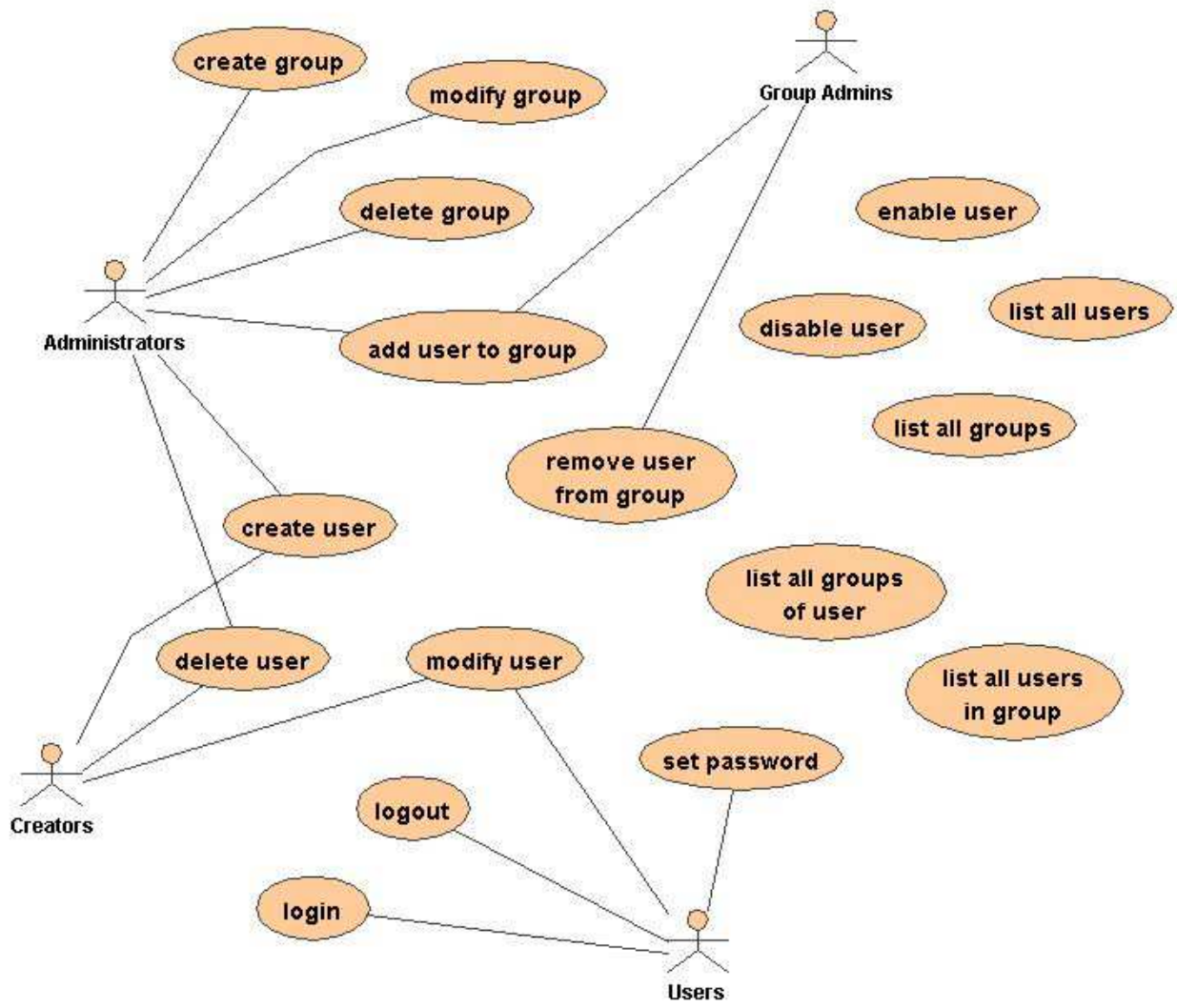


Abbildung 3.2.1: Geschäftsprozesse der Benutzerverwaltung in MyCoRe

3.2.2 Benutzer, Gruppen, Privilegien und Regeln

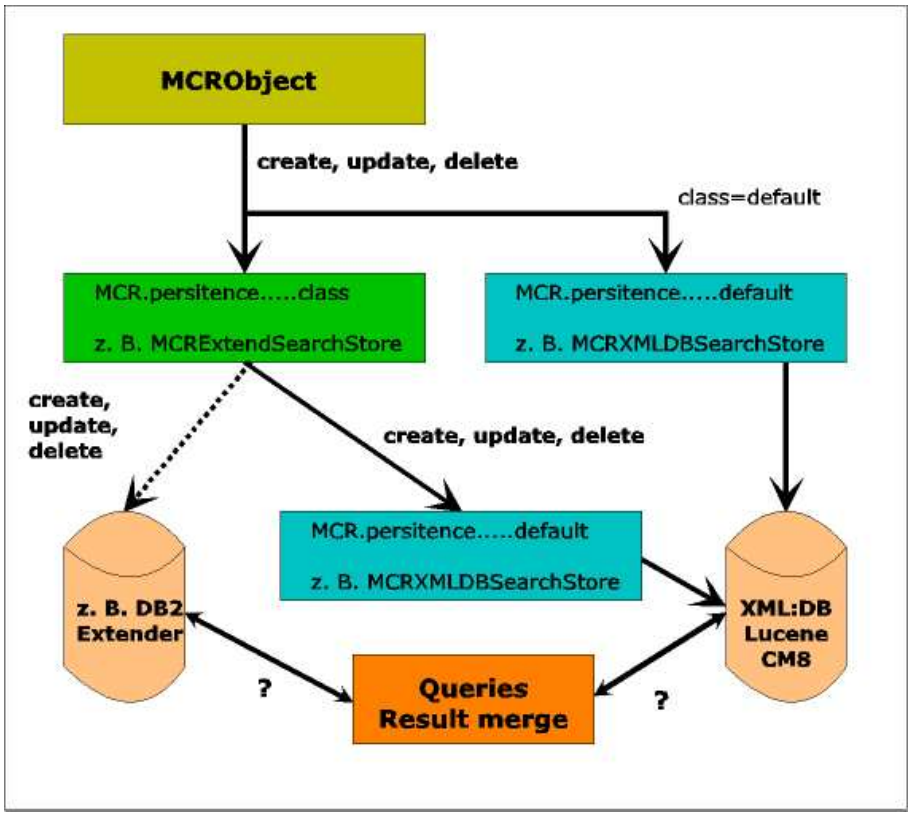
Die Attribute von Benutzern/innen des Systems können in drei Bereiche klassifiziert werden, den account-Informationen wie ID, Passwort, Beschreibung usw., den Adress-Informationen wie Name, Anrede, Fakultätszugehörigkeit usw. sowie den Informationen über die Mitgliedschaft zu Gruppen. Die aktuell implementierten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

3.3 Der Backend-Store

Im Backend-Bereich muss zwischen den Stores für die Metadaten und denen für die eigentlichen Objekte unterschieden werden. Während erstere recht umfangreiche Suchen auf einzelne Datenfelder gestatten müssen, dienen letztere meist nur der Datenspeicherung. Lediglich in einigen Fällen wird diese mit einer Volltextsuche ergänzt.

3.3.1 Backend-Stores für die Metadaten allgemein

Momentan sind im MyCoRe-Kern zwei Backends für die Speicherung der Metadaten implementiert. Das System ist so gestaltet, dass eine Ergänzung dieser Stores durch weitere applikationsabhängige Datenbasen einfach möglich ist. Hierzu kann entwerder der Standard-Store entsprechend erweitert werden oder die im Applikationsbereich vorhandene Klasse MCRExtendStore wird entsprechend ausgebaut, was der bessere Weg ist. Welche Store-Klassen Verwendung finden, wird in der Konfiguration festgelegt, für XML:DB in *mycore.properties.xmldb* und für CM8 in *mycore.properties.cm8*. Per Standardeinstellung sind die Klassen für den Store und den Default gleich, so dass der Extender nicht angesprochen wird. Den ganzen Zusammenhang soll das folgende Schema verdeutlichen.



