Technische Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr.-Ing. Dr. h. c. Theo Härder

Generische Verwaltung von XML-basierten Daten in relationalen Datenbanksystemen im Rahmen des Projektes META-AKAD

Projektarbeit von

Stephan Fudeus
s_fudeus@informatik.uni-kl.de

Betreuer:

Marcus Flehmig
(AG Datenbanken und Informationssysteme)

Dezember 2003

Inhaltsverzeichnis

1. Einleitung

Die vorliegende Problemstellung ist im Rahmen des Projektes META-AKAD angesiedelt. Ziel ist es, eine Komponente zu entwickeln, die eine feste Aufgabe innerhalb des Projektes löst. Die Aufgabe besteht darin, auf eine möglichst generische Art und Weise verschiedene Sätze von Metadaten über einen zentralen Controller lesen und schreiben zu können. Die Lösung erhebt keinen Anspruch darauf, in anderen Projekten wiederverwendbar zu sein. Dennoch soll ein großer Wert auf einen generischen Ansatz gelegt werden. Zuerst soll allerdings der Dienst META-AKAD an sich vorgestellt werden.

1.1. Projektkontext META-AKAD

Über das Internet werden von Hochschulen, Bildungseinrichtungen und Verlagen im In- und Ausland Lehr- und Lernmaterialien zur Verfügung gestellt. Sie sind entweder kostenfrei für jeden Internetbenutzer zugänglich, sind teilweise aber auch kostenpflichtig. Diese Angebote werden jedoch meist nur von einem begrenzten Personenkreis genutzt. Der Bekanntheitsgrad dieser Produkte beschränkt sich z. B. bei Hochschulangeboten zumeist auf einzelne Lehrstühle der anbietenden Hochschule. Die mangelnde Erschließung dieser elektronischen Dokumente macht das Auffinden geeigneter Produkte für Lehrzwecke schwierig.

Das Projekt META-AKAD hat das Ziel, dem Nutzer einen einheitlichen Zugang zu einem umfassenden Angebot von im Internet verfügbaren Lehr- und Lernmaterial zu verschaffen. Zu den Nutzern des Angebots zählen dabei sowohl Lehrende, die das Material im Rahmen ihrer Lehrveranstaltungen einsetzen können, als auch Lernende, für deren Selbststudium das Material zur Verfügung gestellt wird. Das Projekt schafft einen innovativen Pilot-Service durch den Einsatz und die Entwicklung geeigneter technischer Mittel und den Aufbau funktionierender Organisationsstrukturen, um den effizienten Einsatz der elektronischen Materialien für Lehrzwecke und einen schnellen, möglichst umfassenden und bedarfsgerechten Zugriff auf die im Internet verfügbaren Dokumente zu ermöglichen. Dieser neue Nutzer-Service wird exemplarisch für die Fächer Mathematik, Physik, Germanistik, Psychologie und Biologie aufgebaut, soll aber so ausgelegt werden, dass er auf alle Disziplinen erweitert werden kann.

Zur Realisierung dieses Dienstes soll Online-Lehr- bzw. Lernmaterial auf kooperativer Basis gesammelt, durch standardisierte und materialspezifische Meta-Daten erschlossen, nach inhaltlichen und didaktischen Kriterien bewertet und in einer einheitlichen Nutzeroberfläche zugänglich gemacht werden. Dies beinhaltet die Erschließung mit Hilfe von Verfahren für die (semi-) automatische Klassifikation und Vergabe von Metadaten. Zur Verbesserung der Qualität der Erschließung werden Verfahren zur intellektuellen Nachbearbeitung integriert. Das Sammeln der Metadaten zu den Dokumenten soll sowohl automatisch erfolgen, als auch auf kooperativer Basis manuell nachgearbeitet und ergänzt werden. Schließlich sollen die Materialien von Fachleuten bewertet werden. Durch die Integration einer solchen Qualitätskontrolle wird die Verwertbarkeit der Materialien für Lehr- und Lernzwecke deutlich erhöht und damit ein wesentlicher Mehrwert des Dienstes geschaffen, da der Nutzer in META-AKAD nicht nur recherchieren kann, welche Dokumente verfügbar sind, sondern auch detaillierte Informationen über deren Qualität und Nutzbarkeit erhält. Der Zugang zu den Materialien wird

durch ein benutzerfreundliches Interface ermöglicht, das auf der Grundlage von Nutzerstudien evaluiert und laufend optimiert wird.

Das Projekt ordnet sich in derzeit laufende Überlegungen ein, durch die Nutzung des Internet die Lehre an den Hochschulen effektiver zu gestalten. Das am weitesten fortgeschrittene Projekt ist die "Virtuelle Hochschule Bayern". Im Gegensatz zu solchen bereits bestehenden Projekten ist in META-AKAD jedoch keine eigene Produktion von Lehrmaterialien geplant. [Met02]

In der vorliegenden Arbeit soll eine Teilaufgabe des Projektes META-AKAD näher beleuchtet und eine konkrete Implementierung beschrieben werden.

1.2. Problembeschreibung

Eingebettet in den Kontext des Projektes META-AKAD der Universitäten Regensburg und Kaiserslautern werden folgende Anforderungen an eine generische Komponente zur Verwaltung von XML-Daten in ein relationales Datenbanksystem gestellt:

1. Die Komponente soll sowohl das *Einfügen*, das *Aktualisieren* als auch das *Löschen* von Daten ermöglichen. Dazu soll eine einheitliche Schnittstelle bereitgestellt werden, die möglichst transparent bezüglich unterschiedlicher Dokumenttypen sein soll.

2. Zu verarbeitende Dokumente können drei unterschiedliche Typen haben:

   a) Lehr-/Lernmaterialien (*learningresource*)

   b) Gutachten (*peerReview*)

   c) Benutzerkommentare (*userComment*)

   Alle drei Typen sollen möglichst mit gemeinsamen generischen Komponenten arbeiten. Das Modell von META-AKAD sieht vor, dass ein *learningresource*-Objekt die beschreibenden (Meta-) Daten für jeweils ein Lehr-/Lernmaterial enthält. Jedes dieser Dokumente kann/soll von einem unabhängigen Gutachter bewertet werden; Gutachten sind also mit Lehr-/Lernmaterialien verknüpft. Ebenso kann jeder Benutzer des Systems zu einem Dokument bzw. zu den zugehörigen Metadaten Kommentare abgeben, die ebenso der entsprechenden *learningresource* zugeordnet werden müssen.

3. Diese Dokumente unterschiedlicher Typen sollen nicht nur einzeln, sondern auch als zusammengehörige Einheit dem System hinzugefügt werden können. Umgekehrt bedeutet dies, das ein dem System übermitteltes Dokument mehrere Einträge enthalten kann, welche einzeln verarbeitet werden müssen.

4. Das ganze Konzept soll *generisch* sein, d. h. alle Teile der Realisierung sollen möglichst robust gegenüber Änderungen sein von:

   a) XML-Schema

   b) Datenbank-Schema

   c) Datentypen

1.3. Beispiel eines XML-Dokumentes

Um die folgenden Ausführungen besser einordnen zu können ist hier ein kurzes beispielhaftes XML-Dokument, welches Autor, Titel und URL eines Lehr-/Lerndokumentes beschreibt.

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<metabase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.akleon.de/xsd/metabase.xsd">

