UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

# Entwicklung eines lizenzbewussten P2P-Clients zum Austausch von URM-basierten Dateien

Diplomarbeit

zur Erlangung des Grades einer Diplom-Informatikerin
vorgelegt von

Verena Liesenfeld

| | |
|---|---|
| Erstgutachter: | Prof. Dr. Rüdiger Grimm<br>Institut für Wirtschafts- und Verwaltungsinformatik |
| Zweitgutachter: | Dipl.-Inform. Daniel Pähler<br>Institut für Wirtschafts- und Verwaltungsinformatik |
| Weitere Betreuer: | Dipl.-Ing. Helge Hundacker<br>Institut für Wirtschafts- und Verwaltungsinformatik |

Koblenz, im Mai 2010

# 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)

# Danksagung

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich bei der Erstellung dieser Arbeit unterstützt haben.

Ich danke Prof. Dr. Rüdiger Grimm dafür, dass er mir ermöglicht hat, diese Arbeit in seiner Forschungsgruppe über ein spannendes, aktuelles Thema zu schreiben. Ein besonderer Dank gilt meinen Betreuern Daniel Pähler und Helge Hundacker, die stets für mich ansprechbar waren und mit guten Ideen und unermüdlichem Einsatz meine Diplomarbeit betreut haben.

Vor allem möchte ich mich bei meinem Freund Christoph Adolphs für eine wundervolle Unterstützung und den Mut den er mir zu jeder Zeit gemacht hat bedanken. Mein ganz besonderer Dank gilt abschließend meinen Eltern, die mir das Studium erst ermöglicht haben, mir stets helfend zur Seite standen und mir den nötigen Rückhalt vermittelten.

# Inhaltsverzeichnis

# Abkürzungsverzeichnis

| | |
|---|---|
| API | Application Programming Interface |
| CC | Creative Commons |
| CD | Compact Disk |
| CLI | Command Line Interface |
| CUP | Client for URM based P2P filesharing |
| DRM | Digital Rights Management |
| GNU | 'GNU is not Unix' |
| GPL | General Public License |
| IP | Internet Protocol |
| JVM | Java Virtual Machine |
| JXTA | Juxtapose |
| MP3 | MPEG-1 Audio Layer 3 |
| ODRL | Open Digital Rights Language |
| OECD | Organisation for Economic Co-operation and Development |
| OMA | Open Mobile Alliance |
| OMG | Object Management Group |
| OOP | Objektorientierte Programmierung |
| OSD | Open Source Definition |
| OSI | Open Source Initiative |
| P2P | Peer-to-Peer |
| REL | Rights Expression Language |
| TURM | Toolkit for URM |
| UML | Unified Modeling Language |
| UrhG | Urheberrechtsgesetz |
| URM | Usage Rights Management |

W3C ..... World Wide Web Consortium
XCP ..... Extended Copy Protection
XML ..... eXtensible Markup Language
XrML ..... eXtensible rights Markup Language

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Listings

# 1 Einleitung

Zum Zeitpunkt der Erstellung der Diplomarbeit wird in der Forschungsgruppe „IT-Risk-Management“ von Prof. Dr. Rüdiger Grimm an der Universität Koblenz-Landau eine neue Methode zur Unterstützung der Einhaltung des Urheberrechts entwickelt. Der als Usage Rights Management (URM) bezeichnete Ansatz soll Benutzer über ihre Nutzungsrechte an den, auf dem eigenen Rechner vorhandenen, digitalen Mediendateien informieren. Ein weiterer Bestandteil des URMs ist die Unterstützung des Nutzers mit den Dateien rechtskonform umzugehen. URM[1] entstand im Kontext des Projekts „SOAVIWA[2]“ und wird von der Stiftung Rheinland-Pfalz für Innovation gefördert. Ein Projektpartner ist „The ODRL Initiative[3]“.
Zu dem Thema Usage Rights Management wurde bereits im Jahre 2009 von Professor Dr. Rüdiger Grimm und seinen Mitarbeitern Helge Hundacker und Daniel Pähler eine Grundlagenpublikation erstellt [Hundacker u. a., 2009]. Diese Veröffentlichung wurde im September 2009 beim „7. Internationalen Workshop für Technologie, Wirtschaft und rechtsgültigen Aspekten von virtuellen Waren, vereinigt mit dem 5. Internationalen ODRL Workshop[4]“ in Nancy, Frankreich vorgestellt. Des Weiteren

[1] Usage Rights Management (URM); URL: http://urm.iwvi.uni-koblenz.de/; Stand: 23.03.2010

[2] SOAVIWA: Eine Service-orientierte Absicherung von virtuellen Waren; URL: http://www.uni-koblenz-landau.de/koblenz/fb4/institute/iwvi/aggrimm/projekte/SOAVIWA; Stand: 23.03.2010

[3] The ODRL Initiative; URL: http://odrl.net/; Stand: 23.03.2010

[4] Virtual Goods; URL: http://www.virtualgoods.org/2009/; Stand: 14.04.2010

wurde die Thematik im März 2009 und 2010 auf der CeBIT[5] präsentiert (siehe dazu auch das entsprechende Plakat in Abbildung 3.8).