3.3.2 Das Metadaten-Backend für IBM Content Manager 8.2

Hinweis:

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

3.3.3 Das Metadaten-Backend für XML:DB

Hinweis:

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

3.4 Die Frontend Komponenten

3.4.1 Erweiterung des Commandline Tools

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

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

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

Die nachfolgende Skizze soll einen Überblick über die Zusammenhänge der einzelnen Java-Klassen im Zusammenhang mit der Nutzerseitigen Erweiterung des Commandline-Tools geben.

Zeichnung folgt

Es ist relativ einfach, weitere Kommandos hinzuzufügen. Im MyCoReSample sind bereits alles nötigen Muster vorhanden.

1. Im Verzeichnis *~/docportal/sources/org/mycore/frontend/cli* finden Sie eine Java-Klasse MCRMyCommand.java. Diese ist ein Muster, kopieren Sie sie in eine Java-Klssse z. B. MCRTestCommand.java . Die Klasse **muss** im Package **org.mycore.frontend.cli** liegen.
2. Ersetzen Sie alle MCRMyCommand-String durch MCRTestCommand .
3. Im Constructor werden nun alle neuen Kommandos definiert. Hierzu werden der ArrayList **command** jeweils zwei weitere Zeilen hinzugefügt. Die erste enthält den Text-String für das Kommando. Stellen wo Parameter eingefügt werden sollen sind mit {...} zu markieren, wobei ... eine fortlaufende Nummer beginnend mit 0 ist.[6] In der Zweiten Zeile ist nun der Methodenaufruf anzugeben. Für jeden Parameter ist das Schlüsselwort **String** anzugeben.
4. Nun muss das eigentliche Kommando als Methode dieser Kommando-Klasse implementiert

6 siehe Beispielcode

werden. Orientieren Sie sich dabei am mitgelieferten Beispiel.[7]

5. Compilieren Sie nun die neue Klasse mit cd ~/docportal; ant jar
6. Als letztes müssen Sie die Klasse in das System einbinden. Die mit dem MyCoRe-Kern mitgelieferten Kommandos sind bis auf die Basis-Kommandos über die Property-Variable **MCR.internal_command_classes** in *~/docportal/mycore.properties* dem System bekannt gemacht. Für externe Kommandos steht hierfür im Konfigurations-File *mycore.properties.application* die Variable **MCR.external_command_classes** zur Verfügung. Hier können Sie eine mit Komma getrennte Liste Ihrer eigenen Kommando-Java-Klassen angeben.
7. Wenn Sie nun `mycore.sh` bzw. `mycore.cmd` starten und DEBUG für den Logger eingeschaltet haben, so sehen Sie Ihre neu integrierten Kommandos.