    <learningresource guID="TEST1">
        <title>
            <main lang="DE" origin="intellectual">Testdokument</main>
        </title>
        <creator>
            <name origin="intellectual">Fudeus, Stephan</name>
        </creator>
        <identifier origin="automatic" URI="http:/www.akleon.de/test.html" />
    </learningresource>
</metabase>
```

2. Ansatz

2.1. Denkbare Vorgehensweise

Prinzipiell sind bei der Realisierung der Speicherung von XML-Daten in einer relationalen Datenbank zwei Vorgehensweisen möglich:

1. Speichern des kompletten Fragmentes am Stück als Text.
2. Speichern aller Elemente und Attribute des Fragmentes einzeln in der Datenbank, ohne dabei jedoch die Struktur zu verlieren.

Der erste Fall ist sehr einfach zu realisieren. Allerdings ist es sehr ineffizient, auf langen Strings in der Datenbank zu suchen. Da im Projekt META-AKAD eine Suchoberfläche für Lehr-/Lernmaterialien realisiert werden soll, scheidet diese Variante aus.

Die zweite Variante ist sehr effizient für Suchanfragen, allerdings ist der Aufwand sehr hoch, um sowohl die Daten als auch die Originalstruktur zu erhalten. Die Originalstruktur darf jedoch nicht vernachlässigt werden, da in XML-Daten die Reihenfolge von Elementen ebenfalls Information trägt, was aufwändig modelliert und realisiert werden muss, da die Basis ein relationales Datenbanksystem ist.

Beide Varianten werden im Projekt benötigt, da zum einen für die Lieferung der kompletten Metadaten eine vollständige Rekonstruktion nötig ist. Diese alleine aus den einzelnen Elementen inklusive Reihenfolgeerhaltung zu rekonstruieren erfordert eine durchgehend vollständige Abbildung sowie eine hohe Rechenleistung bei Anforderung des Dokumentes. Für die Suche ist allerdings die Zerstückelung des ursprünglichen Dokumentes nötig, da ansonsten Suchanfragen deutlich zu langsam wären.

Anforderung an die Integrationskomponente ist von daher, beide Varianten zu kombinieren, damit sowohl effizient gesucht werden kann als auch die Daten im Original erhalten bleiben. Es soll jedoch trotzdem darauf geachtet werden, die vorhandenen Daten und Strukturinformationen so vollständig wie möglich auf das relationale Datenbankmodell abzubilden.

2.2. Realisierungsmöglichkeiten

Für die Umsetzung der genannten Anforderungen an die Verwaltungskomponente gibt es im Kern zwei unterschiedliche Vorgehensweisen auf verschiedenen Abstraktionsebenen.

- Größtmögliche Kontrolle über den Analysevorgang eines XML-Dokumentes bietet die Möglichkeit, die SAX-Schnittstelle zu nutzen. SAX (Simple API for XML) [Sax00] ist als am weitesten verbreitete API für XML ein de-facto-Standard. SAX stützt sich auf ein eventbasiertes Modell. Für jeden öffnenden oder schließenden XML-Tag sowie für Text zwischen zwei XML-Tags wird ein Event an den sogenannten Content-Handler geschickt. Dieser Content-Handler ist die eigentliche Komponente, die Anweisungen für die Verarbeitung von Elementen des XML-Strings enthält. Es gibt zahlreiche Parser, die die SAX-Schnittstelle unterstützen. Dafür muss der Programmierer selber dafür Sorge tragen, dass der Kontext des

XML-Elementes, d. h. Eltern- und Kindknoten sowie die Elementreihenfolge, gewahrt bleibt, falls diese Informationen für ihn von Belang sind.

- Alternativ besteht die Möglichkeit, das XML-Dokument mittels DOM (Document Object Model [WD02]) in eine Objektstruktur (einen Baum) zu überführen. Mit dem DOM-Framework lässt sich so ein Objekt-Baum durchlaufen, Knoten hinzufügen oder auch löschen. Hierfür gibt es auch nicht nur eine spezifizierte Schnittstelle, sondern auch frei nutzbare Implementierungen und Werkzeuge wie *Castor* [Exo03].

Beide Möglichkeiten lassen sich in der Aufgabe parallel für verschiedene Teilaufgaben verwenden, da sich die Manipulation und Speicherung der XML-Dokumente in Teilaufgaben aufteilen lässt.

3. Konzept

Zum Zeitpunkt der Erstellung dieser Komponente war die Systemarchitektur bereits durch die übrigen Komponenten des Projektes META-AKAD festgelegt. An dieser Vorgabe musste auch die Komponente zur generischen Verwaltung der XML-basierten Daten ausgerichtet werden.

3.1. Java 2 Enterprise Edition

Die Anforderungen an den Dienst führen zu einem System mit einer sehr komplexen Struktur. Die Systemarchitektur muss diese Anforderungen berücksichtigen und zugleich für zukünftige Entwicklungen und Erweiterungen vorbereitet sein. Um dies zu gewährleisten wurde Java2 Enterprise Edition (J2EE) von Sun Microsystems als Plattform gewählt (siehe [SUN03b]). Diese bietet einen herstellerunabhängigen Standard für verteilte, Java-basierte Anwendungen. Durch die Bereitstellung von grundlegenden Diensten (z. B. Namens- und Verzeichnisdiensten (JNDI) oder Transaktionsverwaltung (JTA)) und die Unterstützung von allgemeinen Konzepten (wie beispielsweise Client/-Server-Architektur oder Persistenz von Objekten) ermöglicht sie die Entwicklung von mehrschichtigen, auf standardisierten Komponenten basierenden Systemen. Auf Techniken und Konzepte der Java2 Standard Edition (J2SE) (siehe [SUN03c]). aufbauend, unterstützt J2EE auch Datenbankzugriff mit JDBC, Enterprise-JavaBeans (EJB), Java-Server-Pages (JSP), die Java-Servlet-Schnittstelle, XML und CORBA, was die Entwicklung von unternehmensweiten Anwendungen ermöglicht.

Das J2EE-Modell für unternehmensweite Anwendungen definiert eine Dreiteilung des Systems. Diese Aufsplitterung beruht auf der getrennten Betrachtung (Realisierung) von Präsentations-, Anwendungs- und Datenhaltungsaspekten in unterschiedlichen Schichten.

Web-Tier Die Präsentationslogik (Web-Tier) ist nochmals unterteilt in eine clientseitige und eine serverseitige Präsentationsschicht. Die Serverseite umfasst dabei den Web-Server, der eine entsprechende Infrastruktur (Web-Container) zur Ausführung der JSP-Seiten und Servlets zur Verfügung stellt (siehe [SUN03e]). Auf der Clientseite sind verschiedene Komponenten, wie Web-Browser, Applets oder auch eigenständige Java-Programme berücksichtigt, die auf Dienste der Serverschicht zugreifen können.

EJB-Tier In der Anwendungsschicht (EJB-Tier) findet man die die entsprechende Umgebung (EJBContainer) für die Ausführung von EJB-Komponenten (siehe [SUN03a]). EJBs sind wiederverwendbare Softwarekomponenten, die unabhängig voneinander erzeugt und gepflegt werden können. Ferner sind sie mit standardisierten Schnittstellen ausgestattet, um mit anderen Komponenten zusammenzuarbeiten. Die Infrastruktur dafür wird vom EJB-Container zur Verfügung gestellt. Es gibt drei grundlegende Arten von EJBs. Entity-EJB, Session-EJB und Message-Driven-Beans. Während Session-EJB das Verhalten und die Geschäftslogik realisieren, repräsentieren Entity-Beans persistente Geschäftsobjekte, wie zum Beispiel ein einzelnes Tupel einer Datenbank. Message-Driven-Beans sind für die vorliegende Aufgabe nicht weiter relevant. Der EJB-

Container und der Web-Container bilden zusammen mit weiteren Diensten den Java-Application-Server.

Eis-Tier In der dritten Ebene, der Datenhaltungsschicht (EIS-Tier) findet man Datenbanksysteme und so genannte Enterprise-Information-Systeme. Die einzelnen Schichten und Komponenten können größtenteils getrennt voneinander betrachtet und realisiert werden, was vor allem für Realisierung von großen Projekten von Vorteil ist.

3.2. Konzept des Projekts META-AKAD

Konkret im Projekt ist nicht nur die vertikale Dreiteilung durch die J2EE-Architektur gegeben, sondern auch eine horizontale Dreiteilung. Die Aufgaben im EJB-Tier werden in drei Säulen aufgeteilt (Anfrageverarbeitung, Ergebnisverarbeitung und XML-Dokumenten-Management) (siehe Abbildung 3.2). Die vorliegende Arbeit beschäftigt sich mit der Implementierung der dritten Säule (XML-Dokumenten-Management).

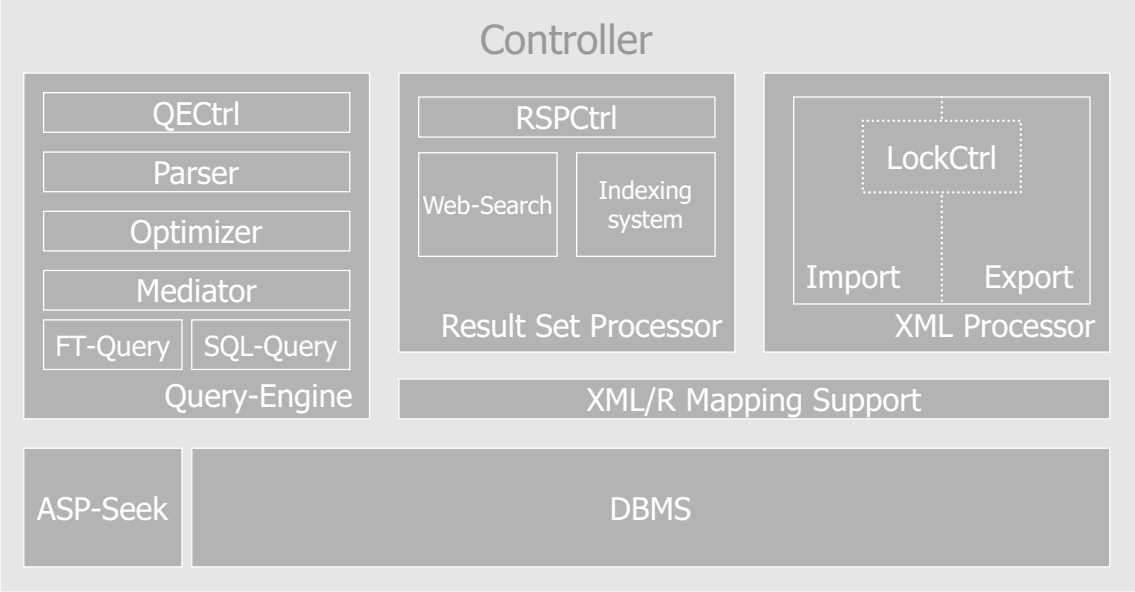

Abbildung 1: 3-Säulen-Architektur von META-AKAD

3.3. Konzept der Verwaltungskomponente

Die zu erstellende Komponente zur generischen Verwaltung der XML-basierten Daten (im weiteren XMLProcessor genannt) soll aus mehreren Schichten aufgebaut sein. So gibt es eine Instanz, die eine zentrale Schnittstelle für alle in der Hierarchie höher liegenden Anwendungen dienen soll. Hier werden vorbereitende Maßnahmen getroffen, wie das Umschreiben von Identifiern, das Aufteilen der XML-Dokumente in Fragmente mit nur jeweils einem einzelnen Dokument.

Dieser zentrale Controller verteilt dann die Arbeit an hierarchisch tiefer liegende typspezifische Controller, welche für ihren Dokumenttyp zugeschnitten sind.

Diese einzelnen Controller steuern das Zerlegen des XML-Fragmentes in seine einzelnen Informationen, übernehmen das Speichern der Einzeldaten in den Suchstrukturen der Datenhaltungskomponente und tragen dafür Sorge, dass auch das komplette Fragment mit allen Verwaltungsgattributen zum schnellen Ausliefern gespeichert wird.

Darunter liegen noch spezifisch für jeden Dokumenttyp Schnittstellen zur Datenbank, welche durch die J2EE-Systemumgebung bereitgestellt werden können, mittels derer die vollständigen Fragmente gespeichert werden.

3.4. Behandlung von Abhängigkeiten

Zwischen den verschiedenen Objekttypen im Projekt META-AKAD bestehen unterschiedliche Abhängigkeiten. So besitzen *usercomments* und *peerreviews* immer eine Referenz auf die *learningresource*, die sie beschreiben bzw. beurteilen. Zusätzlich dazu gibt es Tabellen, in denen zusätzliche Informationen zu den gesammelten Daten erfasst sind.

Es existiert eine Tabelle für einen optimierten Suchindex (*search_index*). In diesem Index werden ausgewählte Attribute von *learningresources* in einzelnen Wortbausteinen erfasst und bieten so die Möglichkeit, schnell Referenzen auf "passende" *learningresources* finden zu können.

Außerdem wird für jede Klassifikation nach RVK (siehe [RVK]) festgehalten, wie viele Dokumente mit dieser klassifiziert wurden (*docfreq*). Diese Hilfstabelle dient dazu, dem Endnutzer schnell anzeigen zu können, wie viele Dokumente mit einer bestimmten Klassifikation in der Datenbank enthalten sind.

Da nur durch Änderungen an der Gesamtmenge der im System vorhandenen Dokumente eine Änderung des optimierten Suchindex und der Zähler für Klassifikationen nach RVK nötig sind und sämtliche Änderungen an der Menge der Dokumente durch die zu realisierende Komponente erfolgt, ist es sinnvoll, solche Anpassungen ebenfalls von dieser Komponente ausführen zu lassen. Hierzu müssen jeweils im Anschluss an eine Änderungsoperation entsprechende Maßnahmen ergriffen werden.

3.5. Interaktion mit der Datenbank

Bereits in der Konzeption soll darauf geachtet werden, dass ein unnötiger Overhead durch übermäßige Kommunikation mit der Datenbank vermieden wird. Deshalb soll beim späteren Eintragen eines Dokumentes in die Datenbank nicht jedes Attribut einzeln, sondern möglichst komplette Tupel (ganze Tabellenzeilen) auf einmal in die Datenbank geschrieben werden. Hierzu ist der einer passenden Cache-Struktur notwendig. In dieser Struktur sollen möglichst viele Daten gesammelt werden bevor diese am Ende des Import-Vorgangs in die Datenbank transferiert werden.

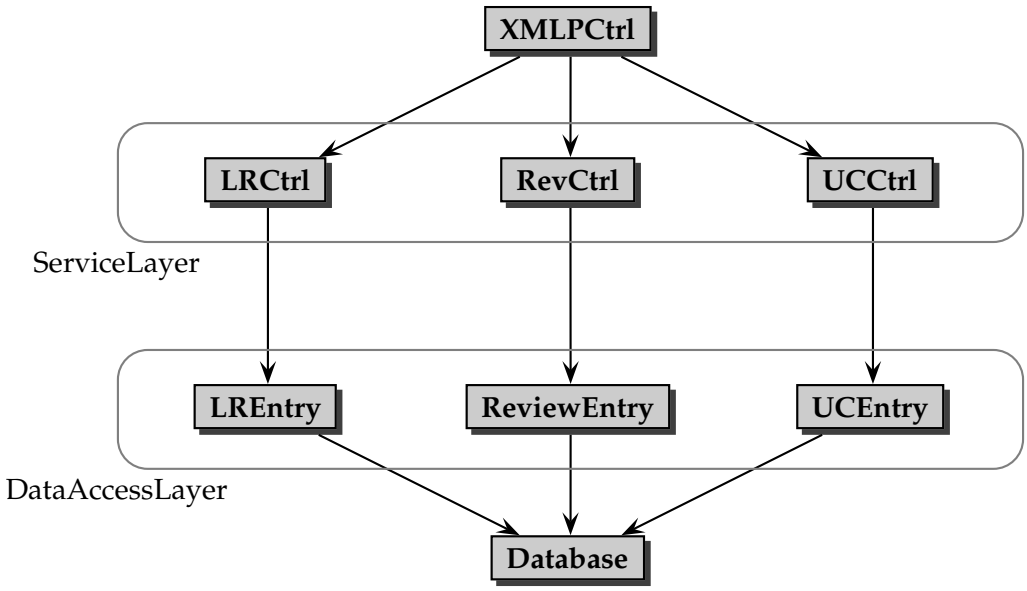

Abbildung 2: Grobkonzept

4. Realisierung

Der im Rahmen der Projektarbeit entworfene XMLProcessor verfolgt ein hierarchisches Modell. Alle Anfragen von Clientseite sollen direkt an den XMLPCTRL gestellt werden. Nach ersten erforderlichen Schritten werden Anfragen an die untergeordneten Strukturen (LRCTRL, REVCTRL, UCCTRL) weitergeleitet. Der Aufbau ist durch das Klassendiagramm in Abbildung 2 dargelegt. Die gesamte Realisierung erfolgte im Rahmen einer J2EE-Umgebung (siehe Abschnitt 3.1).

4.1. Verfügbare Datenstrukturen

4.1.1. Datenbank-Tabellen

Für jeden der drei Dokumenttypen existieren primär zwei Tabellen, in denen alle Informationen (Suchstrukturen, Verwaltungsdaten, etc.) festgehalten werden. Dies sind die Tabellen *learningresource, review, usercomment, lrmaintain, reviewmaintain* und *ucmaintain*. In den *maintain*-Tabellen werden die Verwaltungsinformationen wie Benutzerkennung des Erstellers, Erstelldatum, Änderungsdatum, Veröffentlichungsstatus, etc. sowie auch die vollständigen XML-Fragmente festgehalten. In der jeweils anderen Tabelle (im Fall von *learningresource* noch in zahlreichen weiteren durch Fremdschlüsselbeziehungen verknüpften Tabellen) werden die Suchstrukturen gespeichert.

Zugriff auf die *Maintain*-Tabellen erfolgt hauptsächlich über zugehörige Entity-Beans (siehe Abschnitt 4.2.3).

4.1.2. Metamodell

Um die Generizität des Systems zu gewährleisten, wurde ein Metamodell verwendet, welches auch schon bei der Generierung von Suchanfragen benutzt wird. Das Metamodell enthält Informationen darüber, welche Tabellen mit welchen Attributen vorhanden sind, welche Beziehungen verschiedene Attribute verknüpfen, welche XML-Elemente auf welche Datenbankstrukturen abgebildet werden und welche Abhängigkeiten dort bestehen. Details zum Metamodell sind in Abschnitt 4.3 aufgeführt.

4.1.3. ConfigurationStore

Der CONFIGURATIONSTORE ist eine Datenbankstruktur, die dazu da ist, systemspezifische (z. B. datenbankspezifische) Parameter oder Anfragen bereitzustellen, damit diese nicht fest in den Code eingebaut werden müssen. So muss bei einer Änderung von Systemparametern oder der zugrunde liegenden Datenbank idealerweise nur der CONFIGURATIONSTORE, nicht aber der Code abgeändert werden.

Nichtsdestotrotz gibt es noch immer einige Stellen im Code, die so spezifisch und umfangreich sind, dass eine Auslagerung in den CONFIGURATIONSTORE nicht möglich ist. An stark rekursiven Stellen im Code ist auch die Effizienz ein Problem. So ist es deutlich effizienter, einen konstanten Anfrage-String im Code unterzubringen, statt bei jeder Rekursionsstufe eine Datenbankanfrage an den CONFIGURATIONSTORE zu stellen.

Auch für den CONFIGURATIONSTORE ist ein Entity-Bean implementiert, so dass Manipulationen bzw. Selektionen ohne konkreten JDBC-Aufruf ausgeführt werden können.

4.1.4. XMLDocument

Das XMLDOCUMENT ist eine Datenstruktur, die ein XML-Dokument (als String repräsentiert) sowie eine Menge von Eigenschaften dieses Dokumentes (*Properties*) zusammenfasst. Es ist diese Struktur, die bei Einfüge- oder Aktualisierungsoperationen zwischen den Komponenten ausgetauscht wird, um beliebig viele Daten, welche den XML-String beeinflussen oder beschreiben, beifügen zu können. Der Aufbau eines XMLDOCUMENT ist in Abbildung 3 dargestellt.

4.2. Realisierte Komponenten

Im Rahmen der Projektarbeit wurden folgende Arten von Komponenten realisiert. Hier erfolgt ein grober Überblick über ihre Funktionalität. Einige spezielle Komponenten werden später in Abschnitt 4.5 genauer erläutert.

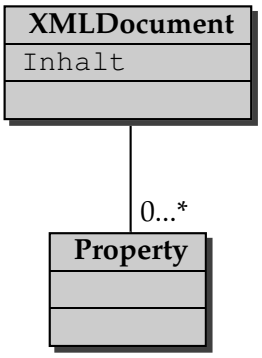

Abbildung 3: Aufbau XMLDocument

4.2.1. XMLPCtrl

Für die "Außenwelt" wird die gesamte Funktionalität (d. h. Einfügen, Aktualisieren, Löschen und Verknüpfen von Dokumenten) des XMLProcessors durch die Schnittstelle XMLPCTRL erbracht. Je nach Typ der zu bearbeitenden Ressource wird der Methodenaufruf an den entsprechenden Typ-Controller (s.u.) weitergeleitet. Der XMLPCTRL stellt neben den spezifischen Methoden *addLearningresource()*, *addPeerreview()* und *addUsercomment()* auch eine Methode *insertDocument()* zur Verfügung. Hier kann ein XML-String beliebigen Typs, insbesondere XML-Fragmente verschiedener Typen, gemischt übergeben werden. Der in diesem Schritt verwendete Parser (siehe auch Abschnitt 4.2.4) trennt die Fragmente sortenrein und vergibt über den IDGENERATOR (siehe Abschnitt 4.2.5) neue globale Identifier. Der XMLPCTRL folgt dem SessionFacade-Pattern [SUN02].

4.2.2. Typ-spezifische Controller

Für jeden Typ (d. h. *learningresource, peerreview, usercomment*) wurde ein eigener Controller implementiert, der die spezifischen Aktionen für den entsprechenden Typ umsetzt. Für *learningresource* ist der LRCTRL, für *peerreview* der REVCTRL und für *usercomment* der UCCTRL zuständig.

Jede dieser Komponenten wurde als *Stateless Session-Bean* realisiert, da von einem zum anderen Einfügevorgang keinerlei Zustandsinformationen gehalten werden müssen; alle nötigen Informationen (z. B. Benutzername) können aus dem aktuellen Aufruf-Kontext abgeleitet werden.

In Abschnitt 4.5.2 wird LRCTRL exemplarisch näher betrachtet.

4.2.3. Entry-Beans

Jeder der drei genannten Typ-Controller verfügt zum Manipulieren der Verwaltungsstrukturen (*maintain-Tabellen*) über je ein Entity-Bean (LRENTRY, REVIEWENTRY, UCENTRY), um unterstützt durch den J2EE-Containers Zugriff auf die Verwaltungsattribute zu haben. Die Container-Unterstützung wird an dieser Stelle genutzt, damit die

Transaktionsverwaltung und die Sicherung der Persistenz der Daten nicht durch den Programmierer sichergestellt werden muss. Ein einzelnes Entity-Bean verkörpert innerhalb des Containers praktisch ein Tupel der zugehörigen Datenbank-Tabelle und kann über spezielle *finder-Methoden* angefragt werden.

Diese Entity-Beans sind eigene gekapselte Komponenten, die theoretisch von jedem Objekt über den *Naming-Service (JNDI)* [SUN03d] aufgelöst werden könnten. So greifen im Rahmen des Imports von *peerreview* und *usercomment* REVCTRL und UCCTRL auch auf LRENTRY-Beans zu, um Referenzen auf die zugehörige *learningresource* zu knüpfen (*belongs_to*). Entity-Beans von *usercomment* und *peerreview* sind durch *Container-Managed-Relationship (CMR)* mit ihrer *learningresource* verknüpft. Dies führt dazu, dass man bei Abfrage des CMR-verwalteten Attributes nicht den Schlüssel des Zielobjektes sondern direkt das zugehörige Objekt (hier: die zugehörige *learningresource*) erhält.

4.2.4. JavaBeans

Unabhängig von den Enterprise-JavaBeans gibt es auch noch JavaBeans – im Prinzip einfache Java-Klassen, die innerhalb des Containers im Kontext des aufrufenden Enterprise-Beans laufen. Damit lassen sich beispielsweise simple Arbeitsklassen aus einem Enterprise-Bean auslagern, welches dann nur noch für die Ablaufsteuerung zuständig ist.

FragmentParser Der FRAGMENTPARSER ist als ein solches JavaBean realisiert. Er wird von allen drei Typ-Controllern als *content- und error-handler* für den SAX-Parser verwendet. Dieser Parser ist dafür verantwortlich, dass alle Informationen der XML-Fragmente in die Suchdatenstrukturen eingetragen werden. Eine detaillierte Beschreibung des FRAGMENTPARSER findet sich in Abschnitt 4.5.1.

ParserContentHandler Dies ist ein weiteres JavaBean, welches das Interface von *content- und error-handler* implementiert. Es wird vom SAX-Parser im ersten Importschritt verwendet, um den erhaltenen XML-String in typspezifische Fragmente zu zerlegen. Dies ist auch die Instanz, die prüft, ob der erhaltene XML-String dem XML-Schema [WX01] für Dokumente des Typs *metabase* entspricht. Das XML-Schema definiert den korrekten Aufbau eines XML-Dokumentes. Sollte diese Validitätsprüfung fehlschlagen, so wird das Dokument verworfen und eine Exception an den aufrufenden Client übermittelt.

Eine weitere wichtige Aufgabe des PARSERCONTENTHANDLER ist der Aufruf des IDGENERATOR (siehe Abschnitt 4.2.5). Da man sich prinzipiell nicht darauf verlassen kann, dass die vom Client vergebenen Identifier für die übertragenen XML-Fragmente eindeutig und noch nicht verwendet sind, werden hier alle Identifier systemseitig neu vergeben. Besonders zu beachten ist hier, dass vom Client vergebene Ids unter Umständen innerhalb der Fragmente wieder referenziert werden (im Rahmen von Relationen wie bspw. *hasPart* oder *hasUsercomment*). Dieses Problem wird durch eine Mapping-Tabelle gelöst, die bereits bekannte Ids immer in die gleiche Id übersetzt. Voraussetzung dafür ist natürlich, dass der Client seine Referenzen innerhalb der Fragmente konsistent gesetzt hat.

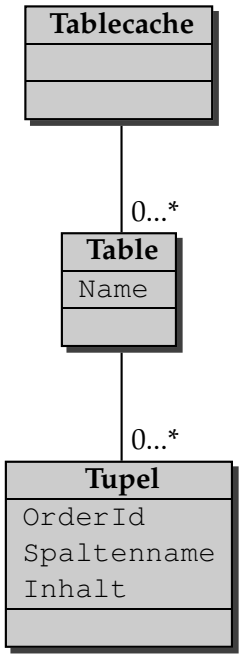

Abbildung 4: Aufbau Tablecache

Tablecache Dies ist ein JavaBean, das eine Datenstruktur repräsentiert, um Inhalte einer Tabelle zwischenzuspeichern. Im Detail wird es vom FRAGMENTPARSER verwendet, um Änderungsoptionen an der Datenbank festzuhalten, damit diese am Ende des Parse-Vorgangs in die Datenbank geschrieben werden. Der Haupteffekt ist dabei, dass ein Tupel erst dann geschrieben wird, wenn alle zu schreibenden Einträge des Tupels vorliegen. Andernfalls müsste auf ein `INSERT` noch ein späteres `UPDATE` folgen. Noch aufwändiger ist überhaupt die Entscheidung, ob erst ein `INSERT` oder bereits ein `UPDATE` erfolgen muss. Der logische Aufbau eines Tablecache wird in Abbildung 4 beschrieben.

4.2.5. IdGenerator

Beim Import von Dokumenten ist es nötig, eindeutige Identifier zu vergeben. Hierzu dient der IDGENERATOR, der als *Stateless Session-Bean* realisiert ist. Er bietet vier spezielle Methoden, um neue Identifier für vier verschiedene Counter (vorgesehen für *learningresource, usercomment, peerreview, creator*) zu erhalten.

Der IDGENERATOR speichert seine Zählvariablen hierbei im CONFIGURATIONSTORE. Wichtig bei der Generierung neuer Identifier ist die Atomarität dieser Aktion. Darum kann der Zugriff auf den CONFIGURATIONSTORE hier auch nicht über das zugehörige Entity-Bean erfolgen, sondern der IDGENERATOR muss direkt auf die Datenbank zugreifen. Um die Atomarität zu gewährleisten, muss zuerst in die Datenbank geschrieben werden, um das Tupel vor Lesezugriff anderer Transaktionen zu schützen. Danach darf der neue Wert dann ausgelesen werden. Dies lässt sich nur mit einer direkten SQL-Anfrage erreichen, z. B.:

