UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Verwaltung von ODRL-Lizenzen im Toolkit für URM

Studienarbeit

vorgelegt von

Rainer Weißenfels

Betreuer: Prof. Dr. Rüdiger Grimm
Institut für Wirtschafts- und Verwaltungsinformatik

Dipl.-Ing. Helge Hundacker
Institut für Wirtschafts- und Verwaltungsinformatik

Dipl.-Informatiker Daniel Pähler
Institut für Wirtschafts- und Verwaltungsinformatik

Koblenz, im Mai 2011

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Seit Beginn des World Wide Web hat sich die Erzeugung und Verteilung digitaler Güter (digital assets) entschieden verändert. Zur Erzeugung, Bearbeitung, Verteilung und Konsumierung bedarf es heute nicht mehr spezieller physischer Gerätschaften. Dadurch hat sich die Geschwindigkeit in der Medien generiert und transportiert werden enorm gesteigert. Auch die Möglichkeiten der Kooperation waren dadurch einem Wandel unterlegen, beziehungsweise wurden mancherorts erst möglich gemacht. (Pic05, S. 7)

Die Nutzung des Internets ermöglichte zwar die Loslösung digitaler Güter von ihren physischen Trägermedien, die Bestimmungen des Urheberrechts gelten jedoch weiterhin. Dies führt gerade bei juristisch weniger erfahrenen Nutzern zu Unsicherheit darüber, wie ein konkretes digitales Gut genutzt werden darf. Andererseits wird von vielen Nutzern das gewohnte Tauschen von Medien auch auf das digitale Umfeld übertragen. Die Urheberrechtsverletzungen die zuvor im privaten Umfeld im kleinen Rahmen statt fanden, geschehen nun global und für alle sichtbar.

Da diese Form des Tausches das primäre Geschäftsmodell der Verwerter gefährdet, wird versucht, die Nutzung digitaler Güter einzuschränken beziehungsweise für nicht berechtigte Nutzer zu unterbinden. Dies geschah und geschieht unter anderem mit Verfahren der digitalen Rechte-Verwaltung (Digital Rights Management - DRM). Diese Verfahren sind

unter Nutzern bisweilen umstritten oder werden sogar offen abgelehnt, da sie die Nutzung digitaler Güter im Vergleich zum physischen Pendant erschweren können (siehe Abschnitt 2.1). Zudem erwiesen sich viele dieser Verfahren als nicht sicher, sodass die verwendeten Verschlüsselungsverfahren gebrochen wurden.

Mit einer „Nutzungsrechte-Verwaltung“ (Usage Rights Management - URM) soll DRM im Kernprinzip zwar erhalten bleiben. Die praktische Umsetzung soll aber in eine andere Richtung vorstoßen.

Der Nutzer bekommt die volle Kontrolle über die digitalen Güter (ohne die restriktiven Maßnahmen klassischer DRM-Umsetzungen), aber auch wieder die volle Verantwortung. Unterstützt wird er dabei von Software, die ihn über die rechtlichen Möglichkeiten informiert und auf Wunsch des Nutzers auch software-technische Schranken in der Benutzung setzt, ähnlich der Rechtedurchsetzung (Enforcement) bei klassischen DRM-Systemen. URM nutzt dabei die offene Rechtedefinitionssprache ODRL.

Die vorliegende Studienarbeit ist Teil des URM-Projektes der Forschungsgruppe IT-Risk-Management, welches wiederum Teil des SOAVIWA-Projektes ist. Ziel der Studienarbeit ist es eine Java-Klasse zu entwickeln, mit der in ODRL verfasste Lizenzen als Java-Objekte abgebildet werden. Weitere zu entwickelnde Komponenten sollen diese Objekte verwalten und das Modifizieren und Erzeugen neuer Objekte zulassen. Alle Komponenten sollen Bestandteil des bereits anfänglich implementierten Toolkit für URM (TURM) sein.

Kapitel 2

Grundlagen

In diesem Kapitel werden einige Grundlagen erläutert, auf denen diese Studienarbeit aufbaut und die für das Verständnis hilfreich sind.

Zunächst wird eine Definition und Zustandsbeschreibung von Digital Rights Management gegeben (Abschnitt 2.1). Danach wird die Rechtedefinitionssprache ODRL in der Entwurfs-Version 2 vorgestellt und anhand von einem Beispiel erläutert (Abschnitt 2.2). Zum Schluss wird das Projekt Usage Rights Management vorgestellt (Abschnitt 2.3). Dabei wird auf das TURM-Framework und voran gegangene Arbeiten eingegangen (Abschnitt 2.4).

2.1 DRM

Ianella (Ian08) beschreibt Digital Rights Management (DRM) folgendermaßen:

> „Digital Rights Management (DRM) involves the description, layering, analysis, valuation, trading and monitoring of the rights over an enterprise's tangible and intangible assets. DRM covers the digital management of rights - be they rights in a physical manifestation of a work (eg a book), or be they rights in a digital manifestation of a work (eg an ebook).“ (Ian08)

DRM ist also die digitale Verwaltung von Rechten über Gütern im weiten Sinne. Sie beschränkt sich also weder auf digitale Güter noch auf digitale Rechte über Güter (Rechte im digitalen Raum).

Softwaretechnisch manifestiert sich DRM in Digital Rights Management Systemen (DRMS). Fränkl und Karpf definieren diese als „technische Lösungen zur sicheren zugangs- und nutzungskontrollierten Distribution, Abrechnung und Verwaltung von digitalem und physischem Content“. (FK04, S. 26) Aufgaben von DRMS sind daher insbesondere Zugangskontrolle, Nutzungskontrolle, Abrechnung und der Umgang mit Rechteverletzungen. Dazu werden Rechtedefinitionssprachen, Verschlüsselungstechniken und digitale Wasserzeichen verwendet. (FK04, S. 29 ff.)

In der öffentlichen Wahrnehmung reduziert sich DRM jedoch in den meisten Fällen auf die technische Durchsetzung von Rechten, in der Regel die Einschränkung der Nutzung. Dies ist vor allem der Tatsache geschuldet, dass die meisten Menschen mit DRM in Verbindung kommen, wenn sie Medien konsumieren (Musik, Filme) oder Software nutzen. Hier waren und sind vor allem Techniken im Einsatz, die Zugang, Nutzung und Weiterverbreitung der digitalen Güter einschränken.

Die Erfahrung der Nutzer mit DRMS ist oft konfliktbehaftet. Damit DRMS ihr Ziel erreichen (die Rechtedurchsetzung), ist eine permanente Kontrolle über die jeweiligen Güter erforderlich. In der Regel werden dazu die Inhalte verschlüsselt und die Rechte beispielsweise vor dem Abspielen oder Kopieren überprüft. Dies erfordert, dass die Abspielgeräte und -software in das DRMS eingebunden sind, das heißt Inhalteanbieter und Geräte- und Softwarehersteller müssen Absprachen über die Umsetzung treffen.[1] Fehlen diese Absprachen, so kann ein Nutzer die erworbenen Medien nicht auf allen Geräten abspielen. Außerdem ist der Nutzer vom Fortbestand des DRMS auf Anbieterseite abhängig: Wird der Dienst eingestellt, werden die erworbenen Dateien unbrauchbar. (HPG09)

Daher haben es sich mittlerweile einige Organisationen zur Arbeit gemacht, über DRM und die beschriebenen Konsequenzen zu informieren.

[1] Ausgenommen hiervon ist der Fall, dass ein Unternehmen ein komplettes System anbietet wie bei *Apple itunes*

Andere Organisationen wie die Free Software Foundation[2] positionieren sich aktiv gegen DRM. Sie sehen das Konzept von DRM-Software als unvereinbar mit der Idee von freier Software und bringen das in Kampagnen wie „Defective by Design“[3] zum Ausdruck (Bro06).

2.2 ODRLv2

Die Open Digital Rights Language (ODRL) ist eine offene Rechtedefinitionssprache (*rights expression language*), und dient daher der Beschreibung von Rechten über digitale Güter (assets). Sie wird von der ODRL Initiative weiterentwickelt und liegt derzeit in der finalen Version 1.1 vor. Parallel dazu wird seit 2004 an der Version 2 gearbeitet[4]. Innerhalb von URM wird ODRL eingesetzt, um Lizenzen zu beschreiben. Diese Lizenzen wiederum beschreiben konkrete Rechte über konkrete Güter.

Für die vorliegende Studienarbeit wurde ein Entwurf der Version 2 verwendet, die sich zu dieser Zeit im fortgeschrittenen Entwicklungsstadium befand (Draft Specification vom 23. September 2009 (GI09b)). Im nachfolgenden Text und in der Implementierung wird stets auf diese Entwurfsversion Bezug genommen. Spätere Änderungen am ODRL Core Model und anderen Komponenten wurden daher nicht berücksichtigt.

2.2.1 Beschreibung

Es soll an dieser Stelle ein Überblick zu ODRL gegeben werden, die Unterschiede zwischen den beiden Versionen beleuchtet werden und die für TURM/URM gemachten Erweiterungen von der ODRL genannt werden.

Mit ODRL lassen sich Rechte über Güter in einer XML-basierten Sprache ausdrücken. Die Rechte werden damit maschinenlesbar und lassen sich von Anwendungen verarbeiten und weitergeben.

Um die Rechte an einem konkreten Gut zu definieren werden diesem

[2] http://www.fsf.org/
[3] http://www.defectivebydesign.org/
[4] ODRL Version 2.0 Working Group: http://odrl.net/2.0/

Permissions und *Prohibitions* zugewiesen. Im Rahmen dieser Studienarbeit werden Permissions und Prohibitions zusammenfassend *Right Expression*[5] genannt. Eine Right Expression besteht mindestens aus einem *Asset* (auf das sich die Right Expression bezieht), einer *Party* (der ein Recht gegeben wird; der so genannte *Assignee*) und einer *Action* (einer konkreten Handlung, die hier zugeschrieben wird). Alle genannten Entitäten (Asset, Party , Action) können und sollten dann jeweils genauer beschrieben werden.

Weiterhin ist es möglich, innerhalb einer Right Expression auch die Party zu benennen, die hier das Recht veräußert (*Assigner*, in Abgrenzung zum empfangenden *Assignee*) und die Action über ein oder mehrere *Constraints* einzuschränken. Innerhalb einer Permission können zusätzlich ein oder mehrere *Duties* (Pflichten, die vom Assignee zu erfüllen sind) benannt werden.

Die einzelnen Entitäten des ODRL Core Model sind in Abbildung 2.1 zu sehen. Im Folgenden werden sie näher beschrieben.