3.4.2 Das Zusammenspiel der Servlets mit dem MCRServlet

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

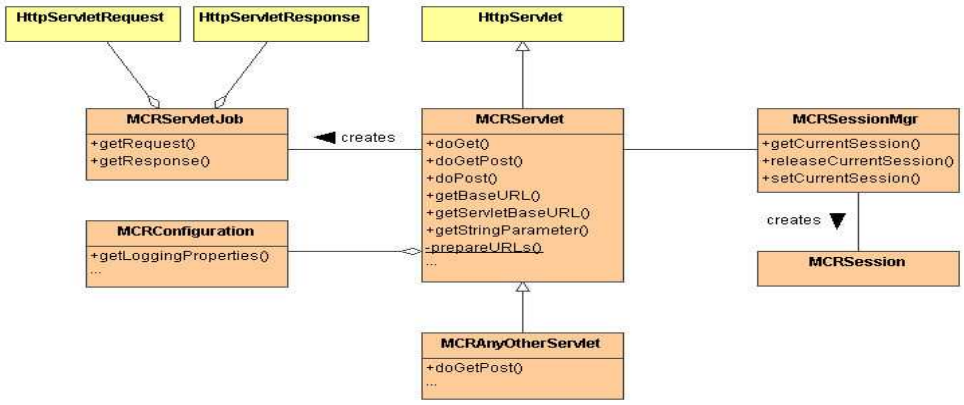


Abbildung 3.4.1: Klassendiagramm Common Servlets

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

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

Der Programmablauf innerhalb von MCRServlet ist im folgenden Sequenzdiagramm (sihe Abbildung) dargestellt. Bei einem eingehenden Request (doGet() oder doPost()) wird zunächst an

7 Die Methode convertData sollten Sie in Ihrer Klasse löschen. Ebenso die Definition in der commands-ArrayList.

MVRServlet.doGetPost() delegiert.[8]

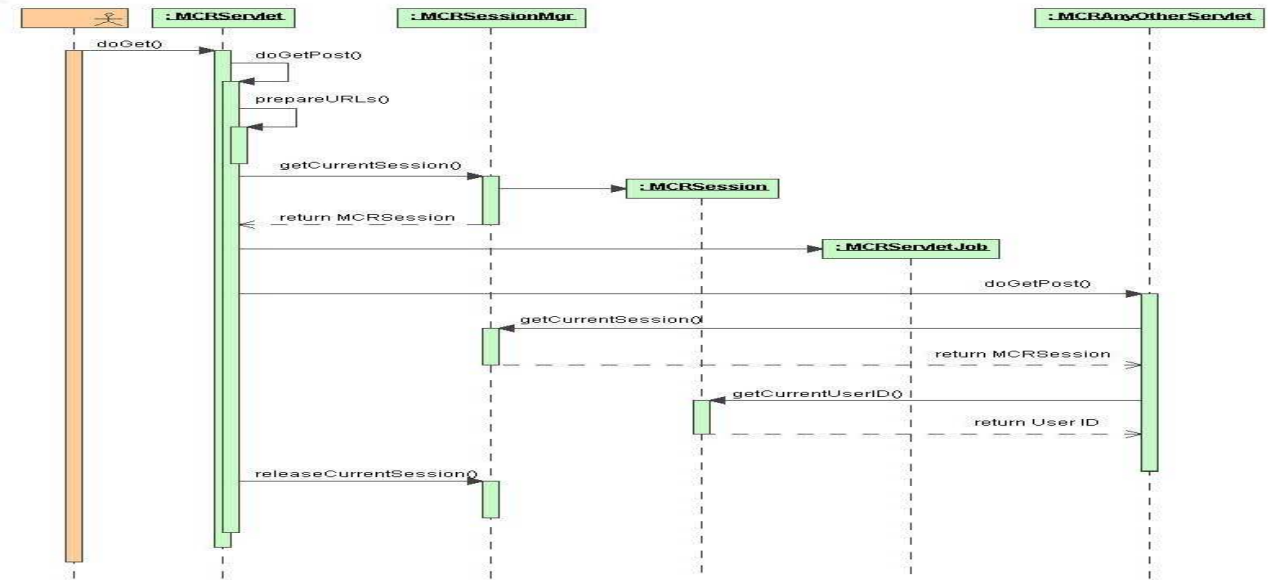


Abbildung 3.4.2: Sequenzdiagramm Common Servlets

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

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

8 Bei dieser Delegation wird ein Parameter mit geführt, über den feststellbar ist, ob es sich um einen GET- oder POST-Request gehandelt hat.