```
UPDATE configurationstore
SET value=value+1
WHERE countername='lrid'
```

SQL-Fragment 1: direkte SQL-Anfrage zum Inkrementieren des Counters

Alternativ dazu ließe sich die Isolationsstufe der Entity-EJBs erhöhen, d. h. man könnte das System so konfigurieren, dass lange Lesesperren vergeben werden, und so bereits das Lesen eines Tupels bis zum Abschluss der Gesamt-Transaktion eine Sperre für andere Transaktionen bewirkt. Dies ist jedoch ein starker Eingriff in die Parallelität des Systems und würde unnötig viele parallel mögliche Zugriffe auf die Datenbank blockieren. Hierauf wird also im verwendeten Ansatz verzichtet.

4.3. Metamodell

Das Metamodell ist eines der wichtigsten Konstrukte, um eine generische Import-Komponente zu realisieren. Wie bereits erwähnt enthält das Metamodell alle Informationen, um aus einem gegebenen XML-Pfad den Speicherort des zugehörigen Inhalts in der Datenbank herauszufinden (Tabelle und Spalte). Desweiteren liefert das Metamodell Informationen dazu, welchen XML-Datentyp ein gegebenes Element hat (Attribut, einfach vorkommendes Element, mehrfach vorkommendes Element) und was der resultierende Java-Datentyp ist.

Im Metamodell sind aber nicht nur direkte Eigenschaften des Elementes mit einem gegebenen XML-Pfad festgehalten, sondern auch abhängige Elemente, jeweils bis herunter auf die Ebene einer *learningresource*. Damit lässt sich immer herausfinden, zu welchem Lerndokument ein Attribut genau gehört. In den im Metamodell textuell beschriebenen Abhängigkeiten lässt sich auslesen, welche Tabellen und welche Spalten dort miteinander verknüpft sind, also über welche Tabellen später ein `JOIN` gebildet werden muss und welche Fremdschlüsselbeziehungen gelten.

Das Metamodell ist nicht nur für Lerndokumente, sondern auch für Gutachten und Benutzerkommentare definiert, auch wenn in diesen beiden Typen deutlicher weniger Elemente verwaltet werden müssen.

Der Zugriff auf das Metamodell erfolgt ausschließlich über zwei Entity-Beans (METAMODELOBJECT, DEPENDENCYOBJECT).

4.4. Ablauf eines Imports

Der aufwändigste Teil des XMLProcessors ist das Einfügen von Dokumenten. In diesem Abschnitt soll kurz exemplarisch der Ablauf eines Imports von Lerndokumenten beschrieben werden.

Initialisiert wird die Bearbeitung über den Aufruf *addLearningresource()* mit einem XMLDOCUMENT als Parameter. In einem ersten Schritt wird das zu verarbeitende XML-Dokument durch den ersten Parser (siehe PARSERCONTENTHANDLER in Abschnitt 4.2.4) in seine Einzelteile zerlegt, d. h. in Lerndokumente, Gutachten und Benutzerkommentare. Dort werden auch für alle Dokumente neue globale Identifier vergeben. An dieser

Stelle nicht benötigte Fragmente wie Gutachten und Benutzerkommentare werden verworfen, da durch den konkreten Methodenaufruf *addLearningresource()* eine Einschränkung auf den Typ *learningresource* vorgegeben ist. Da im ersten Parser neue Container (XMLDOCUMENT) für die XML-Fragmente erzeugt werden, müssen alle Eigenschaften, die mit dem initialen XMLDOCUMENT übergeben wurden, auch in die einzelnen Fragmente kopiert werden. Für jedes dieser Fragmente wird daraufhin die Methode *setLearningresource()* des XMLPCTRL aufgerufen, welche die Referenz auf den LRCTRL auflöst und diesem über die Methode *insertDocument()* das XMLDOCUMENT übergibt.

Ein Teil des LRCTRL ist der FRAGMENTPARSER, welcher für das Eintragen der Suchstrukturen zuständig ist. Eine genaue Erläuterung findet sich in Abschnitt 4.5.1. Hier werden die XML-Pfade über das Metamodell aufgelöst und die Inhalte in die so erhaltenen Tabellen geschrieben. Im Anschluss an diesen relativ umfangreichen Schritt werden die Properties aus dem Fragment-Container ausgelesen, falls nicht explizit gesetzt teilweise mit Standardwerten überschrieben und dann über das Entity-Bean in die entsprechende Verwaltungstabelle eingetragen. Auch das Fragment wird dort komplett gespeichert.

Erst wenn diese Schritte abgeschlossen sind wird die Kontrolle zurück an den XMLPCTRL übergeben, der die globale Id des ersten eingetragenen Dokumentes an den aufrufenden Client zurückliefert.

4.5. Spezielle Komponenten

Im folgenden sollen einige spezielle Komponenten des XMLProcessors etwas näher beleuchtet werden.

4.5.1. FragmentParser

Wie bereits erwähnt ist der FRAGMENTPARSER ein *content- und error-handler*, welcher von den Typ-Controllern aufgerufen wird, um die Informationen aus dem XML-Fragment in Suchstrukturen in der Datenbank festzuhalten. Die verwendete Umgebung geht aus Abbildung 5 hervor.

Essentiell beim FRAGMENTPARSER ist, dass der Ablauf möglichst generisch ist, und nicht für jeden XML-Pfad explizit Zieltabelle und -attribut im Code festgehalten werden. Dazu wird das bereits erwähnte Metamodell verwendet (siehe Abschnitt 4.3).

Spätes Schreiben Weiterhin wurde beim der Realisierung Wert darauf gelegt, dass die Interaktion mit der Datenbank möglichst spät, d. h. am Ende des Parse-Vorgangs erfolgt. Dies hat den Grund, dass erst am Ende des Parse-Vorgangs alle zu schreibenden Attribute für die jeweiligen Tabellen feststehen. So hat man die Möglichkeit, ein komplettes Tabellentupel auf einmal zu schreiben. Wird die Strategie verfolgt, dass möglichst früh ein Tupel geschrieben wird, so muss beim ersten Eintrag in die zugehörige Tabelle ein `INSERT` gemacht werden, später dann nur noch `UPDATE`s, da das Tupel ja bereits existiert. Zusätzliche Schwierigkeit ist dabei, dass vor dem `UPDATE` jedes Mal geprüft werden müsste, ob schon ein Eintrag für das Tupel existiert. An dieser Stelle

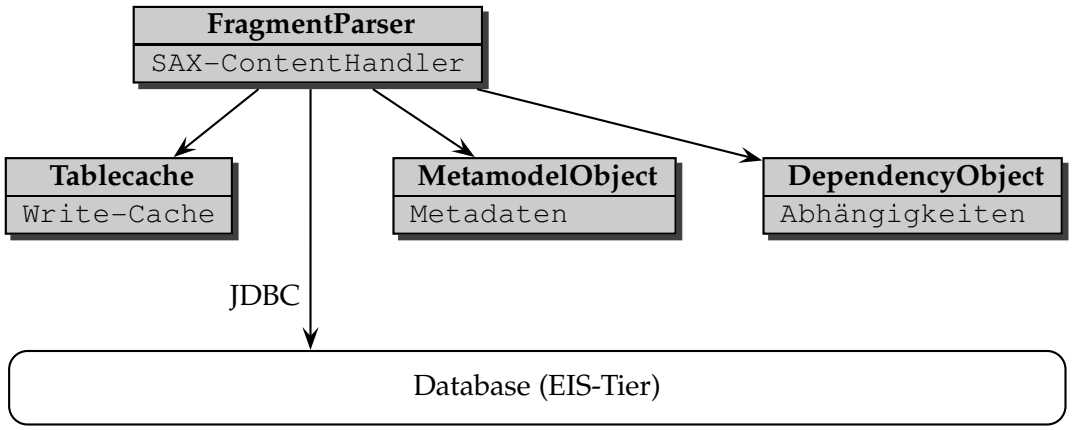

Abbildung 5: Details FragmentParser

trifft man auf das Problem, welches denn das richtige Tupel ist, denn nicht jedes zu schreibende Tupel kommt tatsächlich nur ein Mal vor.

Da es im XML-Schema auch mehrfach vorkommende Elemente gibt (z. B. Autoren, URLs, etc.) müssen zusammengehörige Werte auch tatsächlich zusammengehörig erfasst werden. Hierzu wurde die Datenstruktur TABLECACHE implementiert (siehe Abschnitt 4.2.4). Hier werden Informationen über zu schreibenden Daten tabellenspezifisch erfasst; für jede Tabelle können auch mehrere Tupel, codiert durch die *orderid* gespeichert werden. An dieser Stelle hilft die Annahme, dass komplette Tupel immer örtlich lokal beieinander stehen, d. h. sobald die *orderid* inkrementiert wurde kann das "alte" Tupel nicht mehr referenziert werden. Die *orderid* wird inkrementiert, sobald ein mehrfach vorkommendes XML-Element geöffnet wird, da ab diesem Zeitpunkt keine Daten mehr für das vorherige Tupel anfallen.

Systemkontrolliertes Vokabular Es gibt zwei verschiedene Arten von Vokabularien im Datenmodell von META-AKAD. Einige Felder werden durch ein systemkontrolliertes Vokabular belegt, andere können beliebig bei Bedarf erweitert werden. Dieser Unterschied ist auch im Metamodell festgelegt und muss beim Parsen des XML-Dokumentes beachtet werden.

Bei einem Attribut, das einem kontrollierten Vokabular unterliegt, wird geprüft, ob der verwendete Eintrag im Vokabular enthalten ist. Falls dies nicht der Fall ist, wird der Import abgelehnt.

Bei unkontrollierten Feldern wird überprüft, ob der Eintrag schon existiert und in diesem Fall der passende Schlüssel ausgelesen und verwendet. Sollte der Eintrag nicht existieren, wird er mit einem neuen Schlüssel eingetragen und dieser neue Schlüssel verwendet.

Das Vorgehen bei Feldern mit nicht-systemkontrolliertem Vokabular weicht von der Regel des späten Schreibens ab. Neue Vokabular-Einträge werden sofort in die Datenbank eingetragen. Dies stellt zum Einen eine konsistente Datensicht sicher (da die Datenbank-Tabelle durch den Schreibzugriff gesperrt wird), außerdem muss beim Eintra-

gen ein neuer Schlüssel für den Vokabulareintrag generiert werden, was eine Sonderbehandlung erfordert und von daher nicht am Schluss in der Standard-Prozedur zum Zurückschreiben der Daten in die Datenbank erfolgen kann.

Schlagworte / Klassifikationen Einen Sonderfall stellt die Behandlung von Schlagworten und Klassifikationen dar. Das Datenmodell sieht keine direkte Trennung der verschiedenen Schlagwortkategorien vor. Daher sind die meisten Schlagworte sogar in zwei verschiedenen Tabellen gespeichert. Konkret existiert eine Tabelle, in der alle Schlagworte des Dokumentes (egal welchen Typs) abgelegt werden, dazu für einige Schlagwortkategorien noch je eine zusätzliche (typspezifische) Tabelle, in der Zusatzinformationen abgelegt werden.

Da der Typ des Schlagwortes (das Beschlagwortungsschema) nicht im XML-Element-Namen codiert ist, sondern als zusätzliches XML-Attribut eines Schlagwort-Elementes übergeben wird, musste das Metamodell ein wenig modifiziert und die Verarbeitung eines Subject-Elementes in der Parser-Verarbeitung als Sonderfall abgehandelt werden.

In dieser Sonderbehandlung wird ein künstlicher XML-Pfad erzeugt, bestehend aus dem Standard-Subject-Pfad, ergänzt um den Schema-Namen, (z. B. `metabase.learningresource.subject.swd`). Dieser Pfad lässt sich über das Metamodell zu der konkreten Tabelle auflösen, in der Schlagworte diesen Typs eingetragen werden müssen. Eintragungen in weitere Tabellen lassen sich über Abhängigkeiten durch das Metamodell auflösen.

Eine ganz eigene Behandlung ist für die Klassifikation nach der Regensburger Verbund-Klassifikation (RVK) [RVK] nötig. Dieses Schema besteht aus einem festen Vokabular aus Notationen (Schlüsseln) und textuellen Beschreibungen dieser Schlüssel. Diese Klassifikationen sind in einer Baumstruktur angeordnet und haben damit jeweils einen Vorfahren. Da für die Realisierung der Suchanfrage immer ein Pfad bis hin zur Wurzel benötigt wird, muss im Verlauf des Imports der Baum von unten nach oben verfolgt werden und alle Knoten des Pfades für dieses Dokument gespeichert werden. Für die Materialisierung der RVK existiert im System eine Referenztabelle, in der nahezu alle möglichen Einträge (über 800.000), jeweils mit ihrem Vater-Knoten eingetragen sind. Beim Import werden die Klassifikationen, nach denen das Dokument tatsächlich klassifiziert wurde, sowie alle Vorfahren in den Datenbankeintrag übernommen. Besonders zu berücksichtigen ist, dass keine Duplikate auftreten, wenn eine Klassifikation sowohl wegen ihrer direkten Erwähnung, als auch durch die Tatsache aufgenommen werden muss, dass sie Teil des Pfades von der Wurzel zu einer vergebenen Klassifikation ist.

Behandlung von Autoren Neben der Ausnahmebehandlung für Schlagworte mussten auch Einschränkungen in Bezug auf die Generizität bei der Behandlung von Autoren hingenommen werden. Das Besondere an den Autoren ist, dass auf Grund der Einschränkungen des Datenbank-Schemas bzw. des Verarbeitungskonzeptes (keine zusammengesetzten Fremdschlüssel) für Einträge in der Tabelle *creator* ein eigener Primärschlüssel generiert werden muss. In allen anderen Tabellen leitet sich die Referenz allein aus der ID des importierten Dokumentes her. Damit dieser Schlüssel explizit generiert wird muss auch hier auf ein gewisses Stück Generizität verzichtet werden.

Einfüge-Reihenfolge Um nicht von zur Zeit noch proprietären Datenbankerweiterungen abhängig sein zu müssen wurde ein besonderes Gewicht auf die Einfügereihenfolge von Tupeln gelegt. Aus Gründen der Sicherung der Datenkonsistenz sind datenbankseitig viele Fremdschlüsselbeziehungen eingerichtet. Da man sich nicht auf die Möglichkeit verlassen kann, die Überprüfung der Einhaltung der Schlüsselbeziehungen an das Ende der Import-Transaktion verschieben zu können, müssen jederzeit alle Fremdschlüsselbeziehungen erfüllt sein. Diese Lösung kommt ohne spätere `UPDATE`s aus, das verringert die Anzahl der nötigen Datenbankoperationen.

Um dies zu realisieren wurde im Schlussteil des Imports (beim Schreiben aller gespeicherten Werte in die Datenbank) eine Heuristik implementiert, die gewährleistet, dass alle voneinander abhängigen Tupel geschrieben werden, jedoch keine Schlüsselbeziehungen verletzt werden. Trotzdem wird dabei vermieden, die korrekte Einfügereihenfolge fest vorgeben zu müssen.

Dazu werden alle Einfüge-Operationen, die von der Datenbank abgelehnt werden, zurückgestellt. Jedes erfolgreich eingefügte Tupel erhöht die Chancen für nachfolgende Tupel, erfolgreich eingefügt werden zu können. Nachdem jede Einfüge-Operation einmalig ausgeführt wurde, wird versucht, die zurückgestellten Einträge in die Datenbank einzufügen. Erfolgreich abgeschlossene Operationen werden aus der Rückstellungsliste entfernt. Dies geschieht in einer Schleife so lange, bis entweder alle Tupel eingefügt wurden oder bis in einem Schleifendurchlauf kein neues Tupel eingefügt werden konnte. Sollte dies der Fall sein, so hat das Tupel auch in der nächsten Iteration keine Chance, erfolgreich eingefügt werden zu können, da der Zustand der Datenbank unverändert geblieben ist. In diesem Fall wird mit einer Fehlermeldung abgebrochen. Fehler dieser Art können etwa auftreten, wenn bei Attributen mit festem Vokabular ein unbekannter Inhalt übermittelt werden soll oder einfach ein Datenbankfehler auftritt. Die eingesetzte Heuristik ist noch optimierungsfähig, da sie keinerlei Daten über die Abhängigkeit zwischen zu schreibenden Tupeln beachtet. Hier könnte also bereits auf Anwendungsebene die Reihenfolge festgelegt werden anstatt das Einfügen erst zu probieren. Um zu vermeiden, dass durch die nichtdeterministische Reihenfolge beim Einfügen gerade eine Kerntabelle (z. B. *learningresource*), von der direkt oder indirekt alle anderen Tabellen über Fremdschlüsselbeziehungen abhängen, sehr spät geschrieben wird, werden die Tabellen *learningresource* und *lrmaintain* direkt zu Beginn in die Datenbank übertragen. Die Terminierung der Heuristik ist auf jeden Fall dadurch sichergestellt, dass die Anzahl der zu schreibenden Tupel endlich ist, jedes Tupel nur ein Mail geschrieben wird und die Schleife abbricht, sobald in einem Durchlauf kein Tupel geschrieben werden konnte.

4.5.2. LRCtrl

Beispielhaft für alle Typ-Controller soll hier der Controller für *learningresource* verwendet werden, da zum einen alle Typ-Controller ähnlich aufgebaut sind und zum anderen der LRCTRL der umfangreichste ist (siehe Abbildung 6).

Der Controller zur Behandlung von *learningresource* hält Methoden bereit, um Dokumente einzufügen, zu aktualisieren, zu löschen und neue Referenzen zu knüpfen.

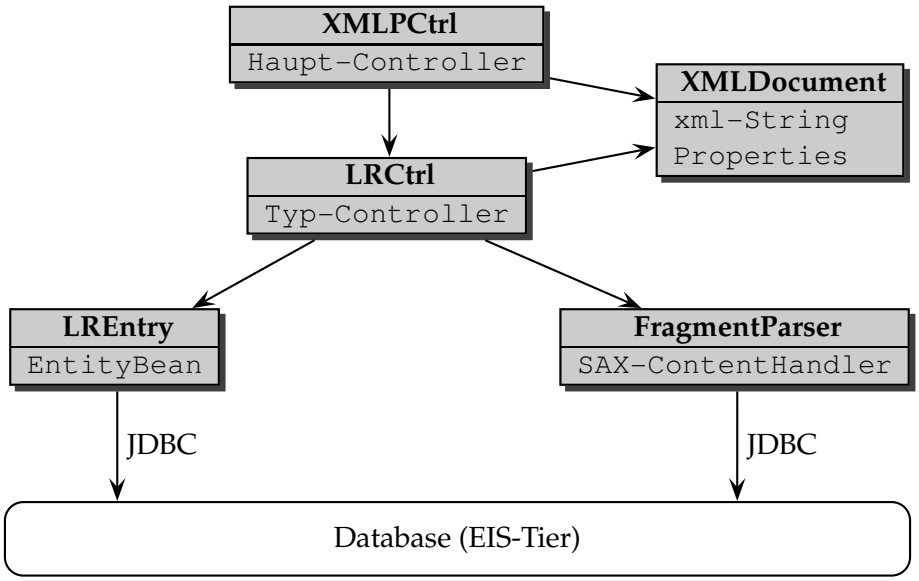

Abbildung 6: Details LRCtrl

Einfügen Im Rahmen einer Einfüge-Operation ruft der LRCTRL den Parser auf, um die Suchstrukturen für dieses neue Dokument zu erzeugen (siehe Abschnitt 4.5.1). Wenn diese Aktion erfolgreich abgeschlossen werden konnte, können die restlichen Eintragungen erfolgen. Durch den Parser-Lauf wurde bereits in der Verwaltungstabelle für Lerndokumente ein Basiseintrag bestehend aus globalem (guID) und lokalem (lrid) Identifier angelegt. Darauf basierend wird das entsprechende LRENTRY-Bean geladen, alle weiteren Aktionen basieren darauf. Aus dem übergebenen XMLDOCUMENT werden alle nötigen Eigenschaften ausgelesen, falls nicht vorhanden werden Standardwerte erzeugt. Schließlich werden alle diese Attribute mittels des Entry-Beans gesetzt.

Aktualisieren Das Aktualisieren von Daten ist in der momentanen Variante relativ trivial gelöst, allerdings sind auch dort einige Besonderheiten zu beachten.

Hauptsächlich basiert das Aktualisieren von Lerndokumenten darauf, das alte Dokument zu löschen und das aktualisierte Dokument neu einzutragen. Es wird jedoch beachtet, dass die Einträge für Erstellzeit (*created_at*) und Ersteller (*created_by*) erhalten bleiben. Diese Werte werden dem XMLDOCUMENT als Property mitgegeben. Für den Erhalt der guID ist bereits der XMLPCTRL zuständig. Während der Aktualisierung muss das Dokument für jeden anderen Nutzer gesperrt werden. Dazu existiert eine Lock-Verwaltung, die auf der guID basiert.