## 1.1 Motivation

Seit der Entstehung und Entwicklung neuer Technologien für die Speicherung und den Transport von Informationen haben sich die medienwirtschaftlichen Rahmenbedingungen in Bezug auf die Verbreitung von Medien bedeutend verändert [Gensch u. a., 2009a]. Inhalte, wie z.B. Musik, Filme und Software, die nicht an physische Medien gebunden sein müssen, sondern digital im Internet bereitgestellt werden können [Stryszowski u. Scorpecci, 2009], werden seit dem Strukturwandel, den [Gensch u. a., 2009a] beschreibt, immer beliebter.
Bereits im Jahr 1982/83 erfolgte mit der Einführung der Compact Disk (CD) durch Philips und Sony nach [Gensch u. a., 2009a] die erste entscheidende Weichenstellung der digitalen Inhalte. Diese Weichenstellung etablierte sich unter anderem mit den folgenden Erscheinungen (in Anlehnung an den Ansatz von [Haring, 2002]):

1993 World Wide Web
1994 Internetbrowser Netscape Navigator
1994 Datenkomprimierungsformat MP3
1999 Musiktauschbörse Napster
2000 Downloadportal iTunes
2001 portables Abspielgerät iPod
2007 Smartphone iPhone
2008 Downloadportal App Store

Die Zeitleiste zeigt, dass sich immer neuere Technologien und Plattformen durchgesetzt haben. Die Vorteile der Käufer in Bezug auf digitale Medien sind durch die schnelle Verfügbarkeit und die Ungebundenheit an

[5] CeBIT 2010 Computermesse; URL: http://www.cebit.de; Stand: 23.03.2010

einen Standort gekennzeichnet. Die Verkäufer haben Kosteneinsparungen zu verzeichnen, wie z.B. bei der Produktion der Medienträger, dem Vertrieb oder der Distribution der Ware. Darüber hinaus ist es den Käufern möglich, von der legal erworbenen digitalen Kopie weitere Duplikate ohne Qualitätsverlust zu erstellen und am Computer zu bearbeiten. Diese Duplikate dürfen ebenfalls laut Schrankenregelung des Urheberrechtsgesetzes (UrhG) § 53 [UrhG, 2008] für private Zwecke genutzt werden (siehe Kapitel 2.1.2). Im Gegensatz dazu haben private Mitschnitte auf analogen Datenspeichern wie z.B. einer sogenannten Kassette (Compact Cassette; 1963 von Philips eingeführt [Daniel u. a., 1999]) oder einer Vinylschallplatte (1948 von Dr. Peter Carl Goldmark erfunden [Gensch u. a., 2009b]), durch die hohe Störempfindlichkeit immer einen Qualitätsverlust zu verzeichnen [Müller-Kalthoff, 2002]. Nur mit hohem Aufwand und weiteren Qualitätsverlusten können Audioaufnahmen auf analogen Datenträgern bearbeitet werden.

Digitale Inhalte bringen jedoch nicht nur positive Aspekte hervor. Durch diese digitalisierten Inhalte wird das Herstellen und Verbreiten einer rechtswidrigen Kopie eines urheberrechtlich geschützten Mediums erleichtert. Dabei ist zu beachten, dass es sich in diesen Fällen nicht um laut UrhG erlaubte Mitschnitte für private Zwecke handelt (siehe § 53 [UrhG, 2008] bzw. Kapitel 2.1.2). Mit einfachen Mitteln ist es möglich, ein-zu-eins Kopien (ohne Qualitätsverlust) von geschützten digitalen Inhalten anzufertigen. Bei Verbreitung der Kopien gelten diese nicht mehr als Privatkopien. Das bedeutet, dass diese Duplikate gegen das Urheberrecht verstoßen. Die Bezahlung des Urhebers oder des Rechteinhabers, die beim Kauf einer legalen Kopie erfolgt wäre, unterbleibt [UrhG, 2008].
Diese sogenannten Schwarzkopien von Medien werden heutzutage von vielen Personen kostenfrei über das Internet zum Download angeboten. Es gibt genug Abnehmer, die solche unentgeltlichen Angebote in Anspruch nehmen, was die Urheberrechtsinhaber und die politischen Entscheidungsträger im Kampf gegen den Datendiebstahl vor neue Herausforderungen stellt [Stryszowski u. Scorpecci, 2009]. Diesem Problem