Rights

Die Entität *Rights* versammelt ein oder mehrere Right Expressions unter sich und beschreibt die Rechteausdrücke auf einer Meta-Ebene. Dazu identifiziert sie eine Lizenz eindeutig mit dem Attribut *uid* und benennt den Typ der Lizenz mit dem Attribut *type*. Bei dem Lizenztyp kann es sich beispielsweise um ein *Offer* (ein Angebot des Assigner), ein *Agreement* (eine Übereinkunft zwischen Assigner und Assignee) oder andere Arten von Lizenzen handeln. (GI09b, Abschnitt 4. Scenarios)

Weitere Attribute dienen zur Beschreibung der Handhabung der Lizenz und der Rechteausdrücke.[6]

[5] Der Begriff *Right Expression* bezieht sich nur auf den Kontext dieser Studienarbeit. Er ist abzugrenzen von den sogenannten *Rights Expressions*, die in den Draft Specifications des ODRL V2.0 Core Model formuliert werden. (GI09b)

[6] *conflict* beschreibt die Konfliktlösung zwischen einzelnen Right Expressions. *undefined* beschreibt das Behandeln nicht definierter Rechte. *inherit* beschreibt die Möglichkeit zur Ableitung von Rechten aus dieser Lizenz

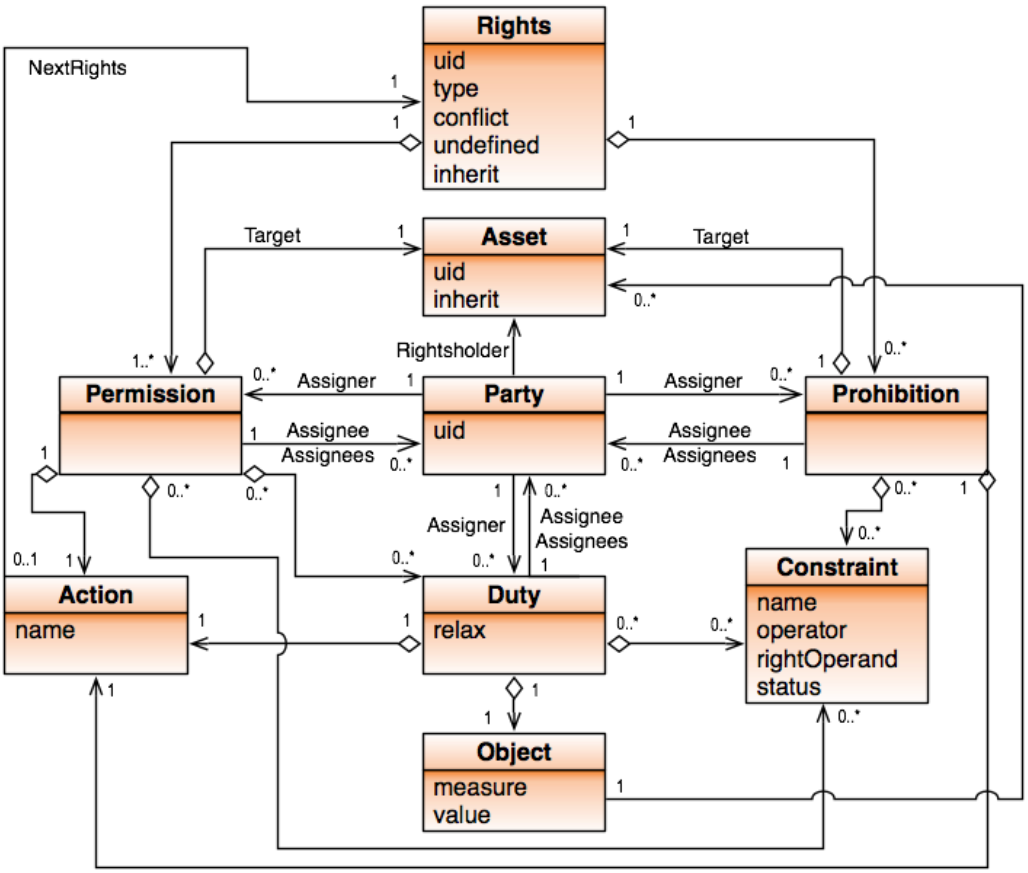

Abbildung 2.1: Core Model Version 2.0 (2009-09-23) (GI09b)

Permission

Beschreibt ein ausdrücklich gewährtes Recht. Eine Permission setzt sich aus den Entitäten Asset, Action und Party (Assignee) zusammen. Wahlweise können auch die Entitäten Party (Assigner) und Duty definiert werden.

Eine Duty beschreibt in diesem Fall, welche Pflichten (zum Beispiel eine Zahlung) vom Assignee zu erfüllen sind, damit er die unter der Permission beschriebenen Rechte erlangt.

Prohibition

Beschreibt ein ausdrücklich versagtes Recht. Genau wie eine Permission setzt sich eine Prohibition aus den Entitäten Asset, Action und Party (Assignee) zusammen. Wahlweise kann auch die Entität Party (Assigner) definiert werden. Die Entität Duty aber entfällt hier.

Asset

Beschreibt ein konkretes Asset. Dieses Asset wird über das Attribut *uid* eindeutig identifiziert. Mit dem optionalen Attribut *inherit* identifiziert man ein weiteres Asset, von dem Rechteinformationen geerbt werden.[7]

Action

Beschreibt ein konkrete Handlung, die mit einem Asset vollzogen beziehungsweise nicht vollzogen werden kann. Diese Handlung wird über das Attribut *name* identifiziert.

Constraint

Beschreibt eine konkrete Einschränkung einer Right Expression. Die Einschränkung wird mit den Attributen *name*, *operator* und *rightOperand* spezifiziert.

Beispielsweise kann die erlaubte Action „play“ mit dem Constraint „count lteq 3“ (*name* = „count“, *operator* = „lteq“ (less than or equal),

[7]Diese Funktionalität wird in späteren Entwürfen des ODRL Core Model zentraler gestaltet. In der vorliegenden Arbeit wird lediglich das Attribut implementiert ohne weitergehende Funktionalität.

rightOperand = „3“) eingeschränkt werden. Das Asset dürfte in diesem Fall maximal dreimal abgespielt werden. (GI09a)

Der aktuelle Zustand eines Constraint kann mit dem Attribut *status* als Zähler festgehalten werden.

Party

Beschreibt einen konkreten Teilnehmer, dem entweder eine Right Expression zugewiesen werden (Assignee) oder der selbst Rechte zuweist (Assigner). Der Teilnehmer wird über das Attribut *uid* eindeutig identifiziert.

Duty

Beschreibt eine Pflicht, die innerhalb einer Permission von der Party (Assignee) erfüllt werden muss. Sie wird über eine *Action* und ein *Object* definiert. Weiter können der *Duty* auch *Constraints* zugewiesen werden.

Die Duty ähnelt in ihrem Aufbau der Permission und der Prohibition. Daher kann sie eigentlich auch als Right Expression aufgefasst werden. Im Gegensatz zur Permission und Prohibition tritt sie aber nie eigenständig innerhalb einer Lizenz auf, sondern ist stets einer Permission zugeordnet. Daher wird in dieser Studienarbeit von einer Klassifizierung als Right Expression abgesehen.

Object

Beschreibt die Quantifizierung einer Duty. Dies geschieht über die Attribute *measure* (Einheit) und *value* (Wert). Dies kann beispielsweise der Geldbetrag einer Zahlung sein.

2.2.2 Beispiel

Um die Verwendung von ODRL zu verdeutlichen, wird diese hier an einer beispielhaften Agreement-Lizenz kurz aufgezeigt.

In diesem Beispiel überlässt der Teilnehmer *urn:sony:10* als Assigner dem Teilnehmer *billie:888* Rechte an dem Asset *urn:music:4545*. Dem Teil-

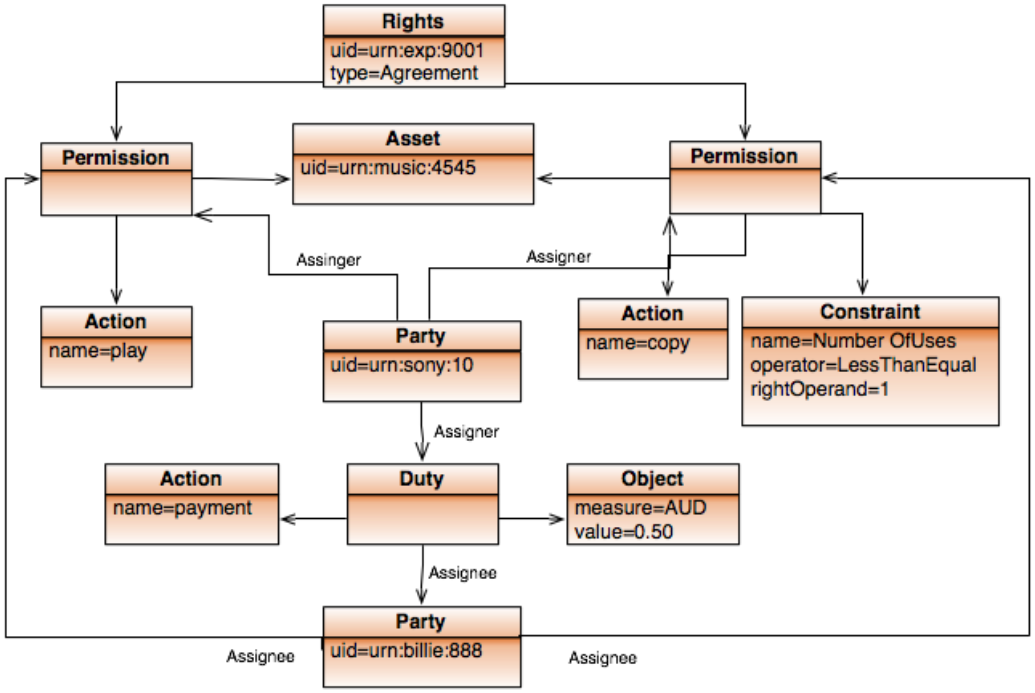

Abbildung 2.2: Schema: Beispiel eines Agreements (GI09b)

nehmer *billie:888* wird gestattet, das Asset abzuspielen und maximal eine Kopie anzufertigen. Gleichzeitig verpflichtet sich *billie:888* an den Assigner 0,50 AUD zu zahlen. Das Agreement selbst trägt die eindeutige Bezeichnung *urn:exp:9001*. Eine schematische Darstellung ist in Abbildung 2.2 zu sehen. (GI09b)

Eine vereinfachte XML-Darstellung des Agreements ist in Listing 2.1 abgebildet. Wurzelelement des ODRL/XML-Dokuments ist das <*rights*>-Element (1). Dieses umschließt die einzelnen <*permission*>-Ausdrücke (2-15, 16-27), die jeweils die Rechte für das Abspielen und das Kopieren des Assets beschreiben. Der <*duty*>-Ausdruck zur Zahlung des Geldbetrags ist nur innerhalb des ersten <*permission*>-Ausdruck vollständig definiert (11-14). Im zweiten <*permission*>-Ausdruck wird über eine Referenz Bezug zu diesem genommen (26). (Ian09)

```
<o:rights xmlns:o="http://odrl.net/2.0" type="o:agreement" uid="
    urn:rights:9001">
    <o:permission>
        <o:asset uid="urn:music:4545"/>
        <o:action name="o:play"/>
        <o:assigner>
            <o:party uid="urn:sony:10"/>
        </o:assigner>
        <o:assignee>
            <o:party uid="urn:billie:888"/>
        </o:assignee>
        <o:duty uid="d1">
            <o:action name="o:pay"/>
            <o:asset uid="urn:ubl:AUD0.50"/>
        </o:duty>
    </o:permission>
    <o:permission>
        <o:asset uid="urn:music:4545"/>
        <o:action name="o:copy"/>
        <o:constraint name="o:numberOfUses" operator="o:lteq"
            rightOperand="1"/>
        <o:assigner>
            <o:party uid="urn:sony:10"/>
        </o:assigner>
        <o:assignee>
            <o:party uid="urn:billie:888"/>
        </o:assignee>
        <o:duty uid="#d1"/>
    </o:permission>
</o:rights>
```

Listing 2.1: XML-Code eines Beispiel-Agreements (Ian09)

2.3 URM

URM (Usage Rights Management) ist ein Projekt der Arbeitsgruppe IT-Risk-Management an der Universität Koblenz und ging aus dem Projekt SOAVIWA (Service-orientierte Absicherung von virtuellen Waren) hervor.[8] Ziel des Projekts ist es, Lösungen zu entwickeln, mit denen sich Rechte an digitalen Gütern (oder auch: virtuellen Waren) abbilden und durchsetzen lassen.[9]

URM wird dabei als Teil des Digital Rights Management im weiteren Sinne verstanden, in Abgrenzung zum technischen DRM im engeren Sinne [10].