Eine Besonderheit bei der Aktualisierung ist die Tatsache, dass Gutachten nicht aktualisiert werden können. Von Gutachten bleiben immer alle Versionen im System, sie referenzieren jeweils ihren Vorgänger.

Die Implementierung der *updateDocument()*-Methode stellt in der momentanen Implementierung nur sicher, dass einzelne Dokumente ohne Referenzen aktualisiert werden können. Komplexere Dokumente werden abgelehnt.

Löschen Das Löschen von Daten wird durch Unterstützung der Datenbank sehr vereinfacht. Den Fremdschlüsseln wurden Anweisungen beigefügt, dass Löschungen kaskadiert werden sollen, d. h. wenn ein Basiseintrag in der Tabelle *learningresource* gelöscht wird, dann werden auch alle davon abhängigen Einträge gelöscht. Explizit ausgeführt werden muss die Löschung also nur für den Basiseintrag und den Eintrag in der Verwaltungstabelle (*lrmaintain*). Dieser könnte zwar auch automatisiert durch die Datenbank gelöscht werden, es wurde aber aus Gründen der Cache-Validität bevorzugt, den J2EE-Container das Löschen des zugehörigen Entity-Beans ausführen zu lassen.

Referenzen Der LRCTRL verfügt im Gegensatz zu den anderen beiden Typ-Controllern über einige Erweiterungen. Diese werden dazu benötigt, Referenzen zwischen Lerndokumenten auszubilden.

Diese Methoden sorgen zum einen dafür, dass die Relation zwischen zwei Dokumenten in die entsprechenden Tabellen in den Suchstrukturen eingetragen wird. Dies ist ein simples `INSERT` eines Tupels. Desweiteren muss aber das Fragment des referenzierenden Lerndokumentes so abgeändert werden, dass eine neue Referenz sich auch im Fragment widerspiegelt. An dieser Stelle wurde *Castor* [Exo03] eingesetzt.

Castor transformiert das XML-Fragment in eine Objektstruktur, so dass einfach ein neues Referenz-Objekt hinzugefügt werden kann. Auch die Rücktransformation übernimmt Castor. Ohne die Transfereigenschaften von Castor hätte man "von Hand" dafür sorgen müssen, dass an der richtigen Stelle (und in der richtigen Reihenfolge) die Referenz eingetragen wird. Auch beim Löschen wird Castor eingesetzt, um jeweils die Fragmente der referenzierenden Dokumente zu modifizieren.

4.6. Anmerkungen

In diesem Abschnitt soll auf einige Besonderheiten hingewiesen werden, die vielleicht auch später bei einer Änderung der Implementierung berücksichtigt werden sollten.

4.6.1. Globale Identifier

Da Dokumente bei Kenntnis ihrer guID problemlos aus dem System abzurufen sind, wurde ein Mechanismus implementiert, damit nicht wahllos alle Dokumente einfach durch Inkrementieren der guID "gestohlen" werden können. Hierzu wird beim Generieren der guID beim Einfügen des Dokumentes ein zufälliger fünfstelliger Präfix vor die interne ID des Dokumentes gehängt. So müssen für jedes Dokument im worst-case 100000 Versuche gemacht werden, bis das Dokument abgerufen werden kann. Dies ist zwar keine sichere Methode, kann aber sicher einige Versuche, den kompletten Datenbestand abzuziehen, verhindern.

4.6.2. Referenzen zwischen Dokumenten

Die Behandlung von Referenzen zwischen Dokumenten wirft einige Komplikationen auf. Darum soll hier beschrieben werden, wo genau Referenzen auftreten und welche Probleme sie dort mit sich bringen.

Referenzen zwischen Lerndokumenten Zwischen Lerndokumenten treten drei verschiedene Arten von Referenzen auf, die jedoch alle gleich behandelt werden können (*hasPart, hasVersion, hasFormat*). Diese Relationen sind zum einen in den Tabellen der Suchstrukturen eingetragen, zum anderen existieren natürlich Verweise in den XML-Fragmenten. Beim Hinzufügen und Löschen muss immer auf beide Rücksicht genommen werden.

Das komplette Einfügen eines "Baumes" von Lerndokumenten ist kein Problem. In diesem Fall ist das Fragment bereits korrekt und beim Anlegen der Suchstrukturen werden alle benötigten Einträge erledigt.

Das nachträgliche Einfügen von Referenzen hat eine Änderung des Fragmentes (durch Castor) zur Folge, außerdem muss ein Eintrag in die entsprechende Tabelle der Suchstrukturen gemacht werden.

Das nachträgliche Löschen von Referenzen ist nicht vorgesehen. Allerdings ist beim Löschen von Lerndokumenten folgendes zu beachten. Hier muss bei allen referenzierenden Dokumenten die Referenz auf das zu löschenden Lerndokument entfernt werden. In den Suchstrukturen ist dies kein Problem, dies wird datenbankseitig erledigt. Manuell zu bearbeiten ist allerdings das XML-Fragment des referenzierenden Dokumentes. Auch müssen referenzierende Benutzerkommentare und Gutachten gelöscht werden, dies ist vollständig manuell zu veranlassen; die referenzierenden Kommentare und Gutachten müssen über ihre entsprechenden *finder*-Methoden herausgesucht werden.

hasReview Für das Einfügen eines Gutachtens zusammen mit seinem Lerndokument gilt das gleiche wie für einen Baum von Lerndokumenten. Hier ist nichts besonders zu beachten.

Das nachträgliche Hinzufügen eines Gutachtens hat zur Folge, dass zum einen die Referenz in das Fragment des begutachteten Lerndokumentes eingetragen werden muss (dies geschieht idealerweise mittels Castor), zum anderen müssen auch hier die Suchstrukturen angepasst werden. Des weiteren existiert in der Verwaltungstabelle bzw. im entsprechenden Entry-Bean der Gutachten ein Attribut *belongs_to*. Hier wird die guID des begutachteten Lerndokumentes gespeichert.

Beim Löschen eines Gutachtens werden Suchstruktur-Referenzen ebenso gelöscht. Manuell zu behandeln bleiben wiederum die Referenzen im Fragment des Lerndokumentes. Auch hier wird Castor verwendet.

Ein Sonderfall ergibt sich bei den Gutachten durch die eingesetzte Versionierung. Im Gegensatz zu Lerndokumenten und Benutzerkommentaren werden Gutachten nicht einfach aktualisiert oder gelöscht, sondern es wird eine neue Version des Gutachtens angelegt, welche einen Verweis auf ihre Vorversion trägt. Für den Endnutzer ist dieses Verfahren transparent, er bekommt immer die aktuellste Instanz des Gutachtens geliefert.

Da sich mehrere Versionen eines Gutachtens gegenseitig referenzieren, muss beim Löschen geprüft werden, welche Instanzen von Gutachten gegebenenfalls ebenfalls gelöscht werden sollten. Hier gibt es drei Ansätze. Zum einen könnten alle tiefer liegenden, also "älteren", Gutachten gelöscht werden, zum anderen könnte das gelöschte Gutachten übersprungen werden. Um dem Konzept der Versionierung am nähesten

zu kommen, müsste ein zusätzliches Flag eingeführt werden, was ein Gutachten als gelöscht markiert, auch wenn es die aktuellste Version eines Gutachtens ist. In diesem Fall ist aus Sicht des Endbenutzers dieses Gutachten nicht vorhanden.

Für alle Varianten gäbe es Verwendung. Die genaue Semantik des Löschens von Gutachten ist jedoch noch nicht endgültig spezifiziert.

hasUsercomment Das Einfügen eines Benutzerkommentars gemeinsam mit seinem Lerndokument erfolgt analog zum Einfügen eines Gutachtens.

Ein nachträgliches Einfügen des Kommentars hat die bereits mehrfach genannten Auswirkungen. Suchstrukturen und Fragment des kommentierten Dokumentes müssen manuell angepasst werden. Im Gegensatz zum Gutachten gibt es keine Referenzen innerhalb von Benutzerkommentaren.

Beim Löschen des Kommentars werden analog zum Gutachten Referenzen in den Suchstrukturen automatisch gelöscht. Das Fragment des kommentierten Dokumentes muss manuell mit Castor bearbeitet werden. Da bei Benutzerkommentaren ansonsten keine Referenzen existieren, muss an dieser Stelle nichts weiter beachtet werden.

5. Zusammenfassung

Im Rahmen der Implementierung einer generischen Verwaltungskomponente für XML-basierte Daten in relationalen Datenbanksystemen wurden, eingebettet in eine J2EE-Umgebung, innerhalb des EJB-Tier zahlreiche Klassen entwickelt. Eine Kernkomponente dieser Implementierung ist die Klasse FRAGMENTPARSER, welche konkret für alle Daten innerhalb eines XML-Dokumentes die Zieltabelle innerhalb der Datenbank ermittelt und den Wert dort ablegt.

5.1. Ergebnis

Durch die erstellten Klassen, können XML-Dokumente beliebigen Typs (bezogen auf das Projekt META-AKAD) und beliebiger Schachtelungstiefe in die Datenverwaltungskomponente importiert und gelöscht werden sowie Dokumente ohne Referenzen auch aktualisiert werden.

5.2. Lösungsweg

Gelöst wurde die Problemstellung durch eine Mischung zweier genereller Ansätze. Ein Teil des Problems wurde durch den Einsatz eines Event-basierten SAX-Parsers gelöst. Hierbei wird ein XML-Dokument an einen Parser übergeben und das Auftreten von XML-Elementen, XML-Attributen und reiner Inhalt durch eine Event an den Content-Handler gemeldet, in welchem die nötigen Aktionen codiert werden müssen.

Wenn es um das simple Einfügen oder Löschen von Elementen aus dem XML-String ging, wurde ein anderes Modell benutzt. Hierbei wird das XML-Dokument in eine baumförmige Objektstruktur überführt (in diesem Fall mittels *Castor*). Dieser Baum lässt sich auf einfache Art und Weise durchsuchen um dann an den passenden Stellen Elemente einfügen oder löschen zu können. Dieses Verfahren wurde an den Stellen benutzt, wo einzelne Stellen im XML-Dokument gezielt verändert werden sollten (Einfügen und Löschen von Referenzen auf andere Dokumente).

5.3. Leistungsbetrachtung

Obwohl beim implementierten Verfahren im Laufe eines Dokument-Imports zahlreiche Aktionen ausgeführt werden müssen, darunter vor allem hauptsächlich selektive aber auch manipulative Datenbankanfragen, konnte der Zeitbedarf für einen Import auf einem eher unterdurchschnittlichen System für ein XML-Dokument von für den Einsatzzweck des Projektes typischer Länge im Bereich von unter 5 Sekunden gehalten werden. Das bereits vorgestellte Beispiel-Dokument, für das im Anhang ein exemplarischer Import gemacht wird, konnte in ca. 1 Sekunde in die Datenbank importiert werden. Da im Rahmen des Projektes META-AKAD die Manipulation der vorhandenen Daten hauptsächlich durch ein Web-Interface erfolgt, ist die Antwortzeit relevant für die Güte des Dienstes.

6. Ausblick

Am momentanen Punkt der Realisierung sind alle Punkte der Anforderung realisiert. Dennoch bleiben zahlreiche Verbesserungs- und Optimierungsmöglichkeiten.

6.1. Codebereinigung

Zuallererst könnte der Code vor allem des Parsers für die Einfügung der Suchstrukturen um einige proprietäre, zum Teil Datenbankhersteller-spezifische Lasten befreit werden und einige nicht beliebig erweiterungsfähige SQL-Anfragen in den zentralen CONFIGURATIONSTORE ausgelagert werden, damit bei einer Änderung am System nicht der Code angepasst werden muss, sondern nur die Konfigurationseinstellungen in der Datenbank.

6.2. Abhängigkeitsüberprüfung

Im Codeabschnitt der Abhängigkeitsüberprüfung im FRAGMENTPARSER werden in der momentanen Realisierungsstufe relativ viele rekursive Aufrufe zur Überprüfung der Abhängigkeit von Tabelleneinträgen gemacht.

So ist über den Verlauf der Implementierung ein System gewachsen, dass nach jedem eingetragenen Attribut (normalerweise nur eine einzelne Tabellenspalte) *alle* Abhängigkeiten erneut überprüft (jeweils rekursiv). Dies ist zum Teil darin begründet, dass schlecht vorhergesagt werden kann, wann ein Quellattribut für eine Abhängigkeit gesetzt wird. Aufgrund der Möglichkeit, dass mehrere Abhängigkeiten von demselben Attribut in einer Kette vorhanden sind, reicht eine einzige Überprüfung am Schluss des Import nicht aus.

Eine Möglichkeit wäre, am Ende so lange rekursiv zu prüfen, bis kein neues Feld mehr hinzukommt. Dies bleibt allerdings noch zu validieren. Probleme kann es hier bei Tabellen geben, in denen mehrere Tupel eingetragen werden sollen, da unter Umständen unklar ist, welches dieser Tupel referenziert werden soll.

6.3. Optimierung der Update-Funktionalität

Die Möglichkeit der Aktualisierung eines Dokumentes ist in der vorliegenden Variante sehr trivial gelöst. Nach Anfordern der entsprechenden Sperre wird das bestehende Dokument gelöscht und die neue Version des Dokumentes muss den Importvorgang einmal komplett durchlaufen. Diese Variante ist deutlich einfacher zu implementieren als ein vollwertiger Update-Mechanismus, der prüft, an welcher Stelle Änderungen aufgetreten sind und nur diese Stellen anpasst. Je nach Effizienz des Update-Algorithmus ist auch die Laufzeit des Re-Imports kürzer. Der derzeit implementierte Algorithmus hat des weiteren den Nachteil, dass er nur auf einfache Dokumente ohne Referenzen angewandt werden kann. Anderenfalls würden Referenzen gelöscht werden, die erst manuell nachgeführt werden müssten.

Ein optimierter Algorithmus hätte den Vorteil, dass nur kleine Bereiche der Datenbank kurzfristig gesperrt werden müssten, statt wie bisher großflächige Teile der Suchstrukturtabellen.

6.4. Schnittstellenkommunikation

Im Zuge einer schöneren bzw. einfacheren oder aussagekräftigeren Kommunikation über die Schnittstellen könnte die Exception-Behandlung verbessert werden. So könnten auf spezielle Fehlerfälle zugeschnittene Exceptions geworfen werden, die dann auf Anwenderseite bzw. im Web-Tier gezielter verarbeitet werden könnten. So würde dadurch auch die Darstellung für den Endanwender verbessert, da auch ihm dann aussagekräftige Fehlermeldungen zur Verfügung stünden.

Der momentane Stand ist, dass zwar in allen Fehlerfällen eine Exception geworfen wird, doch es handelt sich nahezu durchgehend um, häufig auch geschachtelte, Standard-Exceptions, in deren Stack-Trace irgendwo eine textuelle Meldung des Fehlers enthalten ist. Sie dienen hauptsächlich zur Meldung, dass überhaupt ein Fehler aufgetreten ist und ermöglichen dem Entwickler eine genauere Fehlerlokalisierung.

Abbildungsverzeichnis

SQL-Fragmente

Literatur

[Bro02] David Brownell. *SAX2*. O'Reilly, 2002.

[Exo03] Exolab. The Castor Project, 2003.
`http://castor.exolab.org/`.

[Met02] MetaAKAD. Beschreibung des Projektes META-AKAD, 2002.
`http://www.akleon.de/About`.

[ML01] Brett Mc Laughlin. *Java & XML, 2nd Edition*. O'Reilly, September 2001.

[RVK] Regensburger Verbundklassifikation (RVK).
`http://www.bibliothek.uni-regensburg.de/Systematik/systemat.html`.

[Sax00] SaxProject. The Simple API for XML, 2000.
`http://www.saxproject.org/`.

[SUN02] SUN. Design Pattern: SessionFacade, 2002.
`http://java.sun.com/blueprints/patterns/SessionFacade.html`.

[SUN03a] SUN. Enterprise JavaBeans Technology, 2003.
`http://java.sun.com/products/ejb/`.

[SUN03b] SUN. Java 2 Platform, Enterprise Edition, 2003.
`http://java.sun.com/j2ee/`.

[SUN03c] SUN. Java 2 Platform, Standard Edition, 2003.
`http://java.sun.com/j2se/`.

[SUN03d] SUN. Java Naming and Directory Interface, 2003.
`http://java.sun.com/products/jndi/`.

[SUN03e] SUN. JavaServer Pages Technology, 2003.
`http://java.sun.com/products/jsp/`.

[WD02] Workgroup W3C-DOM. Document Object Model, 2002.
`http://www.w3.org/DOM/`.

[WX01] Workgroup W3C-XML. XML Schema, 2001.
`http://www.w3.org/XML/Schema/`.

A. Beispiel (Einfügen eines Dokumentes)

Im folgenden Beispiel soll das Einfügen eines Dokumentes (genauer: einer *learningresource*) exemplarisch gezeigt werden. Die hier behandelte Learningresource ist absichtlich sehr klein gehalten, um die Übersichtlichkeit zu wahren.

A.1. Das XML-Dokument

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<metabase xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://www.akleon.de/xsd/metabase.xsd">

    <learningresource guID="TEST1">
        <title>
            <main lang="DE" origin="intellectual">Testdokument</main>
        </title>
        <creator>
            <name origin="intellectual">Fudeus, Stephan</name>
        </creator>
        <identifier origin="automatic" URI="http:/www.akleon.de/test.html" />
    </learningresource>
</metabase>
```