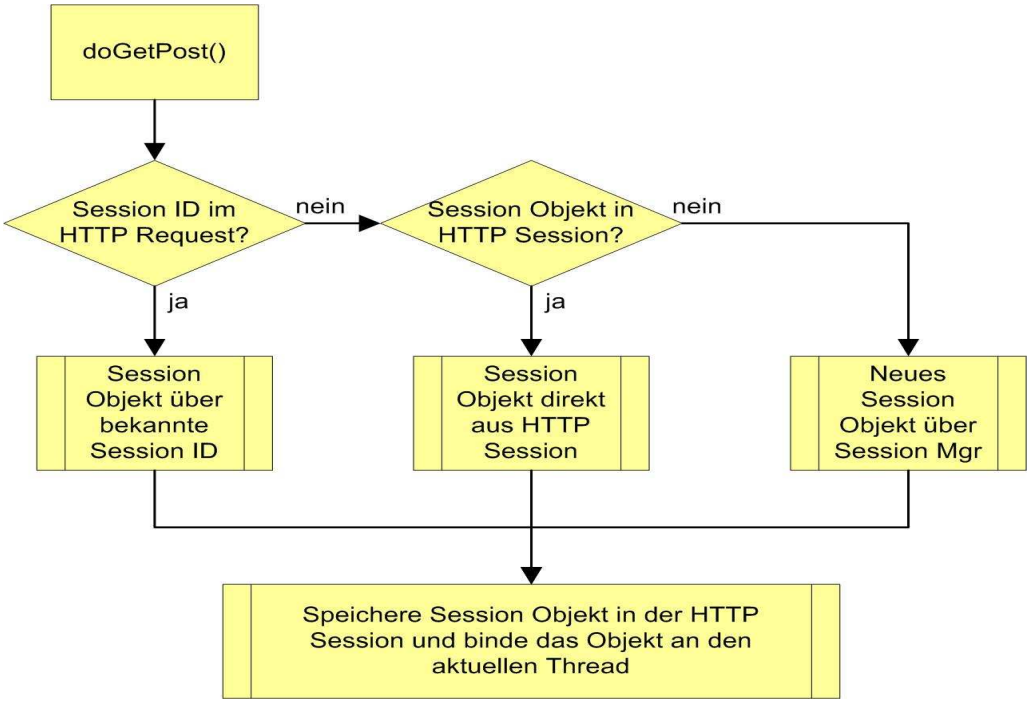


Abbildung 3.4.3: Ablaufdiagramm für MCRServlet.doGetPost()

Im Sequenzdiagramm gehen wird avonaus, dass die Sitzung neu ist und deswegen ein Session-Objekt über MCRSessionMgr.getCurrentSession() erzeugt werden muss. Schliesslich wird eine Instanz von MCRServletJob erzeugt. Diese Klasse ist nichts weiter als ein Container für die aktuellen HttpServletRequest und HttpServletResponse Objekte und hat keine weitere Funktionalität (siehe Klassendiagramm).[9]

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

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

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

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

Es sei bemerkt, dass dies nicht notwendigerweise genau so durch geführt werden muss. Da wegen den geschilderten Problemen mit threadlocal Variablen in Servlet-Umgebungen das Session-Objekt auch in der HttpSession abgelegt sein muss, könnte man die Kontext informationen auch aus der übergebenen Instanz von MCRServletJob gewinnen.

9 Das Speichern des Session-Objekts in der HttpSession ist notwendig, weil in einer typischen Servlet-Engine mit Thread-Pool Umgebung nicht davon ausgegangen werden darf, dass bei aufeinanderfolgenden Anfragen aus dem selben Kontext auch der selbe Thread zugeweisen wird.