DRM im engeren Sinne setzt vor allem auf eine technische Durchsetzung der Rechte. Es wird also versucht, die Beschränkungen der analoger Güter und Datenträger auf digitale abzubilden.[11] Diese Maßnahmen sollen möglichst verhindern, dass ein Nutzer überhaupt in die Lage gelangt, die Rechte Dritter zu verletzen. (Gri05)

Beim URM hingegen stehen vor allem Information und Dokumentation von Rechten an digitalen Gütern im Vordergrund. Der Nutzer soll möglichst einfach und direkt über seine Rechte informiert werden. Restriktive Maßnahmen zur Rechtedurchsetzung können auch daran anschließen. Sie sollten im Allgemeinen aber auf der freien Entscheidung des Nutzers gründen. (HPG09)

2.4 TURM

TURM (Toolkit for URM) ist ein Java-Framework, das in der Arbeitsgruppe IT-Risk-Management entwickelt wird und die formulierten Ziele des

[8] http://www.uni-koblenz-landau.de/koblenz/fb4/institute/iwvi/aggrimm/projekte/SOAVIWA - Abruf: 06.05.2011

[9] http://www.uni-koblenz-landau.de/koblenz/fb4/institute/iwvi/aggrimm/projekte/SOAVIWA/URM - Abruf: 06.05.2011

[10] Auch „klassisches DRM“ genannt (Gri05)

[11] Dass die Beschränkungen analoger Güter durchaus umgangen werden können soll hier nicht weiter beachtet werden.

URM softwaretechnisch umsetzt. Das heißt, es ermöglicht den Nutzer sich über seine Rechte zu informieren und diese zu verwalten.

2.4.1 Voran gegangene Arbeiten

Dem Framework gingen einzelne Arbeiten voraus, in denen eigenständige Software entwickelt wurde. Das sind die ODRL-Medienbibliothek von Nico Jahn und der Private License Generator, ein Winamp Plugin von Eugen Müller.

2.4.1.1 ODRL-Medienbibliothek

Die ODRL-Medienbibliothek ist ein Programm, dass die Verwaltung von Musikdateien und zugehörigen ODRL-Lizenzen erlaubt und im Rahmen einer Studienarbeit von Nico Jahn entwickelt wurde. (Jah10)

Ähnlich wie beim Private License Generator (siehe entsprechenden Abschnitt 2.4.1.2), werden die eigentlichen Nutzdaten einer Musikdatei extrahiert und daraus ein Hash-Wert gebildet. Dieser Hash-Wert identifiziert die Musikdatei eindeutig. In einer ODRL-1.1-Lizenz wird er dann verwendet um der Musikdatei Rechteinformationen zuzuordnen.

Zusätzlich werden auch die Nutzer eindeutig anhand einer Email-Adresse identifiziert. In der Bibliothek werden dann nur jene Dateien und Lizenzen angezeigt und verarbeitet, die den aktuellen Nutzer betreffen.

Die Rechte in den Lizenzen werden ausgewertet und dem Nutzer wird durch eine farbige Kodierung der Rechte-Status jeder Musikdatei angezeigt. Dieser unterteilt sich in:

- Status unbekannt
- Wiedergabe verboten
- Wiedergabe erlaubt
- Weitergabe erlaubt

Die Mediathek wurde in einer späteren Studienarbeit von Florian Schlünß nochmal neu in Java implementiert und in das TURM Framework integriert.

2.4.1.2 Private Licence Generator

Der Private License Generator ist ein Plugin für das Medienabspielprogramm Winamp, das die Erzeugung von ODRL-1.1-Lizenzen für Musikdateien erlaubt, die zuvor von einer CD extrahiert wurden. Dazu verwendet das Plugin den Hashwert der erzeugten Musikdatei (respektive ihrer Nutzdaten), um sie eindeutig zu identifizieren. Zusätzlich werden Meta-Daten der Musik-CD und Benutzereingaben in die Lizenz geschrieben. (Mül09)

Der benutzte Algorithmus zur Berechnung des Hashwertes wurde in Java übersetzt und als Plugin in TURM eingebunden.

2.4.2 Das Framework

Das TURM-Framework wurde mit der Absicht etabliert, die voran gegangenen Arbeiten auf einer einheitlichen Plattform zusammenzuführen und diese als Ausgangspunkt für weitere Software-Entwicklungen innerhalb des URM zu nutzen.

Der Aufbau besteht im Wesentlichen aus dem TURM Core als zentrale Entität, über die weitere Komponenten eingebunden werden. Auch soll TURM das Laden von Plugins ermöglichen, was dem Nutzer und Dritt-Entwicklern Möglichkeiten zur Anpassung der Software bietet.

Für die Belange dieser Studienarbeit sind unter den Komponenten vor allem der *Configuration Manager*, der *P2P-Client* und der *MediaLicenseMapper* relevant.

Der *ConfigurationManager* speichert und verwaltet Konfigurationen (wie etwa Dateipfad-Angaben) der TURM-Komponenten über die Laufzeit des Systems hinaus. Die Einträge sind nach dem Muster *Schlüssel=Wert* in einer Datei hinterlegt. Die einzelnen Komponenten können über Methoden

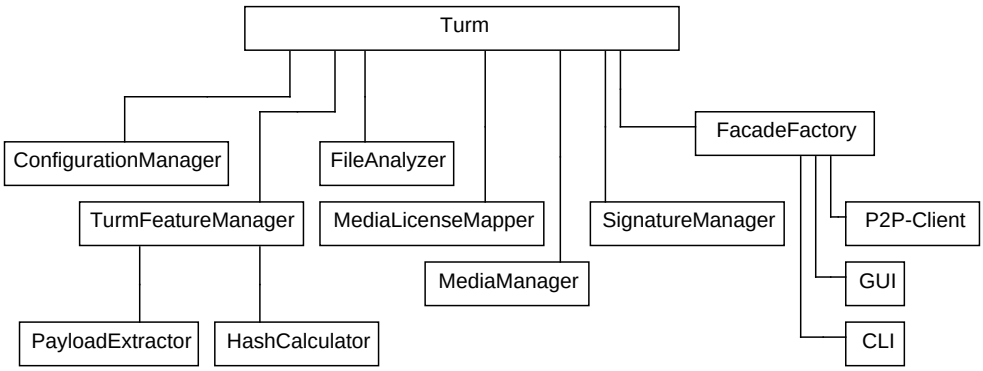

Abbildung 2.3: Komponenten von TURM (in Anlehnung an (Sch11) und (Lie10))

des ConfigurationManager dieser Einträge lesen beziehungsweise schreiben (siehe auch (Lie10, S. 86)).

Der *P2P-Client* implementiert ein P2P[12]-Filesharing-Programm, dass einen lizenzbewussten Umgang mit Dateien ermöglicht. Beim Austausch von Dateien werden die Rechteinformationen in den zugehörigen Lizenz-Dateien berücksichtigt. (Lie10) In dieser Studienarbeit sollen unter anderem Methoden entwickelt werden, die dem P2P-Client benötigte Informationen zur Verfügung stellen.

Die Komponente *MediaLicenseMapper* verknüpft Mediendateien des Nutzers mit vorhandenen Lizenzdateien. Der MediaLicenseMapper nutzt dazu Informationen des *MediaManager*, der die Mediendateien innerhalb von TURM verwaltet. Die Lizenzinformationen werden von Komponenten bereit gestellt, die es in dieser Arbeit ebenfalls zu entwickeln gilt. (Sch11, S. 21 f.)

[12]Peer-to-Peer

Kapitel 3

Implementation

In diesem Kapitel wird die entwickelte Implementierung dargestellt. Dazu werden zunächst die gestellten Anforderungen aufgezeigt (Abschnitt 3.1. Im darauf folgenden Schritt wird eine Übersicht über das entworfene Modell gegeben (Abschnitt 3.2. Zum Schluss werden die umgesetzten Klassen im Detail vorgestellt und die Interaktionen zwischen diesen verdeutlicht (Abschnitt 3.3.

3.1 Aufgabenstellung

Es sollen verschiedene Java-Klassen entwickelt werden, die es erlauben, ODRL-Lizenz-Dateien einzulesen und als Objekte abzubilden, diese Objekte zu verändern, neue Instanzen zu erstellen und diese Objekte als ODRL-Lizenz-Dateien zu schreiben. Weiterhin sollen Methoden entwickelt werden, um diese Objekte zu verwalten und anderen Klassen innerhalb des TURM Zugriff auf diese Objekte (direkt oder indirekt) zu geben.

Es soll eine Java-Klasse *LicenseManager* entwickelt und implementiert werden, die ODRL-Lizenzen verwaltet. Vorhandene Lizenzen sollen damit eingelesen und bearbeitet werden können. Neu erstellte und veränderte Lizenzen sollen damit abgespeichert werden können. Die Klasse soll Bestandteil des TURM-Framework sein und ihre Methoden anderen Komponenten zur Verfügung stellen.

Dafür soll zunächst eine Java-Klasse *ODRLLicense* entwickelt werden, die es ermöglicht, ODRL-Lizenzen als Java-Objekte zu beschreiben. Im nächsten Schritt sollen Komponenten entwickelt werden, die Konvertierungen zwischen ODRL-Lizenz-Dateien und ODRL-Lizenz-Objekten ermöglichen. Sprich: Das Importieren und Exportieren von ODRL-Lizenzen erlauben (Klassen *LicenseReader/LicenseWriter*).

Im dritten Schritt soll eine Datenbanklösung entwickelt werden, die ein einfaches Abspeichern und Wiedereinlesen der verwalteten ODRL-Lizenz-Objekte erlaubt (Klasse *DBManager*).

Bei allen Entwicklungsschritten sollen dabei auch die bereits fest gelegten Erweiterungen des ODRL-Modells innerhalb von URM berücksichtigt werden.

3.2 Entwurf

Wie in Abschnitt 3.1 erwähnt, unterteilt sich das Modell maßgeblich in fünf Klassen/Pakete. Diese sind als Komponenten in Abbildung 3.1 blau dargestellt. In der Abbildung nicht vorhanden ist das Paket *odrllicense*, in dem die Klasse ODRLLicense beschrieben wird. Die Klassen/Pakete und ihre Funktion sollen hier kurz beschrieben werden.

3.2.1 Übersicht der Klassen

Folgende Klassen werden entwickelt:

ODRLLicense/URMLicense

Die ODRL-License-Klasse bildet eine ODRL-Lizenz als Java-Objekt ab. Der Aufbau der Klasse leitet sich vom ODRL Core Model (siehe Abschnitt 2.2.1) ab. Sie beinhaltet zuoberst ein Rights-Objekt, das wiederum Listen von Permission- und Prohibition-Objekten beinhaltet usw. (Abschnitt 3.3.1).

Die URMLicense-Klasse erweitert die ODRL-License-Klasse um Eigenschaften, die spezifisch für eine URM-Lizenz sind. Das betrifft

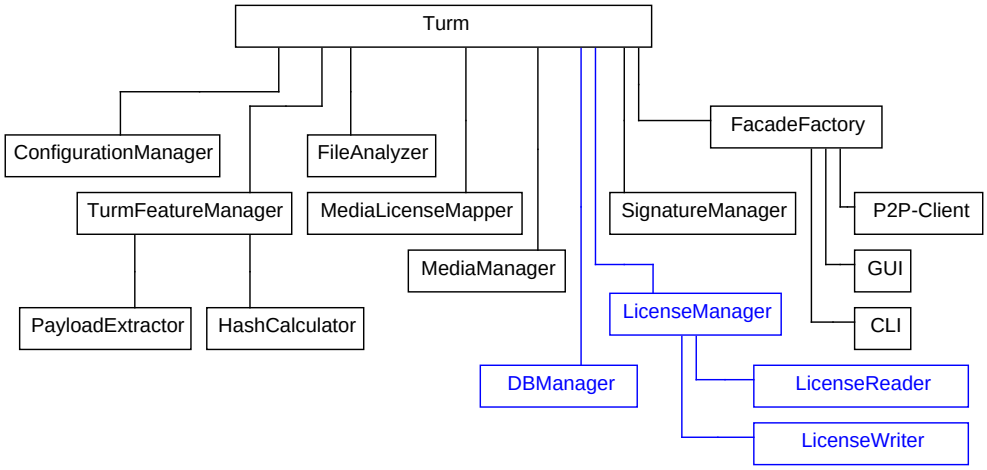

Abbildung 3.1: Komponenten von Turm inklusive LicenseManager (in Anlehnung an (Sch11) und (Lie10)). Die hier entwickelten Komponenten sind blau hinterlegt.

vor allem die Klasse Asset (mit ihrer Tochterklasse URMAsset), aber auch die Klassen Party (URMParty) und Rights (URMRights). Genauere Erläuterungen folgen in Abschnitt 3.3.1.1.