Dieses XML-Dokument wird nun (verpackt in ein XMLDOCUMENT) ohne zusätzliche Eigenschaften dem XMLPCTRL über die Methode *addLearningresource()* übergeben.

Der Ablauf des Einfügens ist am generierten Debug-Output gut zu erkennen. Aus Gründen der Übersichtlichkeit wird dieser Output etwas gekürzt angegeben:

A.2. Der Ablauf

A.2.1. XMLPCtrl / ParserContentHandler

```
[XMLPCtrlEJB] addLearningresource starting...
[ParserContentHandler] Started parsing document
[ParserContentHandler] Resolving learningresource-id for xmlid TEST1
[ParserContentHandler] No mapping in HashMap -> resolving new one
[IdGenerator] getNewLRid called
[IdGenerator] Executing update
[IdGenerator] Executing query
[IdGenerator] getNewLRid finished
[ParserContentHandler] resolved id LR0771610002. Adding to HashMap
[ParserContentHandler] Added learningresource with id LR0771610002
[XMLPCtrlEJB] Forwarding learningresource to lrctrl
[XMLPCtrlEJB] Looking for LRCtrl
[XMLPCtrlEJB] Found LRCtrl
```

Hier wurde also mittels des IDGENERATOR eine neue eindeutige Id generiert und das XML-Fragment dahingehend geändert. An dieser Stelle sieht das Fragment dann mit umgeschriebener guID folgendermaßen aus:

```
<learningresource guID="LR0771610002">
    <title>
        <main lang="DE" origin="intellectual">Testdokument</main>
    </title>
    <creator>
        <name origin="intellectual">Fudeus, Stephan</name>
    </creator>
    <identifier origin="automatic" URI="http:/www.akleon.de/test.html"></identifier>
</learningresource>
```

A.2.2. LRCtrl / FragmentParser

```
[LRCtrlEJB] Received document for insertion with id LR0771610002
[LRCtrlEJB] starting preparing indexstructures for guid LR0771610002
[LRCtrlEJB] Handing over to FragmentParser
[FragmentParser] Starting parsing of metabase-document
```

Hier wird jetzt für jedes gefundene Element (falls es ein Element ist, das mehrfach vorkommen kann) eine *orderId* angelegt, dann alle Attribute behandelt und schließlich der Inhalt zum Schreiben im jeweiligen Tablecache festgehalten. Sollte noch kein Tablecache existieren, muss der natürlich auch noch angelegt werden.

```
[FragmentParser] found element learningresource
[FragmentParser] getting orderId
[FragmentParser] creating new orderId-counter
[FragmentParser] preparing dataset for mb_learningresource.lrid with javatype
                 integer and sqltype DECIMAL
[FragmentParser] content: 10002
[FragmentParser] creating new tablecache for table mb_learningresource and orderid 1
[FragmentParser] starting handling of attribute with path learningresource.guID
[FragmentParser] content: LR0771610002
[FragmentParser] found metaModelObject for path learningresource.guID
[FragmentParser] fetching dependencies
[FragmentParser] findingDependencyObjectByTable
[FragmentParser] finding dependencycollection for path learningresource.guID
[FragmentParser] dependencies not found in cache
[FragmentParser] finding dependencyObjectCollection for path learningresource.guID
[FragmentParser] found dependencyObjectCollection for path learningresource.guID
[FragmentParser] storing dependencies in cache by metamodel
[FragmentParser] storing dependencies in cache by fromtable
[FragmentParser] adding new map: mb_lrmaintain.lridref
[FragmentParser] found no dependencies for learningresource.guID
[FragmentParser] preparing dataset for mb_lrmaintain.guid with
                 javatype java.lang.String and sqltype CHAR
[FragmentParser] content: LR0771610002
[FragmentParser] creating new tablecache for table mb_lrmaintain and orderid 1
```

Im obigen Abschnitt wurde jetzt auch auf Abhängigkeiten geprüft. Aus Optimierungsgründen wurde dafür zuerst eine Cache-Struktur überprüft. Falls dort keine Abhängigkeiten registriert sind, wird die Anfrage an die Datenbank gestellt, und das Ergebnis über zwei Schlüssel (*metamodel* und *fromtable*) im Cache gesichert. In diesem Fall waren aber keine Abhängigkeiten vorhanden.

Im weiteren Verlauf ist die Abhängigkeitsüberprüfung verkürzt angegeben. Auf Besonderheiten wird gesondert hingewiesen.

```
[FragmentParser] found element title
[FragmentParser] getting orderId
[FragmentParser] creating new orderId-counter
```

```
[FragmentParser] found element main
[FragmentParser] getting orderId
[FragmentParser] creating new orderId-counter
```

```
[FragmentParser] starting handling of attribute with path learningresource.title.main.lang
[FragmentParser] content: DE
[FragmentParser] found metaModelObject for path learningresource.title.main.lang
[ ... Überprüfung von Abhängigkeiten ... ]
[FragmentParser] value was added or is system-controlled
[FragmentParser] SQL: SELECT vid FROM value_language WHERE myvalue='DE'
[FragmentParser] preparing dataset for lr_title.main_langidref with
                 javatype integer and sqltype DECIMAL
[FragmentParser] content: 2
[FragmentParser] creating new tablecache for table lr_title and orderid 1
[FragmentParser] starting handling of attribute with path learningresource.title.main.origin
[FragmentParser] content: intellectual
[FragmentParser] found metaModelObject for path learningresource.title.main.origin
[ ... Überprüfung von Abhängigkeiten ... ]
[FragmentParser] value was added or is system-controlled
[FragmentParser] SQL: SELECT vid FROM value_origin WHERE myvalue='intellectual'
[FragmentParser] preparing dataset for lr_title.main_originidref with
                 javatype integer and sqltype DECIMAL
[FragmentParser] content: 4
[FragmentParser] adding dataset to existing tablecache for table
                 lr_title and orderid 1
```

Im letzten Abschnitt kam es gleich zwei Mal vor, dass ein Attribut mit systemkontrolliertem Vokabular benutzt wurde (*origin* und *lang*). Bei beiden Fällen ist zu sehen, dass eine direkte Anfrage an die Datenbank gestellt wird, um den eigentlich einzutragenden Primärschlüssel abzufragen.

```
[FragmentParser] finishing element main
[FragmentParser] starting handling of content for path learningresource.title.main
[FragmentParser] content: Testdokument
[FragmentParser] found metaModelObject for path learningresource.title.main
[FragmentParser] found xmltype ELEMENT.SINGLEVALUE -> handing over to handleValue()
[ ... Überprüfung von Abhängigkeiten ... ]
[FragmentParser] preparing dataset for lr_title.main with
                 javatype java.lang.String and sqltype LVARCHAR
[FragmentParser] content: Testdokument
[FragmentParser] adding dataset to existing tablecache for table
                 lr_title and orderid 1


[FragmentParser] *** fulfilling Dependencies after Element main
[FragmentParser] starting to fulfill table-dependencies
[FragmentParser] dependencies for tables, which are to be written
[FragmentParser] looking for dependencies of table lr_title
[FragmentParser] fieldname resourceidref has dependencies
[FragmentParser] resourceidref is a field we did not write
[FragmentParser] trying to get value of mb_learningresource.lrid from pre-cache
[FragmentParser] preparing dataset for lr_title.resourceidref with
                 javatype integer and sqltype DECIMAL
[FragmentParser] content: 10002
[FragmentParser] adding dataset to existing tablecache for table
                 lr_title and orderid 1
[FragmentParser] finished dependencies for resourceidref
[FragmentParser] fieldname main_originidref has dependencies
[FragmentParser] finished dependencies for main_originidref
[FragmentParser] fieldname main_langidref has dependencies
[FragmentParser] finished dependencies for main_langidref
[FragmentParser] finished dependencies of table lr_title

[FragmentParser] looking for dependencies of table mb_lrmaintain
[FragmentParser] registering that deps are fulfilled for mb_lrmaintain
[FragmentParser] fieldname lridref has dependencies
[FragmentParser] lridref is a field we did not write
[FragmentParser] trying to get value of mb_learningresource.lrid from pre-cache
[FragmentParser] preparing dataset for mb_lrmaintain.lridref with
                 javatype integer and sqltype DECIMAL
[FragmentParser] content: 10002
[FragmentParser] adding dataset to existing tablecache for table
```

```
                mb_lrmaintain and orderid 1
[FragmentParser] finished dependencies for lridref
[FragmentParser] finished dependencies of table mb_lrmaintain
[FragmentParser] looking for dependencies of table mb_learningresource
[FragmentParser] registering that deps are fulfilled for mb_learningresource
[FragmentParser] mb_learningresource has no dependencies
[FragmentParser] finished fulfilling table-dependencies
[FragmentParser] done finishing element main
```

Im letzten Abschnitt fand die Abhängigkeitsüberprüfung bei Schließen eines Elements statt. Hier ist auch zum ersten Mal während dieses Import zu sehen, dass Abhängigkeiten vorhanden sind (und auch erfüllt werden können). Für jede Tabelle, in der schon Daten zum Eintragen vorgemerkt sind, wird getestet, ob in den Cache-Tabellen für die Abhängigkeiten Einträge existieren und diese noch nicht mit Daten belegt sind. Für diese abhängigen Felder wird überprüft, ob die Abhängigkeiten erfüllbar sind, d. h. ob das Zielfeld schon Daten enthält. Falls ja, wird der Eintrag übernommen. In diesem Beispiel sieht man dies bei den Abhängigkeiten `lr_title.resourceidref` und `mb_lrmaintain.lridref`. Sehr kurze Einträge treten auf, wenn zwar Abhängigkeiten existieren, diese jedoch bereits erfüllt wurden d. h. wenn das abhängige Feld bereits ausgefüllt ist.

```
[FragmentParser] finishing element title
[FragmentParser] starting handling of content for path learningresource.title
[FragmentParser] content:
[FragmentParser] finding metaModelObject for path learningresource.title
[FragmentParser] found metaModelObject for path learningresource.title
[FragmentParser] found xmltype ELEMENT.MULTIPLEVALUE.ORDERED -> handing over
                to handleMultipleValue()
[FragmentParser] handling multiple value for path learningresource.title,
                setting to value:
[FragmentParser] preparing dataset for lr_title.orderid with
                javatype integer and sqltype DECIMAL
[FragmentParser] content: 1
[FragmentParser] adding dataset to existing tablecache for table
                lr_title and orderid 1
[FragmentParser] starting handling of path learningresource.title
[ ... Überprüfung von Abhängigkeiten ... ]
```

```
[FragmentParser] *** fulfilling Dependencies after Element title
[FragmentParser] starting to fulfill table-dependencies
[FragmentParser] dependencies for tables, which are to be written
[FragmentParser] looking for dependencies of table lr_title
[FragmentParser] fieldname resourceidref has dependencies
[FragmentParser] finished dependencies for resourceidref
[FragmentParser] fieldname main_originidref has dependencies
[FragmentParser] finished dependencies for main_originidref
[FragmentParser] fieldname main_langidref has dependencies
[FragmentParser] finished dependencies for main_langidref
[FragmentParser] finished dependencies of table lr_title
[FragmentParser] looking for dependencies of table mb_lrmaintain
[FragmentParser] fieldname lridref has dependencies
[FragmentParser] finished dependencies for lridref
[FragmentParser] finished dependencies of table mb_lrmaintain
[FragmentParser] looking for dependencies of table mb_learningresource
[FragmentParser] mb_learningresource has no dependencies
[FragmentParser] done finishing element title
```

In diesem Absatz wurden erneut (wegen Beendigung des Elements *title*) alle Abhängigkeiten geprüft, jedoch keine neuen nicht-erfüllten Abhängigkeiten gefunden.

```
[FragmentParser] found element creator
[FragmentParser] getting orderId
[FragmentParser] creating new orderId-counter
[FragmentParser] registering primary key for creator
[IdGenerator] getNewCreatorVid called
```

```
[IdGenerator] Executing update
[IdGenerator] Executing query
[IdGenerator] getNewCreatorVid finished
[FragmentParser] preparing dataset for lr_creator.vid with
                 javatype integer and sqltype DECIMAL
[FragmentParser] content: 80002
[FragmentParser] creating new tablecache for table
                 lr_creator and orderid 1
[FragmentParser] resolved vid for creator


[FragmentParser] found element name
[FragmentParser] getting orderId
[FragmentParser] creating new orderId-counter
[FragmentParser] starting handling of attribute with path learningresource.creator.name.origin
[FragmentParser] content: intellectual
[FragmentParser] finding metaModelObject for path learningresource.creator.name.origin
[FragmentParser] found metaModelObject for path learningresource.creator.name.origin
[ ... Überprüfung von Abhängigkeiten ... ]
[FragmentParser] value was added or is system-controlled
[FragmentParser] SQL: SELECT vid FROM value_origin WHERE myvalue='intellectual'
[FragmentParser] preparing dataset for lr_creator.name_originidref with
                  javatype integer and sqltype DECIMAL
[FragmentParser] content: 4
[FragmentParser] adding dataset to existing tablecache for table
                  lr_creator and orderid 1
```

