MyCoRe Progammer Guide

Frank Lützenkirchen
Jens Kupferschmidt
Detlev Degenhardt
Johannes Bühler

18. April 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Diese wohl umfangreichste Teil der MyCoRe-Dokumentation beschreibt die Designkriterien und ihre Umsetzung in der vorliegenden Version 0.9 . Mit ihrer Hilfe sollte es Ihnen möglich sein, Details von MyCoRe zu verstehen und eigene Anwendungen bauen zu können.

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. 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).

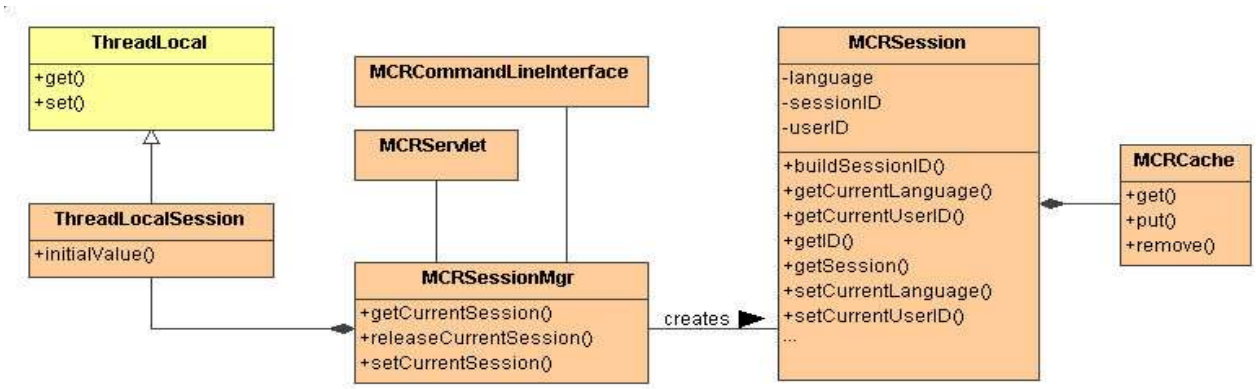

Abbildung 2.4.1: Die Klassen der Sessionverwaltung

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

3.Funktionsprinzipien und Implementierungen

3.1 Das QueryModell von MyCoRe

MyCoRe bemüht sich, das Synatx-Modell der Suchabfragen an die existierenden Standardsdes W3C für XML Path Language (Xpath) Version 1.0 (W3C Recommendation 16. November 1999) anzugleichen, da die derzeit häufigste Abfragesynatx im XML-Bereich ist. Dabei muss aber auch Rücksicht auf die Spezifk 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 Spezifkationen dar. Er genügt aber in der Praxis, um recht komplexe Projekte zu realisieren. Eine Annäherung an Xpath2.0 und Xquery 1.0 wird ersterfolgen, 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 festgelegt und an die entsprechenden Query-Schnittstellen im API 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 ... }
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 Synatx-Form so nicht Bestandteil der W3C Spezifkationen, haben sich aber in der Praxis bewehrt.

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 zusammengehöhren. Alle hier aufgeführten Möglichkeiten sind relativ zu XML-Dokument-Wurzeln. Andere Pfadangaben wie beispielsweise attribute:: für @ sind aus Gründen derKompatibilität 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[3]. 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 beschiebene 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 dieXML-Struktur des Resulates einer Query:

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

Listing: Die Klassen der Sessionverwaltung

- Das Attribut **parsedByLayoutServlet** ist ein Flag, welches eine etwaige Vorverarbeitung des Ergebnisses durch das LayoutServlet anzeigt.

3 Die Typen class und derivate sind reservierte Typen.

- 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** ???
- Das Attribut **hasSucc** ???

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 AbfragenvonKlassi.kationen

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 "..."}]
```