TURMLicenseManager

Die Hauptklasse TURMLicenseManager (abgekürzt: *LicenseManager*) verwaltet die ODRL-Lizenzen als Objekte vom Typ ODRLLicense. Zum Importieren der Lizenzen benutzt sie die Klasse LicenseReader (siehe ebd.). Neu erstellte oder veränderte Lizenzen werden mit Hilfe der Klasse LicenseWriter exportiert (siehe ebd.). Der LicenseManager stellt die Lizenzen und weitere Methoden anderen Komponenten bereit. Eine persistente Speicherung des Lizenz-Bestands erfolgt über die Klasse DBManager (siehe ebd.). Die Synchronisierung zwischen LicenseManager, DBManager und den Lizenz-Dateien auf der Festplatte ist ebenso Aufgabe des LicenseManager.

LicenseReader

Die LicenseReader-Klasse stellt Methoden bereit, um den Inhalt einer ODRL-Lizenzdatei im XML-Format einzulesen und daraus ein

ODRLLicense-Objekt zu erstellen. Dabei wird der Dateiinhalt zunächst mit einem DOM/XML-Parser in einen DOM-Tree[1] übertragen. Dieser wird anschließend ausgelesen, um das ODRL-License-Objekt zu erstellen.

LicenseWriter

Analog zur LicenseReader-Klasse stellt die LicenseWriter-Klasse Methoden bereit, um den Inhalt eines ODRLLicense-Objekt in eine ODRL-Lizenzdatei zu schreiben. Genau wie beim LicenseReader wird zunächst der Inhalt in einen DOM-Tree übertragen. Anschließend schreibt ein XSLT-Prozessor den Inhalt des DOM-Tree in eine ODRL-Lizenzdatei.

DBManager

Die Klasse DBManager verwaltet eine SQLite-Datenbank, in der die ODRLLicense-Objekte des LicenseManager redundant gehalten werden. Die Datenbank soll dadurch eine schnellere Suche in den Lizenzinformationen ermöglichen. Außerdem soll sie, ähnlich einer Mediathek, die Lizenzinformationen abspeichern können, um sie bei erneutem Start des LicenseManager direkt bereitstellen zu können, ohne die Lizenz-Dateien erneut einzulesen.

Der DBManager wurde in Zusammenarbeit mit Florian Schlünß entwickelt, der diesen ebenfalls im Rahmen seiner Bachelorarbeit einsetzt. (Sch11, S. 22 ff.)

3.3 Umsetzung

In diesem Abschnitt werden die entwickelten Java-Klassen und -Pakete im Detail erläutert. Im Anschluss wird kurz der Programmablauf aus Sicht des LicenseManager wiedergegeben.

[1] Genauere Erläuterungen folgen in Abschnitt 3.3.3

3.3.1 ODRLLicense

Die ODLRLicense-Klasse bildet ein Kernstück der Arbeit. Mit ihr ist es möglich eine vorhandene oder zu erstellende ODRL/XML-Lizenz in einem Objekt-Modell abzubilden, dass innerhalb von TURM zum Austausch und zur weiteren Verarbeitung von Lizenzinformationen verwendet werden kann. Die ODRLLicense-Klasse folgt dabei in ihrem Aufbau dem ODRLv2 Core Model (siehe Abschnitt 2.2). Das heißt jede Entität im Core Model wird als Klasse repräsentiert und die Attribute als jeweilige Klassenattribute. Ebenso werden die Assoziationen zwischen den Entitäten abgebildet. Die Klassen des Pakets *odrllicense* und ihre Beziehungen sind in Abbildung 3.2 dargestellt.

Als Hauptklasse fungiert dabei *ODRLLicense*, die eine komplette ODRL-Lizenz repräsentiert. Sie enthält neben dem zusätzlichen Attribut *filepath* (dem Speicherort der Lizenz) ein *Rights*-Objekt.

Die Klasse *Rights* besitzt die ODRL-Attribute *uid*, *type*, *conflict*, *inherit* und *undefined*. Die zugehörigen Objekte vom Typ *Permission* und *Prohibition* sind in den Listen *permissionList* und *prohibitionList* organisiert.

Permissions und Prohibitions unterscheiden sich nur durch die Tatsache, dass Permissions auch Duties beinhalten können. Daher sind die gemeinsame Merkmale von Permission und Prohibition in der abstrakten Super-Klasse *RightExpression* beschrieben. *RightExpression* enthält die Attribute *action*, *asset* und *assigner* (vom Typ *Party*). Constraints und Assignees der RightExpression sind in den Listen *constraintList* und *assignees* organisiert.

Die Klasse *Permission* erweitert RightExpression noch um die Liste *dutyList* vom Typ Duty. Die Klasse *Prohibition* hingegen enthält keine Erweiterungen.

Die Klasse *Asset* besitzt die Attribute *uid* und *inherit*. Das *inherit*-Attribut ist aber nicht weiter implementiert und wird insbesondere in der URM-Variante nicht unterstützt.

Die Klasse *Party* besitzt ein *uid*-Attribut.

Die Klasse *Action* besitzt ein *name*-Attribut.

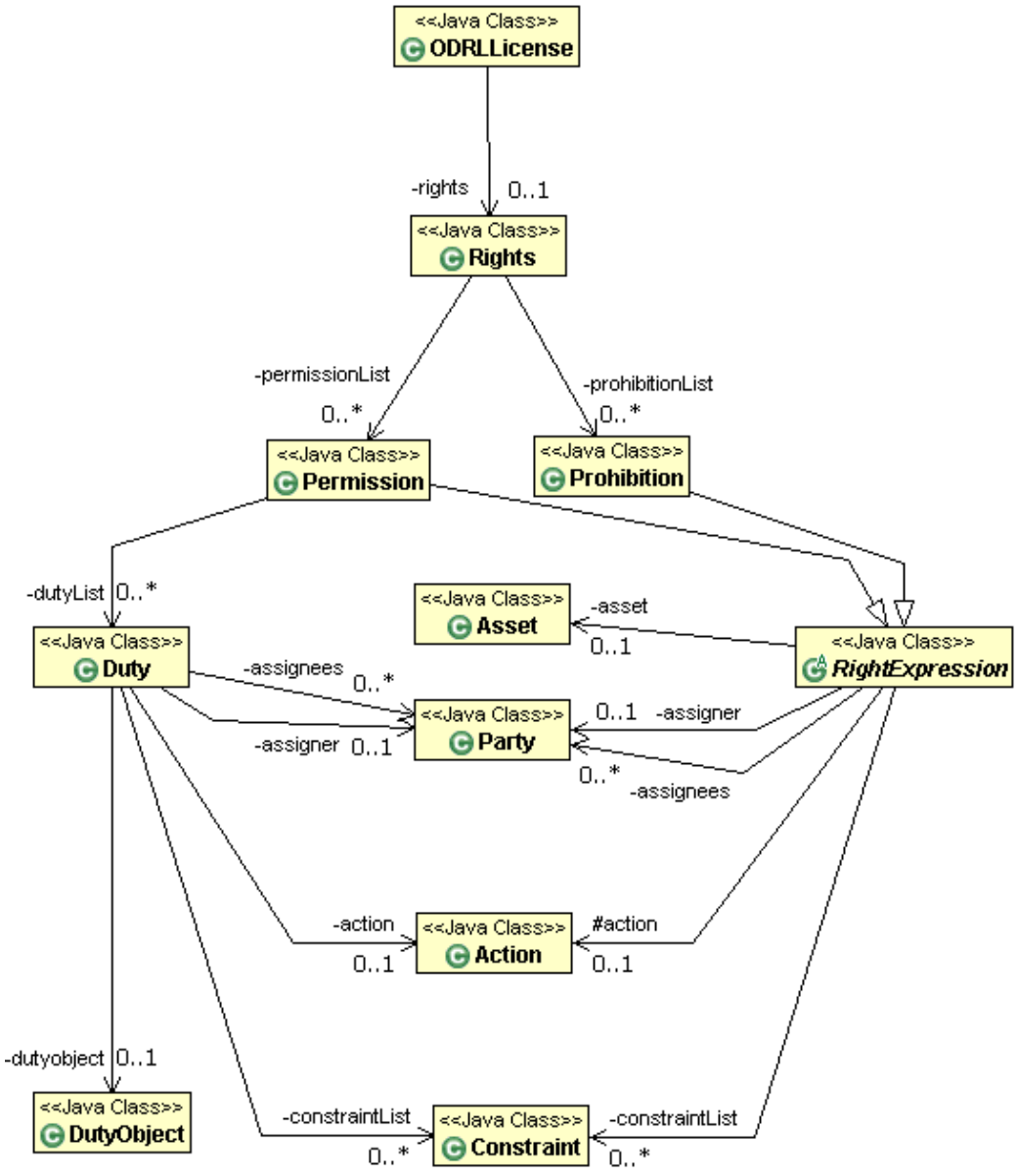

Abbildung 3.2: Klassendiagramm: Übersicht des Pakets *odrllicense* (erstellt mit *ObjectAid UML Explorer*)

Die Klasse *Constraint* besitzt neben dem name-Attribut die Attribute *operator*, *rightOperand* und *status*.

Die Klasse *Duty* besitzt ähnlich wie die Klasse *RightExpression* die Attribute *action*, *assigner*, *constraintList* und *assignees*. Hinzu kommt das Attribut *dutyobject* vom Typ *DutyObject*. Nach dem Core Model kann die Entität Duty auch auf ein Asset verweisen (GI09b). Diese Funktionalität wurde hier aber nicht implementiert, um die Komplexität des Modells zu begrenzen. Die Klasse *DutyObject* selbst besitzt die Attribute *measure* und *value*.

Die Klassen *Asset*, *Party*, *Action* besitzen keine öffentlichen Konstruktor-Methoden. Instanzen lassen sich außerhalb des Pakets *odrllicense* nur über die Klasse *ODRLEntityFactory* (nicht in der Abbildung enthalten) erzeugen. Damit soll vermieden werden, dass mehrere Objekte mit dem gleichen Inhalt erzeugt werden, wie es bei diesen Klassen wegen ihres häufigen Vorkommens innerhalb einer Lizenzsammlung zwangsläufig passieren würde. Die ODRLEntityFactory überprüft nach Aufruf einer ihrer Methoden die übergebenen Parameter darauf, ob bereits ein passendes Objekt erzeugt wurde und gibt dieses zurück. Handelt es sich um neue Angaben, so wird eine neue Instanz erzeugt und zurück gegeben. Gleichzeitig wird die neue Instanz auch in einer Liste für den zukünftigen Abgleich hinterlegt.

Die meisten Klassen besitzen zudem das nicht-öffentliche Integer-Attribut *id* und einen nicht-öffentlichen beziehungsweise geschützten Konstruktor ohne Argumente. Beides dient der Persistenz-Verwaltung durch Hibernate (siehe Abschnitt 3.3.5.1).