Im kommenden Abschnitt ist wieder eine Besonderheit zu sehen. Das Element *Name* hat kein systemkontrolliertes Vokabular. Also muss für den Wert geprüft werden, ob er bereits in der Datenbank vorhanden ist oder ob er neu eingetragen werden muss. Im folgenden ist der Eintrag bereits vorhanden.

```
[FragmentParser] finishing element name
[FragmentParser] starting handling of content for path learningresource.creator.name
[FragmentParser] content: Fudeus, Stephan
[FragmentParser] finding metaModelObject for path learningresource.creator.name
[FragmentParser] found metaModelObject for path learningresource.creator.name
[FragmentParser] found xmltype ELEMENT.SINGLEVALUE -> handing over to handleValue()
[FragmentParser] starting handling of path learningresource.creator.name
[ ... Überprüfung von Abhängigkeiten ... ]
[FragmentParser] learningresource.creator.name is not a system-controlled value
[FragmentParser] checking, if value exists
[FragmentParser] SQL: SELECT * FROM lr_name WHERE content='Fudeus, Stephan'
[FragmentParser] value exists
[FragmentParser] preparing cache-dataset for lr_name.NAMEID
[FragmentParser] content: 1
[FragmentParser] creating new tablecache for table lr_name and orderid 1
[FragmentParser] preparing cache-dataset for lr_name.content
[FragmentParser] content: Fudeus, Stephan
[FragmentParser] adding dataset to existing tablecache for table lr_name and orderid 1
[FragmentParser] successfully tested and set value in db
[FragmentParser] value was added or is system-controlled
[FragmentParser] SQL: SELECT nameid FROM lr_name WHERE content='Fudeus, Stephan'
[FragmentParser] preparing dataset for lr_creator.nameidref with
                  javatype integer and sqltype DECIMAL
[FragmentParser] content: 1
[FragmentParser] adding dataset to existing tablecache for table
                  lr_creator and orderid 1


[FragmentParser] *** fulfilling Dependencies after Element name
[ ... Prüfen von bereits mehrfach geprüften Feldern (s.o.) ...]
[FragmentParser] looking for dependencies of table lr_creator
[FragmentParser] registering that deps are fulfilled for lr_creator
[FragmentParser] fieldname resourceidref has dependencies
[FragmentParser] resourceidref is a field we did not write
[FragmentParser] trying to get value of mb_learningresource.lrid from pre-cache
[FragmentParser] preparing dataset for lr_creator.resourceidref with
```

```
                javatype integer and sqltype DECIMAL
[FragmentParser] content: 10002
[FragmentParser] adding dataset to existing tablecache for table
                lr_creator and orderid 1
[FragmentParser] finished dependencies for resourceidref
[FragmentParser] fieldname nameidref has dependencies
[FragmentParser] finished dependencies for nameidref
[FragmentParser] fieldname name_originidref has dependencies
[FragmentParser] finished dependencies for name_originidref
[FragmentParser] finished dependencies of table lr_creator
[FragmentParser] looking for dependencies of table mb_learningresource
[FragmentParser] mb_learningresource has no dependencies
[FragmentParser] done finishing element name


[FragmentParser] finishing element creator
[FragmentParser] starting handling of content for path learningresource.creator
[FragmentParser] content:
[FragmentParser] finding metaModelObject for path learningresource.creator
[FragmentParser] found metaModelObject for path learningresource.creator
[FragmentParser] found xmltype ELEMENT.MULTIPLEVALUE.ORDERED -> handing over
                to handleMultipleValue()
[FragmentParser] handling multiple value for path learningresource.creator,
                setting to value:
[FragmentParser] preparing dataset for lr_creator.orderid with
                javatype integer and sqltype DECIMAL
[FragmentParser] content: 1
[FragmentParser] adding dataset to existing tablecache for table
                lr_creator and orderid 1
[FragmentParser] starting handling of path learningresource.creator
[ ... Überprüfung von Abhängigkeiten ... ]


[FragmentParser] *** fulfilling Dependencies after Element creator
[FragmentParser] starting to fulfill table-dependencies
[ ... Prüfen von bereits mehrfach geprüften Feldern (s.o.) ...]
[FragmentParser] looking for dependencies of table lr_creator
[FragmentParser] fieldname resourceidref has dependencies
[FragmentParser] finished dependencies for resourceidref
[FragmentParser] fieldname nameidref has dependencies
[FragmentParser] finished dependencies for nameidref
[FragmentParser] fieldname name_originidref has dependencies
[FragmentParser] finished dependencies for name_originidref
[FragmentParser] finished dependencies of table lr_creator
[FragmentParser] looking for dependencies of table mb_learningresource
[FragmentParser] mb_learningresource has no dependencies
[FragmentParser] done finishing element creator


[FragmentParser] found element identifier
[FragmentParser] getting orderId
[FragmentParser] creating new orderId-counter
[FragmentParser] starting handling of attribute with path learningresource.identifier.origin
[FragmentParser] content: automatic
[FragmentParser] found metaModelObject for path learningresource.identifier.origin
[ ... Überprüfung von Abhängigkeiten ... ]
[FragmentParser] value was added or is system-controlled
[FragmentParser] SQL: SELECT vid FROM value_origin WHERE myvalue='automatic'
[FragmentParser] preparing dataset for lr_identifier.originidref with
                javatype integer and sqltype DECIMAL
[FragmentParser] content: 1
[FragmentParser] creating new tablecache for table lr_identifier and orderid 1
[FragmentParser] starting handling of attribute with path learningresource.identifier.URI
[FragmentParser] content: http:/www.akleon.de/test.html
[FragmentParser] finding metaModelObject for path learningresource.identifier.URI
[FragmentParser] found metaModelObject for path learningresource.identifier.URI
[FragmentParser] starting handling of path learningresource.identifier.URI
[ ... Überprüfung von Abhängigkeiten ... ]
[FragmentParser] preparing dataset for lr_identifier.uri with
                javatype java.lang.String and sqltype LVARCHAR
```

```
[FragmentParser] content: http:/www.akleon.de/test.html
[FragmentParser] adding dataset to existing tablecache for table lr_identifier
                 and orderid 1


[FragmentParser] finishing element identifier
[FragmentParser] starting handling of content for path learningresource.identifier
[FragmentParser] content:
[FragmentParser] finding metaModelObject for path learningresource.identifier
[FragmentParser] found metaModelObject for path learningresource.identifier
[FragmentParser] found xmltype ELEMENT.MULTIPLEVALUE.ORDERED -> handing over
                 to handleMultipleValue()
[FragmentParser] handling multiple value for path learningresource.identifier,
                 setting to value:
[FragmentParser] preparing dataset for lr_identifier.orderid with
                 javatype integer and sqltype DECIMAL
[FragmentParser] content: 1
[FragmentParser] adding dataset to existing tablecache for table lr_identifier
                 and orderid 1
[FragmentParser] starting handling of path learningresource.identifier
[ ... Überprüfung von Abhängigkeiten ... ]


[FragmentParser] *** fulfilling Dependencies after Element identifier
[ ... Prüfen von bereits mehrfach geprüften Feldern (s.o.) ...]
[FragmentParser] looking for dependencies of table lr_identifier
[FragmentParser] registering that deps are fulfilled for lr_identifier
[FragmentParser] fieldname identifierid has dependencies
[FragmentParser] identifierid is a field we did not write
[FragmentParser] trying to get value of mb_learningresource.lrid from pre-cache
[FragmentParser] preparing dataset for lr_identifier.identifierid with
                 javatype integer and sqltype DECIMAL
[FragmentParser] content: 10002
[FragmentParser] adding dataset to existing tablecache for table
                 lr_identifier and orderid 1
[FragmentParser] finished dependencies for identifierid
[FragmentParser] fieldname originidref has dependencies
[FragmentParser] finished dependencies for originidref
[FragmentParser] finished dependencies of table lr_identifier
[FragmentParser] done finishing element identifier


[FragmentParser] finishing element learningresource
[FragmentParser] starting handling of content for path learningresource
[FragmentParser] content:
[FragmentParser] finding metaModelObject for path learningresource
[FragmentParser] found metaModelObject for path learningresource
[FragmentParser] found xmltype VIRTUAL for path learningresource which is not handled


[FragmentParser] *** fulfilling Dependencies after Element learningresource
[ ... Prüfen von bereits mehrfach geprüften Feldern (s.o.) ...]
```

A.2.3. Abschluss FragmentParser

In diesem Abschnitt werden jetzt alle Werte, die im Cache festgehalten wurden, in die Datenbank geschrieben. Damit ist der Lauf des FRAGMENTPARSER dann auch abgeschlossen.

```
[FragmentParser] ******************************
[FragmentParser] Finishing Document
[FragmentParser] ******************************
[FragmentParser] SQL: INSERT INTO mb_learningresource (lrid) VALUES (10002)
[FragmentParser] SQL: INSERT INTO lr_title
                           (resourceidref,main,orderid,main_originidref,main_langidref)
                           VALUES (10002,'Testdokument',1,4,2)
[FragmentParser] SQL: INSERT INTO mb_lrmaintain
                           (guid,lridref)
                           VALUES ('LR0771610002',10002)
```

```
[FragmentParser] SQL: INSERT INTO lr_creator
                          (resourceidref,nameidref,orderid,vid,name_originidref)
                          VALUES (10002,1,1,80002,4)
[FragmentParser] SQL: INSERT INTO lr_identifier
                          (identifierid,uri,originidref,orderid)
                          VALUES (10002,'http:/www.akleon.de/test.html',1,1)

[FragmentParser] finished parsing metabase-document
```

A.2.4. Abschluss LRCtrl

Im letzten Abschnitt muss der LRCTRL nur noch das zugehörige Entity-Bean auflösen und dort alle Attribute entsprechend der Properties des XMLDOCUMENT bzw. anhand von Standardwerten setzen.

```
[LRCtrlEJB] Parser returned without error
[LRCtrlEJB] indexstructures prepared for guid LR0771610002
[LRCtrlEJB] Looking up lrentry
[LRCtrlEJB] resolving lrentry for guid LR0771610002
[LRCtrlEJB] Trying to set attributes
[LRCtrlEJB] setting fragment to
<learningresource guID="LR0771610002">
<title><main lang="DE" origin="intellectual">Testdokument</main></title>
<creator><name origin="intellectual">Fudeus, Stephan</name></creator>
<identifier origin="automatic" URI="http:/www.akleon.de/test.html"></identifier>
</learningresource>

[LRCtrlEJB] setting status to 1
[LRCtrlEJB] setting priority to 2
[LRCtrlEJB] setting published to true
[LRCtrlEJB] setting resourceHasChanged to false
[LRCtrlEJB] setting createdAt to Wed Oct 01 15:15:10 CEST 2003
[LRCtrlEJB] setting lastmodifiedAt to Wed Oct 01 15:15:10 CEST 2003
[LRCtrlEJB] setting createdBy
[LRCtrlEJB] setting lastModifiedBy
[LRCtrlEJB] setting remarks
```

B. Code-Dokumentation

B.1. Package metabase.xmlprocessor.beans

Package Contents *Page*

Classes

B.1.1. Classes

CLASS **FragmentParser**

The FragmentParser parses XMLFragments of the type *learningresource*, *peerReview* and *userComment*. For each element and attribute the parser looks up the target for the content in the database and either directly writes back the value or saves it for the writeBack-phase at the end the parser-session. The FragmentParser depends on the metamodel. It is meant to be called by a type-controller, i.e. LRCtrl, RevCtrl or UCCtrl. The FragmentParser relies on Tablecache, which provides a datastructure for caching data for later write-back.

DECLARATION

---

public class FragmentParser

extends Object
implements ContentHandler, ErrorHandler

CONSTRUCTORS

- `public` **FragmentParser**`( long` **idref**`, StringBuffer` **outputBuffer** `)`
  - **Usage**
    - Creates a new instance of FragmentParser. The contructor performs the necessary initialization of all caches and temporary data structures
  - **Parameters**
    - `idref` - Internal identifier within the database
    - `outputBuffer` - The modified XML-Fragment will be written back into this buffer

METHODS

- `public void` **characters**`( char[]` **values**`, int` **offset**`, int` **length** `)`
  - **Usage**
    - SAX-Callback-Handler for the characters-event. The content is appended to the output-buffer. XML-special charcters are replaced.

- `public void` **endDocument**`( )`
  - **Usage**

        * SAX-Callback-Handler for the endDocument-event. When the end of the document is reached, all cached database-values will be written to the darabase. The database-connection is closed.

- `public void` **endElement**`( String` **uri**`, String` **lName**`, String` **qName** `)`
  - **Usage**
    * SAX-Callback-Handler for the endElement-event. When an XML-element is finished, all dependencies are checked recursive, this means a certain overhead. A special routine for subject-handling is called.

- `public void` **endPrefixMapping**`( String` **prefix** `)`
  - **Usage**
    * This SAX-Event is not used here.

- `public void` **error**`( SAXParseException` **err** `)`
  - **Usage**
    * SAX-Callback-Handler for the error-event. A error provides log-output and will be thrown to the caller.

- `public void` **fatalError**`( SAXParseException` **err** `)`
  - **Usage**
    * SAX-Callback-Handler for the fatalError-event. A fatal error provides log-output and will be thrown to the caller.

- `public void` **ignorableWhitespace**`( char[]` **values**`, int` **offset**`, int` **length** `)`
  - **Usage**
    * This SAX-Event is not used here.

- `public void` **processingInstruction**`( String` **target**`, String` **data** `)`
  - **Usage**
    * This SAX-Event is not used here.

- `public void` **setDocumentLocator**`( Locator` **locator** `)`
  - **Usage**
    * The document locator will be set.

- `public void` **skippedEntity**`( String` **name** `)`
  - **Usage**
    * This SAX-Event is not used here.

- `public void` **startDocument**`( )`
  - **Usage**
    - SAX-Callback-Handler for the startDocument-event. A DB-connection is opened and configured.

- `public void` **startElement**`( String` **uri**`, String` **lName**`, String` **qName**`, Attributes` **attributes** `)`
  - **Usage**
    - SAX-Callbach-Handler for the startElement-event. New order-ids are assigned or incremented. Special handler will be called for non-generic elements i.e. *subject*, *creator*. XML-special characters will be escaped.

- `public void` **startPrefixMapping**`( String` **prefix**`, String` **uri** `)`
  - **Usage**
    - This SAX-Event is not used here.

- `public void` **warning**`( SAXParseException` **err** `)`
  - **Usage**
    - SAX-Callback-Handler for the warning-event. A warning provides log-output and will NOT be thrown to the caller.

CLASS **Tablecache**

The tablecache is a caching structure. It stores write-back values for db-tables. Tables are identified by tablename and id, so several tables can have the same name (or multiple tuples can be cached for write-back in the same table). The interface is designed to exchange whole tuples, not single attributes in a tuple.

DECLARATION

---

public class Tablecache **extends** Object

CONSTRUCTORS

- `public` **Tablecache**`( )`
  - **Usage**
    - Creates a new instance of Tablecache.

- `public void` **addTable**`( String` **tablename**`, Long` **index**`, HashMap` **tablecontent** `)`
  - **Usage**
    - Add a table to the tablecache. If a table exists for the given name an index, it will be overwritten. If the given content is *null* no changes are made.
  - **Parameters**
    - `tablename` - Name of the table to be cached
    - `index` - Index of the table to be cached
    - `tablecontent` - Content of the table to be cached

- `public boolean` **containsTable**`( String` **tablename**`, Long` **index** `)`
  - **Usage**
    - Check if a table with the given name and index already exists in cache.
  - **Parameters**
    - `tablename` - Name of the table to be checked
    - `index` - Index of the table to be checked
  - **Returns** - *true*, if a table with the given name and index is known

- `public HashMap` **getLastTableByName**`( String` **tablename** `)`
  - **Usage**
    - Resolve the tablecache with the given name which has the highest index.
  - **Parameters**
    - `tablename` - Name of the requested table.
  - **Returns** - Single HashMap with all attributes of the tuple or *null* if there is no table with the given name.

- `public HashMap` **getTableByNameAndIndex**`( String` **tablename**`, Long` **index** `)`
  - **Usage**
    - Resolve a single tablecache by tablename and index.
  - **Parameters**
    - `tablename` - Name of the requested table
    - `index` - Index of the requested table
  - **Returns** - Single HashMap with all attributes of the tuple or *null* if there is no table with the given name and index cached.

- `public HashMap` **getTableMapByName**`( String` **tablename** `)`
  - **Usage**
    - Resolve a set of tables by tablename. The returned structure contains tables for all indices
  - **Parameters**

    * `tablename` - Name of the set of tables to be requested
  - **Returns** - Set of tables, sorted by index

- `public Set` **keySet**`( )`
  - **Usage**
    * Resolve all names of cached tables.
  - **Returns** - A set of tablenames for which a cache exists

CLASS **XMLDocument**

The XMLDocument is the datastructure which is used to exchange metabase-documents with administrative properties.