3.4.3 Das Login-Servlet und MCRSession

Das LoginServlet, implementiert durch die Klasse MCRLoginServlet, dient zum Anmelden von Benutzern und Benutzerinnen über ein We-Formular. Die Funktionsweise ist wie folgt: Wie in Abschnitt 3.7.2 empfohlen, überschreibt MCRLoginServlet nicht die von MCRServlet geerbten Standard-Methoden doGet() und doPost(). Meldet sich ein Benutzer oder eine Benutzerin über das MCRLoginServlet an, so wird dahe rzunächst die Funktionalität von MCRServlet ausgenutzt und die in Abschnitt3.7.2 beschriebene Verbindung zur Sessionverwaltung herstellt. Wie dort ebenfalls beschrieben, wird der Programmfluss an das Login-Servlet über die Methode MCRLoginServlet.doGetPost() delegiert. Der Ablauf in doGetPost() wird imfolgenden Diagramm dargestellt und ist selbsterklärend.

Der resultierende XML Outpu-Stream muss vom zugehörigen Stylesheet verarbeitet werden und hat die folgende Syntax:

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

Abbildung 3.4.4: XML Output des LoginServlets

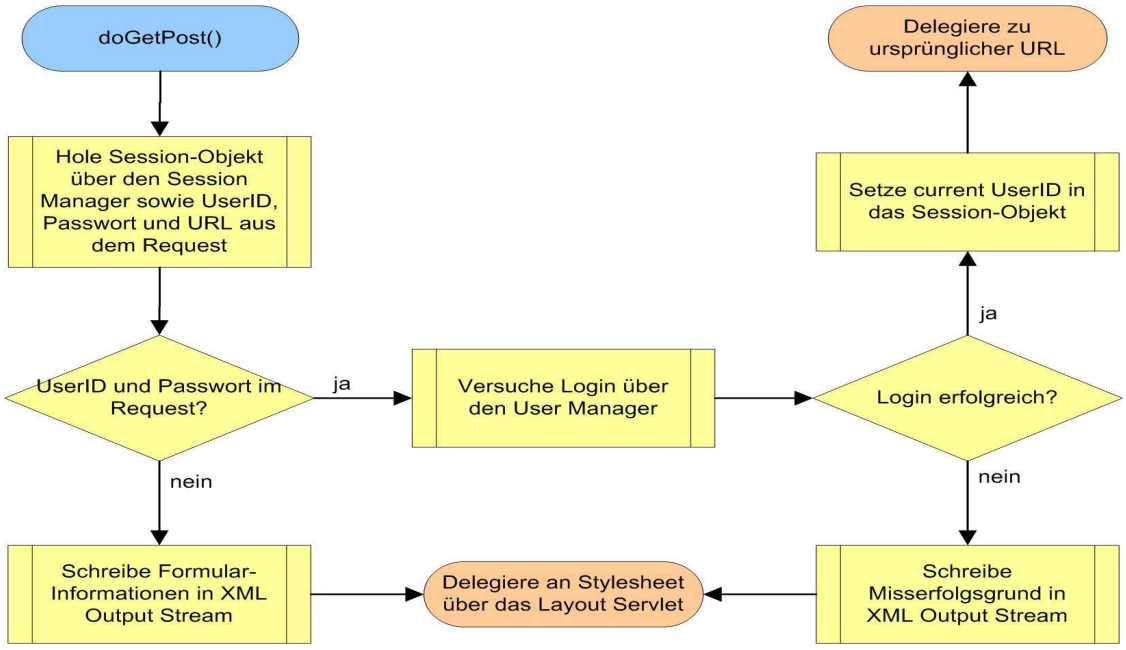


Abbildung 3.4.5: Ablaufdiagramm für MCRLoginServlet.doGetPost()

Bei einer missglückten Anmeldung wird der Grund dafür in Form eines Attributes auf true oder false gesetzt. Das Stylesheet kann dann die entsprechende Meldung ausgeben. Die GastUser-ID und das GastPasswort werden aus einer Konfigurationsdatei gelesen. Die URL schließlich wird dem HttpRequest entnommen und sollte dort von der aufrufenden Seite bzw. vom aufrufenden Servlet

gesetzt sein. Ist sie nicht gesetzt, so wird die Base-URL des MyCoRe-Systems verwendet.