3.3.1.1 URMLicense-Klasse

Für das URM-Projekt wurde die Entitäten der ODRLLicense erweitert und neue Komponenten hinzugefügt. Diese Erweiterungen wurden in der abgeleiteten Klasse URMLicense (im Paket *urmlicense*[2]) implementiert. Neben der Hauptklasse URMLicense wurden nennenswerte Erweiterungen

[2]vollständiger Name: *de.uni_koblenz.aggrimm.turm.license_manager.odrllicense.urmlicense*

an den Klassen Rights (*URMRights*), Asset (*URMAsset*) und Party (*URMParty*) vorgenommen. Eingeführt wurden zusätzlich die Klasse *URMSource* und deren abgeleitete Klasse *URMCDRip* (siehe Abbildung 3.3).

So enthält die abgeleitete Klasse *URMAsset* das Attribut *source* vom Typ *URMSource*. Mit der Klasse *URMSource* lässt sich der Ursprung des Assets definieren. URMSource fungiert dabei als Super-Klasse. Die eigentlichen Definitionen werden in abgeleiteten Klassen vorgenommen. Mit Hilfe dieser Quellennachweise sollen vor allem implizite oder abgeleitete Rechte an Gütern (zum Beispiel: eine Privatkopie) nachgewiesen werden können. Die Quellenbeschreibungen und die benötigten Klassen sollen in Zukunft innerhalb des URM-Projekt entwickelt werden.

Zur Anschauung wurde in dieser Studienarbeit *URMCDRip* als solche Klasse definiert. Entstammt ein Asset von einer Musik-CD, so kann diese Ursprungs-CD mit der Klasse *URMCDRip* spezifiziert werden. Dazu stehen die Merkmale Katalognummer (Attribut *catalogueNumber*), Label Code (*labelCode*), Titelnummer auf der CD (*trackNumber*), Datum der Extraktion (*ripDate*) und physische Ort der CD (*plocation*) zur Verfügung. In Listing 3.1 ist als Beispiel ein fiktiver CD-Rip-Quellenachweis in der ODRL/XML-Darstellung zu sehen (3 - 11).

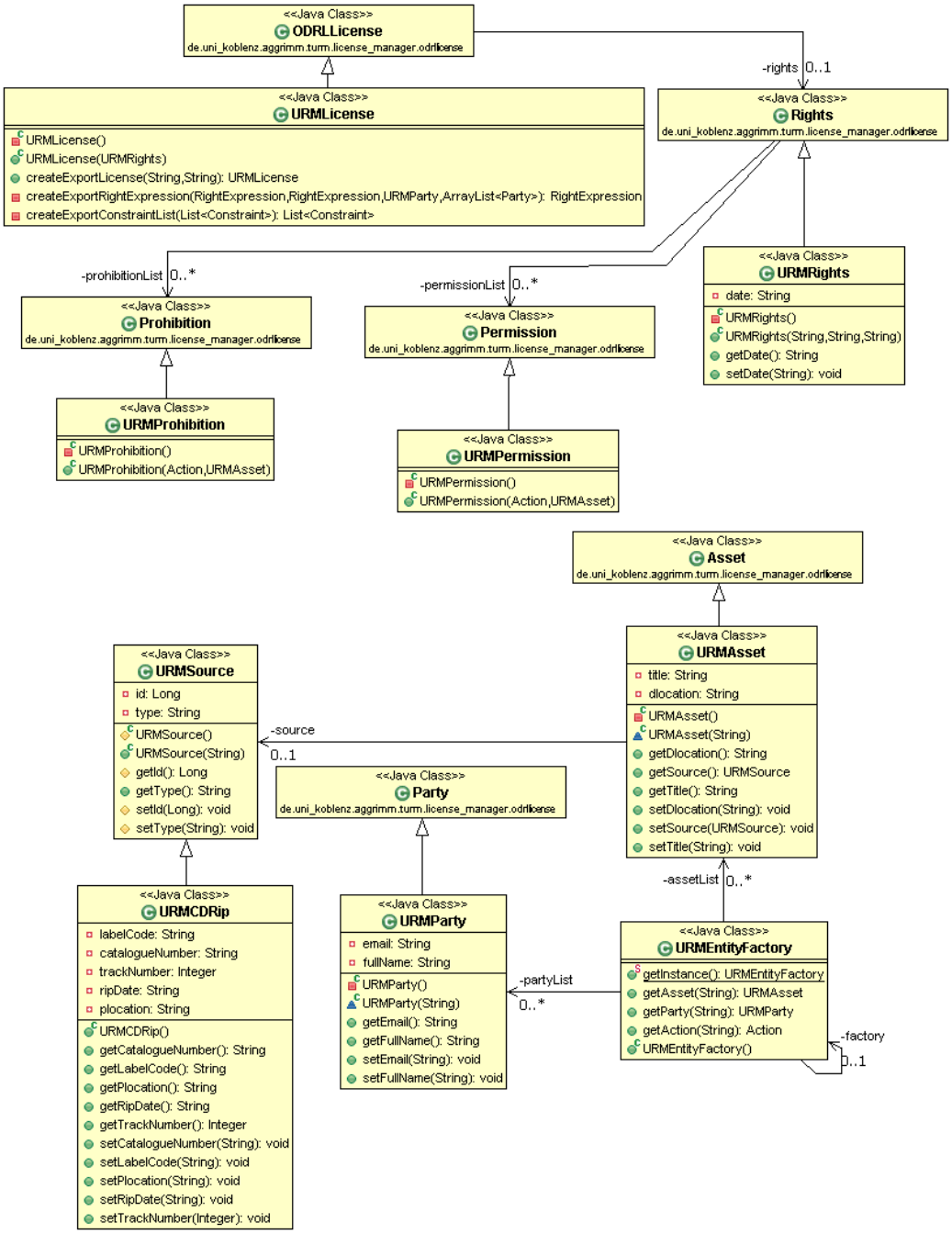

Abbildung 3.3: Klassendiagramm: Erweiterungen der *ODRLLicense* im Paket *urmlicense*(erstellt mit *ObjectAid UML Explorer*)

```
<o:asset xml:id="asset1" uid="SHA-1
    _95f7744065905b5b8e653cda1b3ebcf702e99c4">
     <d:title>Andrea Barone – Arabique</d:title>
         <source>
             <cd-rip>
                 <labelCode>exa-mple</labelCode>
                 <catalogueNumber>999-999</catalogueNumber>
                 <trackNumber>09</trackNumber>
                 <ripDate>1999-09-09</ripDate>
                 <plocation>cd rack</plocation>
             </cd-rip>
         </source>
     <dlocation>file://F:\01 – Arabique.mp3</dlocation>
</o:asset>
```

Listing 3.1: Beispiel eines CD-Rip

Weiterhin wurde in der Klasse *URMAsset* das Attribut *title* eingeführt, dass die Bezeichnung eines Assets ermöglicht (z.B. „Artist Name - Song Title“). Mit dem Attribut *dlocation* lässt sich der (zuletzt verwendete) Speicherort des Assets festhalten. Beide Attribute sind ebenfalls im Listing 3.1 beispielhaft aufgeführt.

Die Klasse *URMParty* enthält gegenüber ihrer Super-Klasse Party zusätzlich die Attribute *email* (Email-Adresse des Teilnehmers) und *fullName* (ausgeschriebener Name des Teilnehmers) sowie die entsprechenden Methoden zum Lesen und Schreiben dieser Werte.

Die Klasse *URMRights* besitzt zusätzlich das Attribut *date*, welches das Ausstellungs-Datum der Lizenz angibt.

Die Hauptklasse *URMLicense* stellt die Methode *createExportLicense(String assignerPartyUID, String assigneePartyUID)* bereit, die eine abgeleitete Instanz der URMLicense zurück gibt. Eine sogenannte Export-Lizenz kann vom Nutzer erzeugt werden, um die Rechte, die er aufgrund einer vorhandenen Lizenz besitzt, an Dritte weiterzugeben (siehe auch (Jah10, S. 12 f., 16 f.) und (Lie10, S. 50 ff.)).

Dazu werden der Methode die *uid*-Attribute des neuen Assigner und neuen Assignee übergeben. In der Kopie der URMLicense werden die-

se Angaben dann ersetzt. Das *uid*- und das *date*-Attribut des URMRights-Objekts werden ebenfalls neu gesetzt. Die restlichen Angaben werden aus der originalen URMLicense übernommen.

In der Regel erfolgt der Aufruf dieser Methode aus dem LicenseManager heraus über die Methode *createExportLicense*

3.3.2 LicenseManager

Der LicenseManager bildet das andere Kernstück dieser Arbeit. Es ist die Klasse, die Methoden zur Lizenzverwaltung nach außen zur Verfügung stellt und den Ablauf bei Operationen auf den Lizenzen sicherstellt.

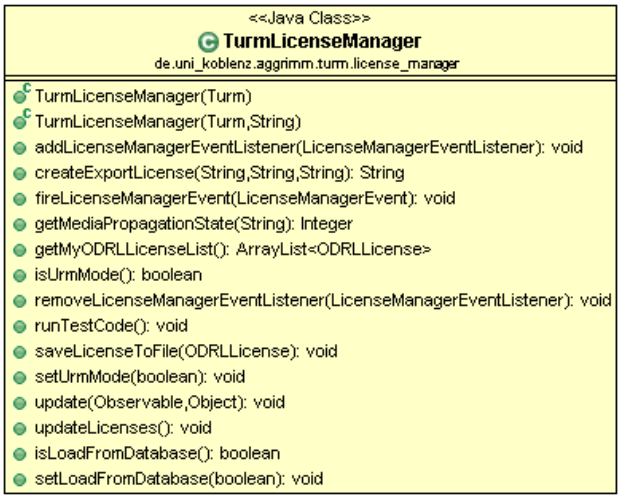

Abbildung 3.4: Klassendiagramm: *public*-Methoden der Klasse LicenseManager (erstellt mit *ObjectAid UML Explorer*)

Einige Konfigurationsmerkmale des LicenseManager werden vom ConfigurationManager des TURM verwaltet, die zum Zeitpunkt der Initialisierung vom LicenseManager abgefragt werden (siehe Abschnitt 2.4.2). Das ist zum einen der Datei-Pfad des Lizenz-Ordners (Attribut *folderPath*), der unter dem Schlüssel *core:licensesfilespath* im ConfigurationManager abgelegt ist, und zum anderen das boolesche Attribut *loadFromDatabase*, das angibt, ob die Lizenzen zu Beginn aus der Datenbank geladen werden sollen und unter dem Schlüssel *core:loadLicensesFromDatabaseAtStartup* abge-

legt ist.

Um auf Veränderungen der Konfiguration während der Laufzeit reagieren zu können, ist beim LicenseManager das Interface *Observer* implementiert. Findet eine Änderung an den Parametern innerhalb des ConfigurationManager statt, so wird die Methode *update(Observable, Object)* aufgerufen. Daraufhin muss zunächst geprüft werden, ob die Parameter des LicenseManager sich geändert haben und diesem Fall die lokalen Attribut-Werte angepasst werden. Dies betrifft in der derzeitigen Implementierung nur das Attribut *folderPath*.

Der Wert des Attributes *loadFromDatabase*, das festlegt, ob die Lizenzen aus der Datenbank geladen werden, kann über die Methode *isLoadFromDatabase()* abgefragt werden. Mit *setLoadFromDatabase(boolean)* kann der Wert verändert werden. Eine Änderung wird dem ConfigurationManager übergeben und damit erst bei erneuter Initialisierung des LicenseManager wirksam.

Der LicenseManager unterscheidet zwischen Lizenzen, bei denen der Nutzer in der Rolle des Assignee ist und Lizenzen, die er selbst als Assigner an Dritte ausgibt. Erstere werden in der Liste *myODRLLicenseList*, letztere in der Liste *exportODRLLicenseList*. Auf der Festplatte werden die Lizenzen getrennt in den Unterordnern *myLicenses* respektive *exportLicenses* verwaltet. Diese Unterteilung wird so schon in der voran gegangenen ODRL-Medienbibliothek (siehe 2.4.1.1 und (Jah10)) verwendet. Die Pfadangaben der Unterordner sind in den Attributen *exportLicenseDirectoryPath* / *myLicenseDirectoryPath* hinterlegt. Unter den Attributen *exportLicenseFileList* / *myLicenseFileList* sind Referenzen zu den Lizenz-Dateien in Form von *File*-Objekten hinterlegt.