The following properties are known:

- guid (String)
- createdat (Date)
- createdby (UserInfo)
- lastmodifiedat (Date)
- lastmodifiedby (UserInfo)
- resourcehaschanged (Boolean)
- published (Boolean)
- status (Long)
- priority (Long)
- remarks (String)

DECLARATION

---

public class XMLDocument **extends** Object **implements** Serializable

CONSTRUCTORS

- `public` **XMLDocument**`( )`
  - **Usage**
    * Create an empty XMLDocument

- `public` **XMLDocument**`( String` **content** `)`
  - **Usage**

        * Create a new XMLDocument with a given xml-content
    - **Parameters**
        * `content` - xml-content as String

METHODS

- `public void` **addProperty**`( String` **name**`, Object` **property** `)`
    - **Usage**
        * Add a specific property
    - **Parameters**
        * `name` - Name of the property to be added
        * `property` - Object with property-content

- `public String` **getContent**`( )`
    - **Usage**
        * Get xml-content of the document
    - **Returns** - String with xml-content

- `public Iterator` **getProperties**`( )`
    - **Usage**
        * Get list of properties
    - **Returns** - List of names for attached properties

- `public Object` **getProperty**`( String` **name** `)`
    - **Usage**
        * Get a specific property
    - **Parameters**
        * `name` - Name of the property to be fetched
    - **Returns** - wanted property

- `public void` **setContent**`( String` **newContent** `)`
    - **Usage**
        * Set content of the document
    - **Parameters**
        * `newContent` - new xml-content for the document

B.2. Package metabase.xmlprocessor.ejb.controller

Package Contents *Page*

Interfaces

Classes

B.2.1. Interfaces

INTERFACE **XMLPCtrl**

This is the interface to the main controller which is the interface between the datastore and some user applications or web-services. Every modification to searchstructures or xml-fragments originated by some human user has to be done via this controller. Insert, update and delete operations are provided with all necessary means of synchronization. See the documentation of the interface methods for more information.

DECLARATION

---

public interface XMLPCtrl **implements** EJBObject

METHODS

- `public void` **addFormat**`( String` **parentGuid**`, String` **formatGuid** `)`
  - **Usage**
    - Connect an existing learningresource to another via the "hasFormatRelation.
  - **Parameters**
    - `parentGuid` - guID of the referencing learningresource
    - `partGuid` - guID of the learningresource, which is provided in an alternative format than the other one
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public String` **addLearningresource**`( XMLDocument` **learningresource** `)`
  - **Usage**
    - Add a single learningresource without references to other documents. The learningresource has to be provided as a valid metabase-document. Provided peerReviews and userComments will be ignored. It is allowed to provide multiple learningresources, each one will be independently added to the database.
  - **Parameters**
    - `learningresource` - valid metabase-document containing a single or muliple learningresources
  - **Returns** - guID of the FIRST submitted learningresource
  - **Exceptions**

    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    * `LRCtrl` (in B.3.1, page 59)

- `public void` **addPart**`( String` **parentGuid**`, String` **partGuid** `)`
  - **Usage**
    * Connect an existing learningresource to another via the "hasPartRelation.
  - **Parameters**
    * `parentGuid` - guID of the referencing learningresource
    * `partGuid` - guID of the learningresource, which is part of the other one
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public String` **addPeerreview**`( String` **lrGuid**`, XMLDocument` **peerreview** `)`
  - **Usage**
    * Add a single peerReview, which is provided XML and connect it to the referencing learningresource. The PeerReview has to be a valid metabase-document. Provided userComments and learningresources will be ignored. It is allowed to provide multiple peerReviews, each one will be independently connected to the learningresource provided.
  - **Parameters**
    * `lrGuid` - guID of the referencing learningresource
    * `peerreview` - valid metabase-document containing a single or multiple peerreview
  - **Returns** - guID of the FIRST submitted peerreview
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    * `RevCtrl` (in B.4.1, page 68)

- `public String` **addUsercomment**`( String` **lrGuid**`, XMLDocument` **usercomment** `)`
  - **Usage**
    * Add a UserComment, which is provied in XML and connect it to the referencing learningresource. The UserComment has to be a valid metabase-document. Provided peerReviews and learningresources will be ignored. It is allowed to provide multiple userComments, each one will be independently connected to the learningresource provided.
  - **Parameters**
    * `lrGuid` - guID of the rferencing learningresource
    * `usercomment` - valid metabase-document containing a single or multiple usercomments

- **Returns** - guID of the FIRST submitted usercomment
- **Exceptions**
  - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
- **See Also**
  - `UCCtrl` (in B.5.1, page 73)

- `public void` **addVersion**`( String` **parentGuid**`, String` **versionGuid** `)`
  - **Usage**
    - Connect an existing learningresource to another via the "hasVersionRelation.
  - **Parameters**
    - `parentGuid` - guID of the referencing learningresource
    - `partGuid` - guID of the learningresource, which is a version of the other one
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **deleteDocument**`( String` **guid** `)`
  - **Usage**
    - Delete a single document (*learningresource*, *peerreview* or *usercomment*).
  - **Parameters**
    - `guid` - Identifier of the document which is to be deleted
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    - `LRCtrl` (in B.3.1, page 59)
    - `RevCtrl` (in B.4.1, page 68)
    - `UCCtrl` (in B.5.1, page 73)

- `public XMLDocument` **getDocument**`( String` **guid** `)`
  - **Usage**
    - In the case of XML-based processing no java object is created but the guid is known. So it should be possible to retrieve a document by its GUID.
  - **Parameters**
    - `guid` - GUID of the document
  - **Returns** - XML-based representation of a MetaAkad learningresource, peerreview or usercomment
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    - `XMLDocument` (in B.1.1, page 43)

- `public XMLDocument` **getDocument**`( String` **guid,** `long` **depth** `)`
  - **Usage**
    - In case of XML-based processing no java object is created but the guid is known. So it should be possible to retrieve a document by its GUID.
      - In the case of 0 only the document itself is retrieved (same as without depth),
      - If depth is greater than 0 dependend data is also retrieved,
      - in the case of -1 all dependend learning resources are integrated.
  - **Parameters**
    - `guid` - GUID of a learning resource
    - `depth` - depth of following dependencies
  - **Returns** - XML-based representation of a MetaAkad learning resource
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    - `XMLDocument (` in B.1.1, page 43)

- `public XMLDocument` **getDocumentWithReviews**`( String` **guid,** `long` **depth** `)`
  - **Usage**
    - In case of XML-based processing no java object is created but the guid is known. So it should be possible to retrieve a document by its GUID. Reviews are also included, i.e. PeerReviews and UserComments.
      - In the case of 0 only the document itself is retrieved (same as without depth),
      - If depth is greater than 0 dependend data is also retrieved,
      - in the case of -1 all dependend learning resources are integrated.
  - **Parameters**
    - `guid` - GUID of the learning resource
    - `depth` - depth of following dependencies
  - **Returns** - XML-based representation of a MetaAkad learning resource
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    - `XMLDocument (` in B.1.1, page 43)

- `public void` **insertDocument**`( XMLDocument` **document** `)`
  - **Usage**
    - Insert a multi-document, i.e. a xml-document with both learningresources, peerRevies and userComments
  - **Parameters**
    - `document` - Datastructure of the document to be inserted
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- **See Also**
  - `LRCtrl` (in B.3.1, page 59)
  - `RevCtrl` (in B.4.1, page 68)
  - `UCCtrl` (in B.5.1, page 73)

- `public void` **updateDocument**`( XMLDocument` **document** `)`
  - **Usage**
    - Update a single document (*learningresource*, *peerreview* or *usercomment*) whose guID has to be provided as property (guid) of the given XMLDocument.
  - **Parameters**
    - `document` - Datastructure of the document to be updated
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    - `LRCtrl` (in B.3.1, page 59)
    - `RevCtrl` (in B.4.1, page 68)
    - `UCCtrl` (in B.5.1, page 73)

INTERFACE **XMLPCtrlHome**

This is the home interface for the main access-controller to the xmlprocessor. Each and every interaction by a userspace client with the xmlprocessor has to use the XMLProcessorController (XMLPCtrl).

DECLARATION

public interface XMLPCtrlHome **implements** EJBHome

METHODS

- `public XMLPCtrl` **create**`( )`
  - **Usage**
    - Resolve an instance of the XMLPCtrl-Session-Bean. If none is cached, a new one is generated.
  - **Returns** - An instance of an XMLPCtrl

B.2.2. Classes

CLASS **ParserContentHandler**

This is the content- and error handler for the first step of parsing during insertion of new xml-files. XML is split up into the three types *learningresource*, *peerReview* and *userComment*. New unique identifiers will be attached and whitespace will be eliminated. This handler should be called by the XMLPController.

DECLARATION

---

public class ParserContentHandler

extends Object
implements ContentHandler, ErrorHandler

CONSTRUCTORS

- `public` **ParserContentHandler**`( List` **learningresources**`, List` **peerreviews**`, List` **usercomments**`, String` **forceGuid** `)`

METHODS

- `public void` **characters**`( char[]` **values**`, int` **start**`, int` **length** `)`
  - **Usage**
    - SAX-Callback-Handler for the characters-event. Here, empty whitespace and line-breaks are thrown away and reserved entities are replaced- Finally, the content is appended to the output-buffer.

- `public void` **endDocument**`( )`
  - **Usage**
    - SAX-Callback-Handler for the endOfDocument-event It is called, when the document is completely parsed

- `public void` **endElement**`( String` **uri**`, String` **lName**`, String` **qName** `)`
  - **Usage**

        * SAX-Callback-Handler for the endElement-event. This method is calles, when an xml-element ends during parsing. Content of an element will already be written by the characters-event, If one of the three main types *learningresource*, *peerReview*, or *userComment* is called, its content will be saved as a fragment.

- `public void` **endPrefixMapping**`( String` **str** `)`
  - **Usage**
    * This SAX-Event is not used here

- `public void` **error**`( SAXParseException` **err** `)`
  - **Usage**
    * SAX-Callback-Handler for the error-event. An error provides log-output and will be thrown to the caller.

- `public void` **fatalError**`( SAXParseException` **err** `)`
  - **Usage**
    * SAX-Callback-Handler for the fatalError-event. A fatal error provides log-output and will be thrown to the caller.

- `public void` **ignorableWhitespace**`( char[]` **values**`, int` **start**`, int` **length** `)`
  - **Usage**
    * This SAX-Event is not used here.

- `public void` **processingInstruction**`( String` **str**`, String` **str1** `)`
  - **Usage**
    * This SAX-Event is not used here.

- `public void` **setDocumentLocator**`( Locator` **locator** `)`
  - **Usage**
    * The document locator will be set.

- `public void` **skippedEntity**`( String` **str** `)`
  - **Usage**
    * This SAX-Event is not used here.

- `public void` **startDocument**`( )`
  - **Usage**
    * SAX-Callback-Handler for the startOfDocument-event. Nothing is done here.

- `public void` **startElement**`( String` **uri**`, String` **lName**`, String` **qName**`, Attributes` **attributes** `)`

- `public void` **startPrefixMapping**`( String` **str**`, String` **str1** `)`
  - **Usage**
    - This SAX-Event is not used here

- `public void` **warning**`( SAXParseException` **err** `)`
  - **Usage**
    - SAX-Callback-Handler for the warning-event. A warning provides log-output and will NOT be thrown to the caller.

CLASS **XMLPCtrlEJB**

This is an implementation of the the main controller which is the interface between the database and some user applications or web-services. Every modification to searchstructures or xml-fragments originated by some human user has to be done via this controller. Insert, update and delete operations are provided with alle necessary means of synchronization. The provided XML-code is validated in this stage of proessing.

See the documentation of the interface methods for more information. This is an implementation using a stateless session bean.

DECLARATION

public class XMLPCtrlEJB **extends** Object **implements** SessionBean

CONSTRUCTORS

- `public` **XMLPCtrlEJB**`( )`

METHODS

- `public void` **addFormat**`( String` **parentGuid**`, String` **formatGuid** `)`
  - **Usage**
    - Link two learningresource to each other via the hasFormat-relation.

  - **Parameters**
    * `parentGuid` - Identifier of the referencing document (the original document)
    * `formatGuid` - Identifier of the referenced document (the formatted document)
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public String` **addLearningresource**`( XMLDocument` **document** `)`
  - **Usage**
    * Add one single or even multiple learningresources to the database. All documents which aren't learningresource will not be processed but thrown away. For each learningresource the matching method of the LRCtrl is called. Properties of the XMLDocument are kept for all learningresources.
  - **Parameters**
    * `document` - Datastructure with the learningresource(s) to be added
  - **Returns** - guID of the FIRST submitted document
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    * `LRCtrl` (in B.3.1, page 59)
    * `RevCtrl` (in B.4.1, page 68)
    * `UCCtrl` (in B.5.1, page 73)

- `public void` **addPart**`( String` **parentGuid**`, String` **partGuid** `)`
  - **Usage**
    * Link two learningresource to each other via the hasPart-relation.
  - **Parameters**
    * `parentGuid` - Identifier of the referencing document (the complete document)
    * `partGuid` - Identifier of the referenced document (the part)
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public String` **addPeerreview**`( String` **parentguid**`, XMLDocument` **document** `)`
  - **Usage**
    * Add one single or even multiple peerreviews to the database. All documents which aren't peerreview will not be processed but thrown away. For each peerreview the matching method of the ReviewCtrl is called. Properties of the XMLDocument are kept for all peerreviews.
  - **Parameters**
    * `parentguid` - guId of the learningresource this peerreview copes with

    * `document` - Datastructure with the peerreview(s) to be added
  - **Returns** - guID of the FIRST submitted peerreview
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    * `LRCtrl` (in B.3.1, page 59)
    * `RevCtrl` (in B.4.1, page 68)
    * `UCCtrl` (in B.5.1, page 73)

- `public String` **addUsercomment**`( String` **parentguid**`, XMLDocument` **document** `)`
  - **Usage**
    * Add one single or even multiple usercomments to the database. All documents which aren't usercomment will not be processed but thrown away. For each usercomment the matching method of the UCCtrl is called. Properties of the XMLDocument are kept for all usercomments.
  - **Parameters**
    * `parentguid` - guId of the learningresource this usercomment copes with
    * `document` - Datastructure with the usercomment(s) to be added
  - **Returns** - guID of the FIRST submitted usercomment
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    * `LRCtrl` (in B.3.1, page 59)
    * `RevCtrl` (in B.4.1, page 68)
    * `UCCtrl` (in B.5.1, page 73)

- `public void` **addVersion**`( String` **parentGuid**`, String` **versionGuid** `)`
  - **Usage**
    * Link two learningresource to each other via the hasVersion-relation.
  - **Parameters**
    * `parentGuid` - Identifier of the referencing document (the original)
    * `versionGuid` - Identifier of the referenced document (the version)
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **deleteDocument**`( String` **guid** `)`
  - **Usage**
    * Delete a single document (learningresource, peerreview or usercomment). The request is directly directed to the controller-in-charge.
  - **Parameters**
    * `guid` - Identifier of the document which is to be deleted
  - **Exceptions**

        * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
    - **See Also**
        * `LRCtrl` (in B.3.1, page 59)
        * `RevCtrl` (in B.4.1, page 68)
        * `UCCtrl` (in B.5.1, page 73)

- `public void` **ejbActivate**`( )`

- `public void` **ejbCreate**`( )`

- `public void` **ejbPassivate**`( )`

- `public void` **ejbRemove**`( )`

- `public XMLDocument` **getDocument**`( String` **guid** `)`
    - **Usage**
        * Get exactly one ldocument. This document can be either of type *learningresource*, *peerReview* or *usercomment*. The request is directly directed to the controller-in-charge.
    - **Parameters**
        * `guid` - Identifier of the requested learningresource
    - **Exceptions**
        * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
    - **See Also**
        * `LRCtrl` (in B.3.1, page 59)
        * `RevCtrl` (in B.4.1, page 68)
        * `UCCtrl` (in B.5.1, page 73)

- `public XMLDocument` **getDocument**`( String` **guid**`, long` **depth** `)`
    - **Usage**
        * Get a learningresource-document with all depending learningresources (parts, versions, ...) This call is recursive to the given depth. The request is directly directed to the controller-in-charge.
    - **Parameters**
        * `guid` - Identifier of the requested learningresource
        * `depth` - Number of steps of recursion made while fetching depending learningresources
    - **Exceptions**
        * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
    - **See Also**
        * `LRCtrl` (in B.3.1, page 59)

- `public XMLDocument` **getDocumentWithReviews**`( String` **guid**`, long` **depth** `)`
  - **Usage**
    - Get a learningresource-document with all dependencies (reviews, comments, parts, versions, ...) This call is recursive to the given depth. The request is directly directed to the controller-in-charge.
  - **Parameters**
    - `guid` - Identifier of the requested learningresource
    - `depth` - Number of steps of recursion made while fetching depending documents
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    - `LRCtrl` (in B.3.1, page 59)

- `public void` **insertDocument**`( XMLDocument` **document** `)`
  - **Usage**
    - Insert a multi-document i.e. an XML-Document which contains more than one type of xml-fragment. The given XML-Document is split into its parts and given to the specific controller-in-charge. Properties of the submitted XMLDocument will be applied to each fragment.
  - **Parameters**
    - `document` - Structure with the XML-for several fragments

- `public void` **setSessionContext**`( SessionContext` **context** `)`

- `public void` **unsetEntityContext**`( )`

- `public void` **updateDocument**`( XMLDocument` **document** `)`
  - **Usage**
    - Update a single document (learningresource, peerreview or usercomment) whose guID has to be provided as property (guid) of the given XMLDocument. The request is directly directed to the controller-in-charge.
  - **Parameters**
    - `document` - Datastructure of the document to be updated
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
  - **See Also**
    - `LRCtrl` (in B.3.1, page 59)
    - `RevCtrl` (in B.4.1, page 68)
    - `UCCtrl` (in B.5.1, page 73)

B.3. Package metabase.xmlprocessor.ejb.learningresource.controller

Package Contents *Page*

Interfaces

Classes

B.3.1. Interfaces

INTERFACE **LRCtrl**

This bean controls the mapping from internal datastore to the external XML-based representation. Each manual (user-based) modification of learningresources has to be done via this controller.