Der LicenseManager behandelt die verwalteten Lizenzen entweder als ODRL- oder als URM-Lizenzen und nutzt dementsprechend den passenden *LicenseReader* und *LicenseWriter*. Entscheidend ist dabei, ob das boolesche Attribut *urmMode* gesetzt ist (der Wert *true* erzwingt die URM-Behandlung). Standardmäßig ist dieser Wert auf *true* gesetzt, da in der Entwicklung vor allem mit URM-Lizenzen gearbeitet wird. Der Wert kann über die Methode *isUrmMode()* abgefragt werden. Über die Methode *se-*

tUrmMode(boolean) kann der Wert geändert werden.

Diese starre, im Programmcode getroffene Festlegung auf einen bestimmten *LicenseReader* beziehungsweise LicenseWriter wurde bewusst gewählt, um die Entwicklung zur vereinfachen. In der zukünftigen Entwicklung kann dies jedoch optimiert werden, indem je Lizenz respektive Lizenz-Objekt die passenden *LicenseReader* und *LicenseWriter* gewählt werden.

Referenzen zu den vom LicenseManager verwendeten Instanzen von *Turm*, *DBManager* und *LicenseReader* sind in den jeweiligen Attributen *turm*, *dbmanager* und *licenseReader* hinterlegt.

3.3.2.1 Methoden

Die Initialisierung des LicenseManager erfolgt mit Aufruf der Konstruktormethode *TurmLicenseManager(Turm)*. Dabei wird die zugehörige *Turm*-Instanz übergeben, über die der LicenseManager dann auch auf andere zentrale Komponenten (*ConfigurationManager*, *DBManager*) zugreifen kann. Alternativ lässt sich beim Aufruf mit *TurmLicenseManager(Turm, String)* auch ein Datei-Pfad zu den Lizenzen angeben. Standardmäßig wird sonst der hinterlegte Wert vom ConfigurationManager übernommen.

Die aktuell verwalteten *ODRLLicense*- beziehungsweise *URMLicense*-Objekte, die dem Nutzer Rechte zuschreiben (*myODRLLicenseList*), lassen sich über die Methode *getMyODRLLicenseList()* abrufen. Neu erzeugte *ODRL-/URMLicense*-Objekte lassen sich mit der Methode *saveLicenseToFile(ODRLLicense)* abspeichern.

Eine Export-Lizenz lässt sich mit der Methode *createExportLicense(String, String, String)* erzeugen. Dabei wird auf die Methode *createExportLicense(String, String)* der Klasse *URMLicense* zurückgegriffen. Die genaue Funktionsweise ist daher in Abschnitt 3.3.1.1 beschrieben.

Mit der Methode *updateLicenses()* lässt sich auch von außen eine Aktualisierung des Lizenzbestandes anstoßen. Ansonsten wird sie im internen Programmablauf nach Hinzufügen neuer Lizenzen aufgerufen. Die vorhandenen Lizenzen in den Festplattenordnern (*myLicenses* und *export-*

Licenses) werden dabei mit den verwalteten *ODRL-/URMLicense*-Objekten verglichen. Entscheidend ist hierbei das Attribut *filepath* der *ODRL-/URMLicense*-Objekte. Nicht mehr vorhandene Lizenzen auf der Festplatte werden aus der Liste entfernt. Neue Lizenzen werden der Liste hinzugefügt. Diese Änderungen werden anschließend (wie alle Änderungen an den *ODRLLicense*-Listen) auch in der Datenbank durch Aufruf des DBManager vollzogen.

Die Methode *getMediaPropagationState(String)* gibt für eine Lizenz den Weitergabe-Status in Form eines Zahlenwertes zurück. Sie gibt vereinfacht an, ob ein Asset *unbeschränkt*, *beschränkt* oder *nicht* weitergegeben werden darf. Diese Methode wurde für den von Verena Liesenfeld entwickelten P2P-Client CUP bereitgestellt und wird auch ausschließlich von diesem verwendet (siehe (Lie10)).

Die Methoden *addLicenseManagerEventListener(LicenseManagerEventListener)*, *removeLicenseManagerEventListener(LicenseManagerEventListener)* und *fireLicenseManagerEvent(LicenseManagerEvent)* sind Bestandteil eines *Observer*-Entwurfsmusters und wurden von Florian Schlünß implementiert. Innerhalb dieses Entwurfsmusters fungiert der LicenseManager als *Subjekt*, dass die *Beobachter* informiert, sobald Änderungen an den verwalteten Lizenzen vorgenommen werden. (Sch11, S. 19 ff.)

Als Beobachter registriert sich derzeit nur der *MediaLicenseMapper*. Aber auch andere Komponenten haben die Möglichkeit sich (dynamisch) als Beobachter anzumelden.

3.3.3 LicenseReader

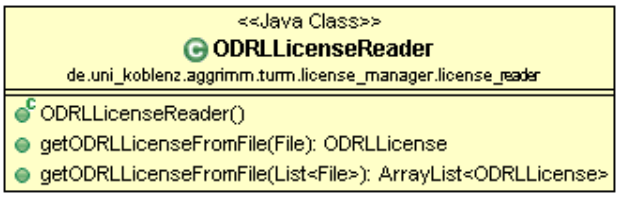

Abbildung 3.5: Klassendiagramm: *public*-Methoden der Klasse LicenseReader (erstellt mit *ObjectAid UML Explorer*)

Die *LicenseReader*-Klasse stellt Methoden bereit, um vorhandene ODRL-Lizenz-Dateien als *ODRLLicense*-Objekte zu importieren.

Dazu wird dem *LicenseReader* beim Methodenaufruf eine Liste von *File*-Objekten übergeben, die auf die zu importierenden Lizenz-Dateien verweisen. Diese ODRL/XML-Dateien werden zunächst mit einem XML-Parser in eine Baumdarstellung im Document-Object-Model-Format[3] (*DOM*) gebracht (Methode *parseXMLFile(String)*). Hierfür werden Parser aus dem Java-eigenen Paket *javax.xml.parsers* genutzt.

Die DOM-Baumdarstellung und die zugehörige API ermöglichen eine sehr strukturierte Darstellung und Verarbeitung des XML-Dokumentes. Gegenüber einer seriellen Verarbeitung hat sie lediglich bei größeren Dokumenten Nachteile, da diese zur Laufzeit mehr Speicherauslastung verursachen. Da die insbesondere hier verwendeten ODRL-Lizenzen in ihrer Ausgestaltung jedoch überschaubar sind, wird dieser Nachteil in Kauf genommen.

Liegt der DOM-Baum des Dokumentes vor, wird dieser von der Wurzel beginnend durchgegangen und aus den Inhalten ein äquivalentes *ODRLLicense*-Objekt erstellt. Der DOM-Baum wird dazu in seinen Knoten (*Nodes*) nach Element- oder Attribut-Begriffen aus dem ODRL Core Model (siehe Abschnitt 2.2) durchsucht. Werden treffende Ausdrücke gefunden, so werden innerhalb der *ODRLLicense* entsprechende Entitäten erzeugt beziehungsweise werden den Attributen der *ODRLLicense* entsprechende Werte zugewiesen.

Die erzeugten *ODRLLicense*-Objekte werden an den LicenseManager zurückgegeben.

Die Unterklasse *URMLicenseReader* erweitert die Funktionalität von *LicenseReader*, um *URMLicense*-Objekte aus URM-Lizenz-Dateien erstellen zu können.

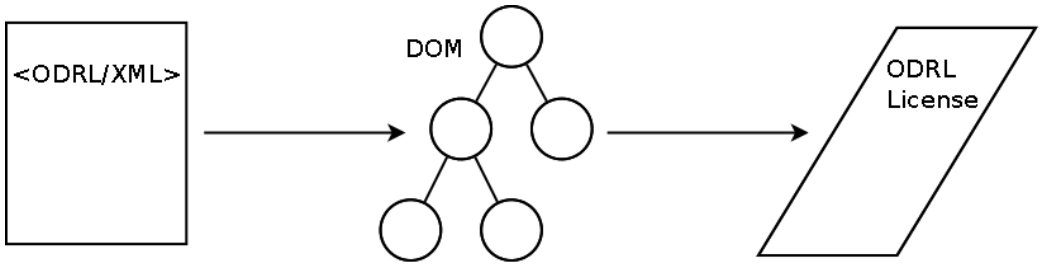

Abbildung 3.6: Schema: Importieren von ODRL/XML-Lizenzen über einen DOM-Tree

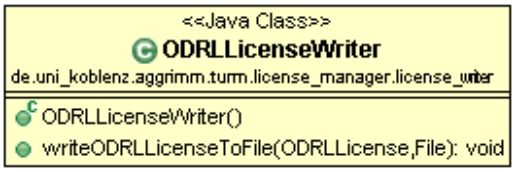

Abbildung 3.7: Klassendiagramm: *public*-Methoden der Klasse LicenseWriter (erstellt mit *ObjectAid UML Explorer*)

3.3.4 LicenseWriter

Die LicenseWriter-Klasse stellt Methoden bereit, um vorhandene *ODRLLicense*-Objekte als ODRL-Lizenz-Datei zu exportieren.

Das übergebene *ODRLLicense*-Objekt wird dazu nach Methodenaufruf in eine DOM-Baumdarstellung überführt. Danach wird diese ins XML-Format konvertiert und in einer Datei ausgegeben .

In ihrer Funktionsweise folgt die *LicenseWriter*-Klasse der *LicenseReader*-Klasse (siehe vorherigen Abschnitt 3.3.3) bei Umkehrung der Flussrichtung:
Das übergebene *ODRLLicense*-Objekt wird von der „Wurzel“-Klasse beginnend untersucht und zeitgleich eine äquivalente Lizenz in der DOM-Baumdarstellung aufgebaut. Für alle vorhandenen (unterstützten) Objekte und Attribute innerhalb des *ODRLLicense*-Objekt werden entsprechende *Nodes* im DOM-Baum erstellt. Der fertige DOM-Baum wird schließlich

[3]Das Document Object Model (DOM) ist eine Schnittstelle für den Zugriff auf HTML- respektive XML-Dokumente. Sie wird vom World Wide Web Consortium spezifiziert. (Wor05)

von einem XSLT[4]-Prozessor in XML umgesetzt und an die im Dateipfad angegeben Stelle als ODRL/XML-Datei geschrieben (Methode *writeDocumentToFile(Document, File)*.

Die ausgegebenen Dateien sind in ihrer Darstellung äußerst simpel und in ihrer Struktur nicht effizient. So werden zum Beispiel mehrfach vorkommende <*asset*>-Ausdrücke innerhalb einer Datei nicht zusammengefasst. Die einzelnen Rechteausdrücke sind damit stets ausführlich. Die Ursache liegt in den verwendeten XML-Bibliotheken, die derartige Zusammenfassungen nicht automatisiert unterstützen. Der Aufwand diese Funktionalität selbst zu programmieren, wäre in diesem Rahmen zu groß. Insbesondere hat die verwendete Darstellung keinen Einfluss auf die inhaltliche Korrektheit der erstellten Lizenzen.

Die Unterklasse *URMLicenseWriter* erweitert die Funktionalität von *LicenseWriter*, um *URMLicense*-Objekte in Lizenz-Dateien umwandeln zu können.