DECLARATION

public interface LRCtrl **implements** EJBObject

METHODS

- `public void` **addFormat**`( String` **parentGuid**`, String` **formatGuid** `)`
  - **Usage**
    - Link two learningresource to each other via the hasFormat-relation.
  - **Parameters**
    - `parentGuid` - Identifier of the referencing document (the original document)
    - `formatGuid` - Identifier of the referenced document (the formatted document)
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **addPart**`( String` **parentGuid**`, String` **partGuid** `)`
  - **Usage**
    - Link two learningresource to each other via the hasPart-relation.
  - **Parameters**
    - `parentGuid` - Identifier of the referencing document (the complete document)
    - `partGuid` - Identifier of the referenced document (the part)
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **addVersion**`( String` **parentGuid**`, String` **versionGuid** `)`
  - **Usage**
    - Link two learningresource to each other via the hasVersion-relation.

- **Parameters**
  - `parentGuid` - Identifier of the referencing document (the original)
  - `versionGuid` - Identifier of the referenced document (the version)
- **Exceptions**
  - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **deleteDocument**`( String` **guid** `)`

  - **Usage**
    - Delete a learningresource from the datastore. All references to other documents will be deleted.
  - **Parameters**
    - `guid` - guID of the learningresource to be deleted

- `public XMLDocument` **getDocument**`( String` **guid** `)`

  - **Usage**
    - In the case of XML-based processing no java object is created but the guid is known. So it should be possible to retrieve a document by its GUID.
  - **Parameters**
    - `guid` - Identifier of the learningresource
  - **Returns** - XML-based representation of a MetaAkad learning resource
  - **See Also**
    - `XMLDocument` (in B.1.1, page 43)

- `public XMLDocument` **getDocument**`( String` **guid,** `long` **depth** `)`

  - **Usage**
    - In case of XML-based processing no java object is created but the guid is known. So it should be possible to retrieve a document by its GUID.
      - In the case of 0 only the document itself is retrieved (same as without depth),
      - If depth is greater than 0 dependend data is also retrieved,
      - in the case of -1 all dependend learning resources are integrated.
  - **Parameters**
    - `guid` - GUID of the learning resource
    - `depth` - depth of following dependencies
  - **Returns** - XML-based representation of a MetaAkad learning resource
  - **See Also**
    - `XMLDocument` (in B.1.1, page 43)

- `public XMLDocument` **getDocumentWithReviews**`( String` **guid,** `long` **depth** `)`

  - **Usage**

        * In case of XML-based processing no java object is created but the guid is known. So it should be possible to retrieve a document by its GUID. Reviews are also included, i.e. PeerReviews and UserComments.
            - In the case of 0 only the document itself is retrieved (same as without depth),
            - If depth is greater than 0 dependend data is also retrieved,
            - in the case of -1 all dependend learning resources are integrated.
    - **Parameters**
        * `guid` - GUID of the learning resource
        * `depth` - depth of following dependencies
    - **Returns** - XML-based representation of a MetaAkad learning resource
    - **See Also**
        * `XMLDocument` (in B.1.1, page 43)

- `public void` **insertDocument**`( XMLDocument` **document** `)`

    - **Usage**
        * Insert a new document into the datastore.
    - **Parameters**
        * `document` - Datastructure with the learningresource to be inserted. The guID of the learningresource has to be provided as property of the XML-Document.

- `public void` **updateDocument**`( XMLDocument` **document** `)`

    - **Usage**
        * Update an existing learningresource in the datastore.
    - **Parameters**
        * `document` - Datastructure with the updated version of the learningresource. The guID of the learningresource to be updated has to be provided as property of the XMLDocument.

INTERFACE **LRCtrlHome**

The home of the stateless Controller Bean to build a facade for accessing an XML document

DECLARATION

---

public interface LRCtrlHome **implements** EJBHome

- `public LRCtrl` **create**`( )`

B.3.2. Classes

CLASS **LRCtrlEJB**

An implementation of the Learningresourcecontroller remote interface. Each and every modifcation of learninresource-data is controlled by an instance of this type.

DECLARATION

public class LRCtrlEJB

extends Object
implements SessionBean

CONSTRUCTORS

- `public` **LRCtrlEJB**`( )`

METHODS

- `public void` **addFormat**`( String` **parentGuid**`, String` **formatGuid** `)`
  - **Usage**
    - Link two learningresource to each other via the hasFormat-relation.
  - **Parameters**
    - `parentGuid` - Identifier of the referencing document (the original document)
    - `formatGuid` - Identifier of the referenced document (the formatted document)
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **addPart**`( String` **parentGuid**`, String` **partGuid** `)`
  - **Usage**
    - Link two learningresource to each other via the hasPart-relation.
  - **Parameters**
    - `parentGuid` - Identifier of the referencing document (the complete document)
    - `partGuid` - Identifier of the referenced document (the part)
  - **Exceptions**

        * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **addVersion**`( String` **parentGuid**`, String` **versionGuid** `)`
  - **Usage**
    * Link two learningresource to each other via the hasVersion-relation.
  - **Parameters**
    * `parentGuid` - Identifier of the referencing document (the original)
    * `versionGuid` - Identifier of the referenced document (the version)
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **deleteDocument**`( String` **guid** `)`
  - **Usage**
    * Deletes a learningresource from the system. Both searchstructures, fragment and properties are deleted. Every URI (declared as ïdentifier", not äccessPage") is added to the uri-blacklist
  - **Parameters**
    * `guid` - guID of the document to be deleted
  - **Returns** - RemoteException Any fatal error is reported as RemoteException
  - **See Also**
    * `metabase.metadata.ejb.blacklist.Blacklist`

- `public void` **deleteFormat**`( String` **parentGuid**`, String` **formatGuid** `)`
  - **Usage**
    * Delete the link between two learningresources linked via the hasFormat-relation
  - **Parameters**
    * `parentGuid` - guID of the referencing learningresource which is to be modified
    * `formatGuid` - guID of the referenced learningresource, which is to be deleted

- `public void` **deletePart**`( String` **parentGuid**`, String` **partGuid** `)`
  - **Usage**
    * Delete the link between two learningresources linked via the hasPart-relation
  - **Parameters**
    * `parentGuid` - guID of the referencing learningresource which is to be modified
    * `partGuid` - guID of the referenced learningresource, which is to be deleted

- `public void` **deleteVersion**`( String` **parentGuid**`, String` **versionGuid** `)`
  - **Usage**
    - Delete the link between two learningresources linked via the hasVersion-relation
  - **Parameters**
    - `parentGuid` - guID of the referencing learningresource which is to be modified
    - `versionGuid` - guID of the referenced learningresource, which is to be deleted
- `public void` **ejbActivate**`( )`
- `public void` **ejbCreate**`( )`
- `public void` **ejbPassivate**`( )`
- `public void` **ejbRemove**`( )`
- `public XMLDocument` **getDocument**`( String` **guid** `)`
- `public XMLDocument` **getDocument**`( String` **guid**`, long` **depth** `)`
- `public XMLDocument` **getDocumentWithReviews**`( String` **guid**`, long` **depth** `)`
- `public void` **insertDocument**`( XMLDocument` **document** `)`
  - **Usage**
    - Inserts one single learningresource and takes care of all search-structures and maintenance-properties. It relies on the FragmentParser to handle the search-structures. The guID of the document must be provided as Property of the XMLDocument.
  - **Parameters**
    - `document` - datastructure which contains the xml-fragment and all other properties connected to that learningresource
  - **Exceptions**
    - `RemoteException` - In any case of error an RemoteException is thrown
  - **See Also**
    - `LREntry`
    - `FragmentParser` (in B.1.1, page 39)
- `public void` **setSessionContext**`( SessionContext` **context** `)`
- `public void` **unsetEntityContext**`( )`

- `public void` **updateDocument**`( XMLDocument` **document** `)`
  - **Usage**
    - Updates search structures, fragment and maintenance-properties for one single learningresource that is for now, the old values will be deleted and the learningresource is re-inserted. This method is not safe for documents with references, therefore in this case, an exception is thrown.
  - **Parameters**
    - `document` - datastructure which is to be inserted after the removal of the old learningresource The guID of the document to be removed and replaces is resolved from the property "guidöf the XMLDocument
  - **Exceptions**
    - `RemoteException` - In case of any fatal error, an RemoteException is thrown

B.4. Package metabase.xmlprocessor.ejb.review.controller

Package Contents *Page*

Interfaces

Classes

B.4.1. Interfaces

INTERFACE **RevCtrl**

This beans controls the mapping from internal datastore to the external XML-based representation. Each manual (user-based) modification of reviews has to be done via this controller.

DECLARATION

---

public interface RevCtrl **implements** EJBObject

METHODS

- `public void` **deleteReview**`( String` **guid** `)`
  - **Usage**
    - Delete a usercomment from the datastore.
  - **Parameters**
    - `guid` - guID of the usercomment to be deleted

- `public XMLDocument` **getReview**`( String` **guid** `)`
  - **Usage**
    - Retrieve a peerreview from the datastore.
  - **Parameters**
    - `guid` - guID of the requested review
  - **Returns** - Datastructure with the requested review

- `public void` **insertReview**`( XMLDocument` **document** `)`
  - **Usage**
    - Insert a new peerreview into the datastore.
  - **Parameters**
    - `document` - Datastructure with the peerreview to be inserted. The guID of the peerreview has to be provided as property of the XMLDocument.

INTERFACE **RevCtrlHome**

The home of the stateless Controller Bean to build a facade for accessing an XML document

DECLARATION

public interface RevCtrlHome **implements** EJBHome

METHODS

- `public RevCtrl` **create**`( )`

B.4.2. Classes

CLASS **RevCtrlEJB**

An implementation of the RevCtrl-RemoteInterface. This implementation uses a stateless session-bean.

DECLARATION

---

public class RevCtrlEJB

extends Object
implements SessionBean

CONSTRUCTORS

- `public` **RevCtrlEJB**`( )`

METHODS

- `public void` **deleteReview**`( String` **guid** `)`
  - **Usage**
    - Deletes a peerreview from the system. Both searchstructures, fragment and properties are deleted.
  - **Parameters**
    - `guid` - guID of the document to be deleted
  - **Returns** - RemoteException Any fatal error is reported as RemoteException

- `public void` **ejbActivate**`( )`

- `public void` **ejbCreate**`( )`

- `public void` **ejbPassivate**`( )`

- `public void` **ejbRemove**`( )`

- `public XMLDocument` **getReview**`( String` **guid** `)`

- `public void` **insertReview**`( XMLDocument` **document** `)`

- **Usage**
  - Inserts one single peerreview and takes care of all search-structures and maintenance-properties. It relies on the FragmentParser to handle the search-structures.
- **Parameters**
  - `document` - datastructure which contains the xml-fragment and all other properties connected to that peerreview
- **Returns** - RemoteException Any fatal error is reported as RemoteException
- **See Also**
  - `ReviewEntry`
  - `FragmentParser` (in B.1.1, page 39)

- `public void` **setSessionContext**`( SessionContext` **context** `)`

- `public void` **unsetEntityContext**`( )`

B.5. Package metabase.xmlprocessor.ejb.usercomment.controller

Package Contents *Page*

Interfaces

Classes

B.5.1. Interfaces

INTERFACE **UCCtrl**

This beans controls the mapping from internal datastore to the external XML-based representation. Each manual (user-based) modification of usercomments has to be done via this controller.

DECLARATION

---

public interface UCCtrl **implements** EJBObject

METHODS

- `public void` **deleteComment**`( String` **guid** `)`
  - **Usage**
    - Delete a usercomment from the datastore.
  - **Parameters**
    - `guid` - guID of the usercomment to be deleted

- `public XMLDocument` **getComment**`( String` **guid** `)`
  - **Usage**
    - Retrieve a usercomment from the datastore.
  - **Parameters**
    - `guid` - guID of the requested usercomment
  - **Returns** - Datastructure with the requested usercomment

- `public void` **insertComment**`( XMLDocument` **document** `)`
  - **Usage**
    - Insert a new usercomment into the datastore.
  - **Parameters**
    - `document` - Datastructure with the usercomment to be inserted. The guID of the usercomment has to be provided as property of the XMLDocument.

- `public void` **updateComment**`( XMLDocument` **document** `)`
  - **Usage**
    - Update an existing usercomment in the datastore.

– **Parameters**
  * `document` - Datastructure with the updated version of the usercomment. The guID of the usercomment to be updated has to be provided as property of the XMLDocument.

INTERFACE **UCCtrlHome**

The home of the stateless Controller Bean to build a facade for accessing an XML document

DECLARATION

---

public interface UCCtrlHome **implements** EJBHome

METHODS

- `public UCCtrl` **create**`( )`

B.5.2. Classes

CLASS **UCCtrlEJB**

An implementation of the UCCtrl-RemoteInterface. This implementation uses a stateless session-bean.

DECLARATION

public class UCCtrlEJB

extends Object
implements SessionBean

CONSTRUCTORS

- `public` **UCCtrlEJB**`( )`

METHODS

- `public void` **deleteComment**`( String` **guid** `)`
  - **Usage**
    - Deletes a usercomment from the system. Both searchstructures, fragment and properties are deleted.
  - **Parameters**
    - `guid` - guID of the document to be deleted
  - **Returns** - RemoteException Any fatal error is reported as RemoteException
- `public void` **ejbActivate**`( )`
- `public void` **ejbCreate**`( )`
- `public void` **ejbPassivate**`( )`
- `public void` **ejbRemove**`( )`
- `public XMLDocument` **getComment**`( String` **guid** `)`
- `public void` **insertComment**`( XMLDocument` **document** `)`

- **Usage**
  - Inserts one single usercomment and takes care of all search-structures and maintenance-properties. It relies on the FragmentParser to handle the search-structures.
- **Parameters**
  - `document` - datastructure which contains the xml-fragment and all other properties connected to that usercomment
- **Returns** - RemoteException Any fatal error is reported as RemoteException
- **See Also**
  - `UCEntry`
  - `FragmentParser` (in B.1.1, page 39)

- `public void` **setSessionContext**`( SessionContext` **context** `)`

- `public void` **unsetEntityContext**`( )`

- `public void` **updateComment**`( XMLDocument` **document** `)`
  - **Usage**
    - Updates search structures, fragment and maintenance-properties for one single usercomment that is for now, the old values will be deleted and the usercomment is re-inserted
  - **Parameters**
    - `document` - datastructure which is to be inserted after the removal of the old usercomment The guID of the document to be removed and replaces is resolved from the property "guidöf the XMLDocument
  - **Exceptions**
    - `RemoteException` - In case of any fatal error, an RemoteException is thrown

B.6. Package metabase.xmlprocessor.ejb.idgenerator

Package Contents *Page*

Interfaces

Classes

B.6.1. Interfaces

INTERFACE **IdGenerator**

This is the interface to the IdGenerator, which takes care of generating unique identifier for four special identifier-types.

DECLARATION

public interface IdGenerator **implements** EJBObject

METHODS

- `public long` **getNewCreatorVid** `( )`
  - **Usage**
    - Resolve an new unique id of the type CreatorVid
- `public long` **getNewLRid** `( )`
  - **Usage**
    - Resolve an new unique id of the type LRid
- `public long` **getNewPRid** `( )`
  - **Usage**
    - Resolve an new unique id of the type PRid
- `public long` **getNewUCid** `( )`
  - **Usage**
    - Resolve an new unique id of the type UCid
- `public void` **init** `( )`
  - **Usage**
    - Initialize the IdGenerator before requsting any id

INTERFACE **IdGeneratorHome**

This is the home interface for the IdGenerator. IdGeneration has to be done via this controller.

DECLARATION

public interface IdGeneratorHome

implements EJBHome

METHODS

- `public IdGenerator` **create**`( )`
  - **Usage**
    - Resolve an instance of an IdGenerator. If there is no instance an new one will be created.
  - **Returns** - An instance of an IdGenerator

B.6.2. Classes

CLASS **IdGeneratorEJB**

This is an implementation of an IdGenerator using a stateless session bean with support of an underlying database-system. The generator has to be initialized before usage.

DECLARATION

public class IdGeneratorEJB

extends Object
implements SessionBean

CONSTRUCTORS

- `public` **IdGeneratorEJB**`( )`

METHODS

- `public void` **ejbActivate**`( )`
- `public void` **ejbCreate**`( )`
- `public void` **ejbPassivate**`( )`
- `public void` **ejbRemove**`( )`
- `public long` **getNewCreatorVid**`( )`
  - **Usage**
    - Resolve a new unique CreatorVid
  - **Returns** - Unique CreatorVid
  - **Exceptions**
    - `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown
- `public long` **getNewLRid**`( )`
  - **Usage**
    - Resolve a new unique LRid

  - **Returns** - Unique lrid
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public long` **getNewPRid**`( )`
  - **Usage**
    * Resolve a new unique PRid
  - **Returns** - Unique prid
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public long` **getNewUCid**`( )`
  - **Usage**
    * Resolve a new unique UCid
  - **Returns** - Unique ucid
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while processing this request, a RemoteException is thrown

- `public void` **init**`( )`
  - **Usage**
    * Initializer for the IdGenerator. The following documentation describes the intended sense of this method, but the applicationserver spoils this because tables locked while this connection is open cannot be used :-/, even in the same transactional context. The initializer is now called for every resolver-method.

      It HAS to be called before usage for queries are resolved in this step. This method reduces overhead-code which has not to be executed every callback.
  - **Exceptions**
    * `RemoteException` - If any fatal error occurs while initializing, a RemoteException is thrown

- `public void` **setSessionContext**`( SessionContext` **sessionContext** `)`