[4]Extensible Stylesheet Language (XSL) Transformation

3.3.5 DBManager

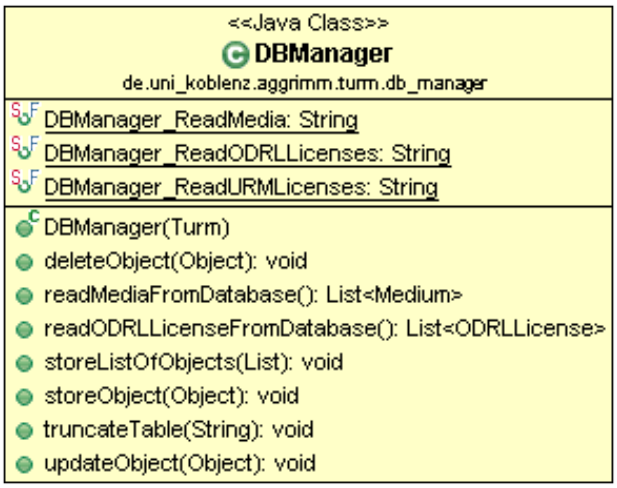

Abbildung 3.8: Klassendiagramm: *public*-Attribute und Methoden der Klasse DBManager (erstellt mit *ObjectAid UML Explorer*)

Der DBManager sorgt für die Persistenz der zur Laufzeit generierten Lizenz-Objekte. Um ein wiederkehrendes Laden der Lizenzen von der Festplatte bei jedem Programmstart zu vermeiden, werden die Objekte in einer SQLite-Datenbank abgelegt.

Zur Abfrage der Datenbank setzt der DBManager auf das Hibernate-Framework (siehe Abschnitt 3.3.5.1). Hibernate stellt Bibliotheken bereit, um Java-Objekte in einer Datenbank persistent zu halten. Dazu müssen die Objekt-Klassen auf ein oder mehrere Datenbank-Tabellen gemappt werden. Hierzu werden XML-Mapping-Dateien benutzt, die beschreiben, wie die Objekte in der Datenbank repräsentiert werden.[5]

Ein Veranschaulichung der Abhängigkeiten beziehungsweise der Zugriffsstruktur innerhalb der Datenbanklösung ist in Abbildung 3.9 zu sehen.

Neben dem LicenseManager wird der DBManager auch vom MediaManager zur Persistenz-Sicherung genutzt, wenn auch in geringerem Umfang. Dies dient vor allem zur Vermeidung von weiteren parallelen Code-

[5] Alternativ kann dies auch über Java-Annotationen gemacht werden.

Entwicklungen. Einige Funktionen beschränken sich daher nicht auf ODRL-License-Objekte und stehen daher auch anderen, zukünftigen Komponenten zur Verfügung.

3.3.5.1 Hibernate

Hibernate ist ein von Red Hat[6] entwickeltes quelloffenes Java-Framework für objektrelationales Mapping. Es ermöglicht die Konvertierung zwischen Java-Objekten und relationalen Datenbanken. Die Schnittstellen zur Datenbank werden dabei innerhalb von Java als Objekte respektive zugehörige Methoden bereit gestellt. Hibernate ist als freie Software unter der *GNU Lesser General Public License* veröffentlicht. (JBo)

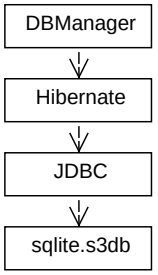

Abbildung 3.9: Schema: Abhängigkeiten innerhalb der Datenbanklösung

Ein objektrelationales Framework bietet aus Sicht dieser Studienarbeit den Vorteil, dass auf vorhandene Programmbibliotheken zurückgegriffen werden kann und keine zusätzliche Schnittstellen eigens entwickelt werden müssen. Auch bei späteren Änderungen am Objekt- oder Datenbankschema dürfte sich der Aufwand gering halten.

Hibernate im Speziellen bietet den Vorteil, dass es sich dabei um ein weit entwickeltes Framework handelt und auf eine große Entwickler-Gemeinschaft bauen kann. Zudem ermöglicht es durch die Quelloffenheit, falls nötig, Änderungen an den bestehenden Bibliotheken vorzunehmen.

Damit die Java-Objekte in der Datenbank abgelegt werden können, muss jede Objektklasse, die persistent gehalten werden soll, entsprechenden Tabellen respektive Feldern im Datenbank-Schema zugewiesen werden. Diese Zuordnungen werden in einer eigenen XML-Sprache verfasst und in Mapping-Dateien abgelegt. Für die Beschreibung der Lizenzen sind das die Dateien *ODRLLicense.hbm.xml* und *URMLicense.hbm.xml*. Auch hier

[6] http://www.redhat.com/

kann ähnlich wie bei der Klassenvererbung auf die Definitionen für ODRL-License innerhalb der Datei *ODRLLicense.hbm.xml* aufgebaut werden, sodass für die Beschreibung für URMLicense in *URMLicense.hbm.xml* nur die entsprechenden Erweiterungen der Klasse zugeordnet werden müssen.

Als Beispiel für ein Mapping ist im Listing 3.2 ein Ausschnitt aus der Mappingdatei *ODRLLicense.hbm.xml* dargestellt. Hier wird in einem <class>-Ausdruck die Objektklasse *Asset* auf die Tabelle *odrl_assets* gemappt.

Der <id>-Ausdruck legt fest, dass eine eindeutige ID vom DBMS[7] erzeugt wird und im Feld *id* in der Tabelle beziehungsweise in dem Attribut *id* des Objektes gespeichert wird. Der <discriminator>-Ausdruck führt ein Feld *asset_type* ein, mit dem sich die *Asset*-Klasse von der *URMAsset*-Klasse unterscheiden lässt.[8] In den beiden <property>-Ausdrücken werden die Attribute *uid* und *inherit* jeweils den Feldern *asset_uid* und *asset_inherit* zugewiesen.

```
1 <class name="Asset" table="odrl_assets">
2         <id name="id" column="o_asset_id" type="long">
3                 <generator class="native"></generator>
4         </id>
5         <discriminator column="asset_type" type="string" force="
              true"/>
6         <property name="uid" column="asset_uid" type="string"></
              property>
7         <property name="inherit" column="asset_inherit" type="
              string"></property>
8 </class>
```

Listing 3.2: Mapping der Objektklasse *Asset* innerhalb der Datei *ODRLLicense.hbm.xml*

Die Mappingdateien werden beim Aufruf der Hibernate-Bibliothek gelesen. Die allgemeine Konfiguration (zu der auch die Auflistung der einzubindenen Mapping-Dateien gehört) wird in der Konfigurationsdatei *hibernate.cfg.xml* verfasst. (KBA+11)

[7] Database Management System

[8] Die *URMAsset*-Objekte werden in derselben Tabelle abgelegt wie die Objekte der Oberklasse *Asset*. Das Mapping der *URMAsset*-Klasse ist, wie oben beschrieben, in der Datei *URMLicense.hbm.xml* hinterlegt.

Auf Basis der Mappingdateien werden auch die Java-Objekte erzeugt, wenn aus der Datenbank gelesen wird. Dazu benötigt Hibernate für jede zu erzeugende Klasse eine zugehörige leere Konstruktormethode und für die Attribute entsprechende *Setter*-Methoden. Es ist jedoch hinreichend, wenn diese Methoden die Sichtbarkeit *private* besitzen.

Als angebundene Datenbank wurde sich für SQLite entschieden (siehe Abschnitt 3.3.5.2). Da Hibernate nativ keine Unterstützung für SQLite mit sich bringt, wurde für den nötigen *SQL Dialect* eine Implementierung von Raojunxue[9] verwendet, die unter der Apache License 2.0 veröffentlicht ist.

3.3.5.2 SQLite

SQLite ist eine Programmbibliothek, die eine SQL-Datenbank implementiert. Dabei kommt SQLite ohne zentralen Server aus, weshalb es sich besonders für portable Software und bei beschränkten Ressourcen eignet. Die Datenbank wird im Einsatz in einer einzelnen Datei abgebildet.[10] Dies hält auch den Umfang der Datenbanklösung extrem gering. SQLite ist in der Public Domain veröffentlicht und damit gemeinfrei.[11]

SQLite eignet sich als einfache Datenbanklösung für TURM, da das Projekt in seinem verhältnismäßig frühen Entwicklungsstadium noch nicht für einen produktiven Einsatz konzipiert ist und es keine präferierte Ziel-Plattform für die Anwendung gibt. Sollten sich diese Vorgaben im späteren Verlauf der Entwicklung ändern, müsste der Einsatz von SQLite neu bewertet werden.

Die Verbindung zwischen der Datenbank und der Java-Software wird über eine JDBC[12]-Schnittstelle hergestellt. Als Treiber für diese Schnittstelle wird in der TURM-Implementierung der SqliteJDBC-Treiber von Zentus genutzt.[13] Dieser ist unter BSD-Lizenz veröffentlicht.

[9] http://code.google.com/p/hibernate-sqlite/ - Abruf: 06.05.2011
[10] http://sqlite.org/about.html - Abruf: 06.05.2011
[11] http://sqlite.org/copyright.html - Abruf: 06.05.2011
[12] Java Database Connectivity
[13] http://www.zentus.com/sqlitejdbc/ - Abruf: 06.05.2011

3.3.6 Programmablauf

Es soll hier die Funktionsweise des LicenseManager und der zugehörigen Komponenten nochmal verdeutlicht werden, indem ein genereller Programmablauf durchgegangen wird.

Der LicenseManager wird üblicherweise durch einen Aufruf/eine Zuweisung aus dem TURM Core heraus initialisiert. Zunächst werden die nötigen Parameter wie Lizenz-Pfad, DBManager etc. initialisiert. Danach werden die Lizenzen eingelesen. Entscheidend ist hier, ob das zugehörige Attribut loadFromDatabase gesetzt ist. Ist dies der Fall, werden die Lizenzen aus der Datenbank importiert, indem ein entsprechender Funktionsaufruf an den DBManager getätigt wird und die zurück gegebene Ergebnisliste eingepflegt wird. Danach erfolgt noch ein Update der Lizenzen durch einen Abgleich mit den real vorhandenen Lizenzen in den Lizenz-Ordnern auf der Festplatte.

Falls das Attribut loadFromDatabase nicht gesetzt ist, müssen die Lizenzen erst von der Festplatte eingelesen werden. Hierzu wird ein entsprechender Funktionsaufruf an einen zuvor initialisierten License Reader gesendet, der Listen mit Lizenzen zurück gibt. Anschließend werden die erhaltenen Lizenzen in die Datenbank geschrieben, bereits vorhandene Lizenzen werden aus der Datenbank gelöscht.

Nun steht der LicenseManager für Anfragen zur Verfügung. Dies beinhaltet das Abfragen der vorhandenen Lizenzen, das Hinzufügen neuer Lizenzen, das Erstellen von abgeleiteten Lizenzen (URM-spezifisch) und das Aktualisieren des Lizenzbestandes. Primär für die Komponente P2P-Manager wird die Methode getMediaPropagationState(String) bereit gestellt, die überprüft, ob eine bestimmte Lizenz die Weitergabe des Assets zulässt.

Alle Änderungen an den Lizenzen werden umgehend in der Datenbank gespeichert. Daher müssen vor dem Beenden des LicenseManager keine Operationen zur Datensicherung durchgeführt werden.

Ein Sequenzdiagramm des Programmablaufs mit beispielhaften Ereignissen ist in Abbildung 3.10 zu sehen. Dass beim Ausführen der Methode

updateLicenses() unter Umständen auch der DBManager und der License-Reader angesprochen werden, wurde hier zur Vereinfachung des Schaubildes außer Acht gelassen.

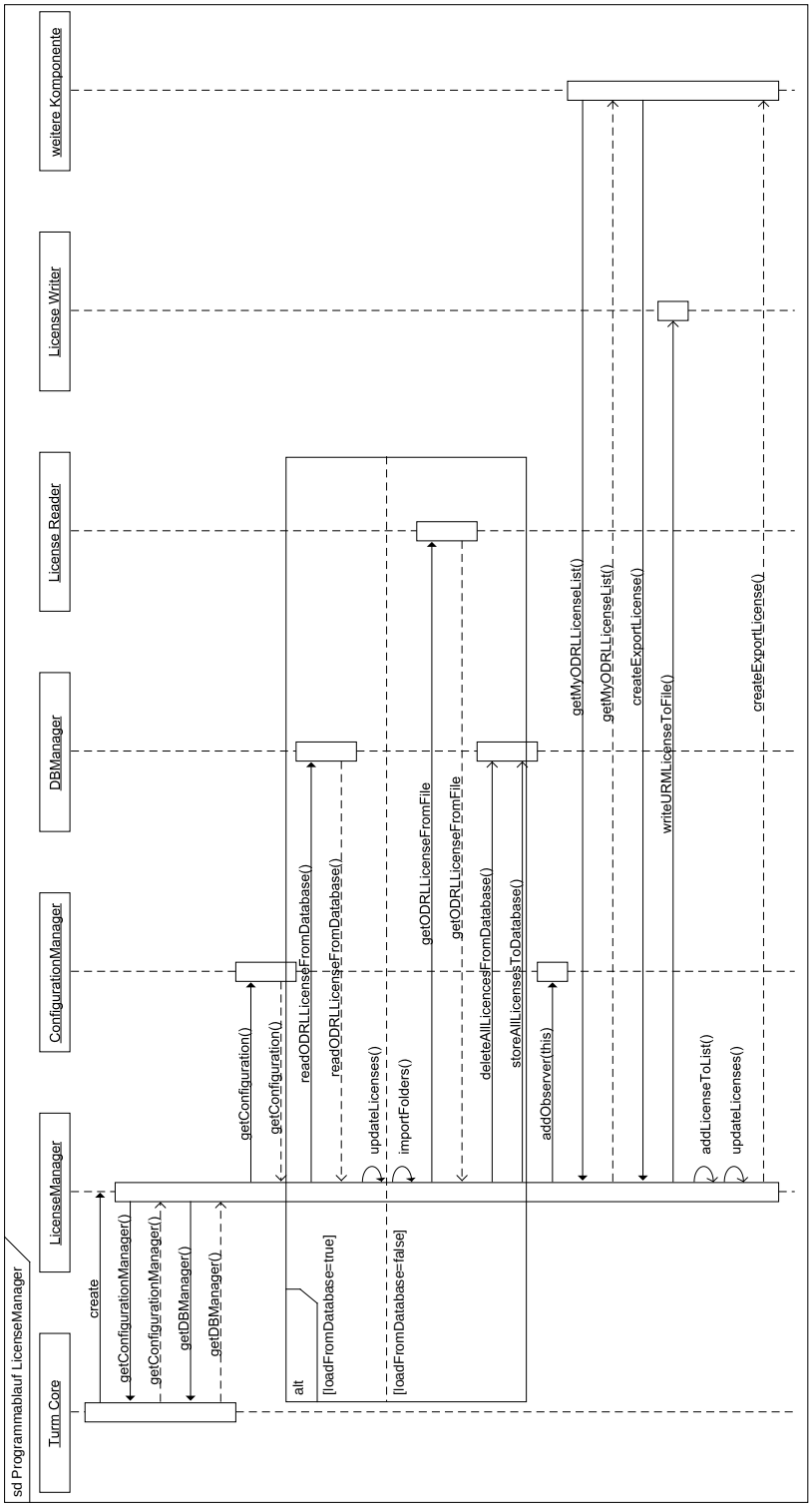

Abbildung 3.10: Sequenzdiagramm: Programmablauf des LicenseManager

Kapitel 4

Fazit

In dieser Studienarbeit wurde eine Entwicklungsversion des ODRL V2 Core Model erstmalig in ein Objekt-Modell als *ODRLLicense* umgesetzt. Es wurden Klassen geschaffen, um kompatible ODRL-Lizenz-Dateien in die Objekt-Darstellung zu konvertieren und ebenso ODRL-Lizenz-Objekte in Lizenz-Dateien auszugeben. Mit dem *LicenseManager* wurde für das TURM-Framework die Grundlage gelegt, um Rechteinformationen zu verarbeiten und auszutauschen. Der LicenseManager stellt damit die zentrale Komponente zur Verwaltung der Lizenzen dar. Eine persistente Speicherung von ODRL-Lizenz-Objekten und damit der Rechteinformationen über die Laufzeit hinaus, wird mit dem *DBManager* ermöglicht.

Der LicenseManager und die mit ihm verbundenen Komponenten geben dem TURM-Framework die Funktionalität, um den Nutzer über die Rechte an seinen Mediendateien zu informieren. Gleichzeitig gibt es jedoch erhebliches Potential für Erweiterungen und Verbesserungen am LicenseManager und am Framework.

Das hier implementierte ODRL V2 Core Model ist zum Zeitpunkt der Fertigstellung dieser Arbeit teilweise bereits veraltet. Es wird daher nötig sein, das Objekt-Modell der *ODRLLicense* entsprechend anzupassen, um die Kompatibilität zu erhalten. Dies betrifft natürlich auch das innerhalb vom URM-Projekt verwaltete URM-Lizenz-Format. Neben der Anpassung an Änderungen des ODRL V2 Core Model bedarf es hier einer

verbindlichen Festschreibung eines URM-Profile für ODRL V2. Entscheidend werden hier jedoch auch die weiteren Entwicklungen innerhalb in der ODRL-Arbeitsgruppe sein.

Wie in Abschnitt 3.3.2 bereits erwähnt, ist ein flexibler Einsatz unterschiedlicher *LicenseReader* und *LicenseWriter* noch nicht möglich. Denkbar wäre die Implementierung von *Factories*, die je nach Lizenztyp die passenden Objekte und Methoden bereitstellen. Weiter könnte über die Bündelung zu Reader/Writer-Paketen nachgedacht werden, die jeweils einen Lizenztyp gemeinsam bedienen.

Ferner wären auch Konverter denkbar, die Lizenz-Objekte aus ODRL-fremden Lizenzdateien generieren oder sich anderer Datenquellen für Lizenz- und Rechteinformationen bedienen.

Der LicenseManager selbst stellt zur Suche nach Lizenzen und Rechteinformationen nur rudimentäre Möglichkeiten bereit. Möglichkeiten zur Erweiterung bieten hier neben weiteren Funktionen des LicenseManager auch zusätzliche Funktionalitäten des DBManager, der den Lizenzbestand über die hinterlegten Informationen in der Datenbank analysieren könnte.

Aussichtsreicher scheint jedoch der Versuch, die die ODRL-Lizenzen beziehungsweise die ODRLLicense-Objekte in RDF[1]-Syntax auszudrücken, wie dies in einer Bachelorarbeit von Jonas Zitz bereits geschieht. Diese Vorgehensweise bietet das Potential, auch nicht explizit vorhandene Rechteinformationen aus einem Lizenzbestand herauszulesen und eröffnet damit weitere Anwendungsmöglichkeiten. Unter Umständen könnte RDF das Objekt-Modell bei der internen Repräsentation von Rechteinformation sogar ablösen.

Die Vorhaltung der Lizenzen als jeweils einzelne konkrete Objekte kann gerade bei größeren Lizenz-Sammlungen zu ebenso größerer Speicherauslastung führen, die in dieser Studienarbeit noch nicht untersucht wurde. Derartige Performance-Tests wären in späteren Entwicklungsstadien auch für andere Komponenten des Frameworks sinnvoll.

Innerhalb des TURM-Frameworks ist langfristig ein Ausbau der unterstützten Medientypen sinnvoll. Bisher beschränkt sich die Auswahl auf

[1] Resource Description Framework

MP3-Dateien, was angesichts des Referenzcharakters des Formats und der bisherigen Ausrichtung voran gegangener Arbeiten gerechtfertigt ist. Die Verwaltung von Rechteinformationen bietet aber nicht nur im Unterhaltungsbereich sondern etwa auch bei geschäftlichen Dokumenten Perspektiven.

Zuletzt muss überlegt werden, wie die beabsichtigte Veröffentlichung des Programmcodes unter einer Open-Source-Lizenz geschehen soll und wie darauf basierend ein Open-Source-Projekt unter Führung der Arbeitsgruppe etabliert werden kann.

Literaturverzeichnis

[Bro06] BROWN, Peter: *What is DRM? - Digital Restrictions Management*. Online Ressource. http://www.defectivebydesign.org/what_is_drm. Version: 2006, Abruf: 04.04.2011

[FK04] FRÄNKL, Gerald ; KARPF, Philipp: *Digital Rights Management Systeme - Einführung, Technologien, Recht, Ökonomie und Marktanalyse*. PG Verlag, 2004 http://www.digital-rights-management.info/. – ISBN 9783937624006

[GI09a] GUTH, Susanne ; IANNELLA, Renato: *ODRL V2.0 - Common Vocabulary, Working Draft: 25 September 2009*. Online Ressource. http://odrl.net/2.0/WD-ODRL-Vocab-20090925.html. Version: September 2009, Abruf: 05.04.2011

[GI09b] GUTH, Susanne ; IANNELLA, Renato: *ODRL V2.0 - Core Model, Draft Specification: 23 September 2009*. Online Ressource. http://odrl.net/2.0/DS-ODRL-Model-20090923.html. Version: September 2009, Abruf: 30.03.2011

[Gri05] GRIMM, Prof. Dr. R.: DRM-Techniken und ihre Grenzen. In: (PT05), S. 85 – 96

[HPG09] HUNDACKER, Helge ; PÄHLER, Daniel ; GRIMM, Rüdiger: URM - Usage Rights Management. In: ARNAB, Alapan (Hrsg.): *Workshop proceedings of the 7th International Workshop for Technical, Economic and Legal Aspects of Business Models for Virtual Goods*, 2009, 125 - 138

[Ian08] IANNELLA, Renato ; THE ODRL INITIATIVE (Hrsg.): *Open Digital Rights Language (ODRL)*. 1.1. : The ODRL Initiative, August 2008. http://odrl.net/1.1/ODRL-11.pdf, Abruf: 02.04.2011

[Ian09] IANNELLA, Renato: *ODRL V2.0 - XML Encoding, Working Draft: 12 February 2010*. Online Ressource. http://odrl.net/2.0/WD-ODRL-XML-20100212.html. Version: Februar 2009, Abruf: 30.03.2011

[Jah10] JAHN, Nico: *ODRL-Mediathek zur Unterstützung von Usage Rights Management*. Studienarbeit, 2010

[JBo] JBoss Community: *About Hibernate*. Online Ressource. http://www.hibernate.org/about, Abruf: 12.04.2011

[KBA+11] KING, Gavin ; BAUER, Christian ; ANDERSEN, Max R. ; BERNARD, Emmanuel ; EBERSOLE, Steve ; FERENTSCHIK, Hardy ; RED HAT, INC. (Hrsg.): *Hibernate Reference Documentation*. 3.6.3.Final. : Red Hat, Inc., 2011. http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/, Abruf: 26.04.2011

[Lie10] LIESENFELD, Verena: *Entwicklung eines lizenzbewussten P2P-Clients zum Austausch von URM-basierten Dateien*, Universität Koblenz-Landau, Campus Koblenz, Diplomarbeit, 2010. http://kola.opus.hbz-nrw.de/volltexte/2010/527/

[Mül09] MÜLLER, Eugen: *Winamp Plugin „Private Licence Generator“*. Studienarbeit, 2009

[Pic05] PICOT, Prof. Dr. A.: Digital Rights Management - ein einführender Überblick. In: (PT05), S. 1 – 14

[PT05] PICOT, Arnold (Hrsg.) ; THIELMANN, Heinz (Hrsg.): *Distribution und Schutz digitaler Medien durch Digital Rights Management*. Berlin : Springer, 2005. – ISBN 9783540238447

[Sch11] SCHLÜNSS, Florian: *Medienmanager und GUI für TURM*. 2011. – Bachelorarbeit

[Wor05] World Wide Web Consortium: *Document Object Model (DOM)*. Online Ressource. http://www.w3.org/DOM/. Version: January 2005, Abruf: 19.04.2011