widmet sich unter anderem die Organisation für wirtschaftliche Zusammenarbeit und Entwicklung (OECD Organisation for Economic Cooperation and Development). Diese Einrichtung ist ein einzigartiges Forum, in dem die Regierungen von 30 demokratischen Staaten zusammenarbeiten, um die wirtschaftlichen, sozialen und ökologischen Herausforderungen der Globalisierung anzugehen. Mit der Studie „Piracy of digital Content“ [Stryszowski u. Scorpecci, 2009] stellt die OECD dar, warum digitale Piraterie betrieben wird und welche Gefahren in Bezug auf digitale Inhalte bestehen. Weiterhin werden Gegenmaßnahmen zur Verhinderung von Urheberrechtsverletzungen beschrieben, die eingeleitet werden können. Dazu gehören „Industrie“-Initiativen, die als Organisationen auftreten und der digitalen Piraterie entgegenarbeiten. Die Urheberrechtsinhaber werden beispielsweise durch die Entwicklung und Vorstellung von Technologien zur Bekämpfung des Datendiebstahls unterstützt. Des Weiteren werden institutionelle Rechtsmittel wie beispielsweise eine einstweilige Verfügung und die Zuerkennung von Schadensersatz für den Urheberrechtsinhaber eingesetzt.

Der Handel von Schwarzkopien wird häufig über sogenannte Tauschbörsen vollzogen, die die Peer-to-Peer (P2P) Technologie (siehe dazu auch Kapitel 2.2.2) verwenden. Dadurch können die Medienkopien direkt von Endbenutzer zu Endbenutzer ausgetauscht werden. Über solche P2P-Filesharing-Systeme, wie z.B. LimeWire oder eMule, werden nicht nur illegale Medien ausgetauscht. Auch legale Inhalte werden ausgetauscht, welche der Benutzer jedoch nicht auf Anhieb von den illegalen Dateien unterscheiden kann. Die Motivation dieser Diplomarbeit ist es also, den Endbenutzer dabei zu unterstützen sich beim Austausch von Dateien legal zu verhalten. Dies geschieht durch die Entwicklung einer P2P-Tauschbörse, die den Benutzer dabei unterstützt nur seine legalen Kopien zu verbreiten und selbst nur legale Kopien erwerben zu können.

## 1.2 Zielsetzung

Das Ziel der vorliegenden Diplomarbeit ist es, ein System zum Datenaustausch zu entwerfen und prototypisch umzusetzen. Dieses System soll den Benutzern die Möglichkeit geben, sowohl unter URM stehende digitale Mediendateien von anderen Personen einsehen und downloaden zu können als auch eigene URM-Dateien anderen Benutzern zur Verfügung stellen zu können. Diese Anforderungen sollen mittels eines lizenzbewussten P2P-Client umgesetzt werden (siehe [Hundacker u. a., 2009]).

Der Client wird im Zuge der Arbeit durch das Konzipieren und Implementieren einer Javaanwendung (siehe Kapitel 2.3.4) realisiert. Den Benutzern soll ermöglicht werden, sich über den Lizenzstatus der vorhandenen digitalen Mediendateien zu informieren und nur als rechtsgültig gekennzeichnete Dateien auszutauschen. Die digitalen Dateien, die unbegrenzt weitergegeben werden dürfen, können mit allen angemeldeten Personen ausgetauscht werden. Die Mediendateien die nur begrenzt weitergegeben werden dürfen, werden nur den in der Freundesliste eingetragenen Benutzern zur Verfügung gestellt.

Das Ziel der Arbeit wird durch die Erstellung eines Peer-to-Peer Netzwerkclients realisiert, durch den Rechner mit Rechner kommunizieren und untereinander Dateien austauschen können. Dabei werden die Richtlinien des Usage Rights Managements (URM) zugrunde gelegt, um den Benutzern eine Plattform zu bieten, legal Dateien austauschen und Informationen über den Weitergabestatus der entsprechenden Dateien erhalten zu können.
Des Weiteren soll der URM-P2P-Client (im Folgenden auch als CUP (**C**lient for **U**RM based **P**2P filesharing) bezeichnet) an das zu Beginn der Diplomarbeit im Aufbau befindliche URM-System TURM (Toolkit for URM) angegliedert werden. Damit wird der CUP als TURM-Komponente umgesetzt. Der CUP regelt zum einen die Kommunikation zwischen den verschiedenen Nutzern und zum anderen die Verwaltung und Verwendung von URM-konformen Lizenzen (z.B. das Auslesen, Erzeugen und

Weiterleiten der Lizenzen). Für den Umgang mit den Lizenzen soll der in das TURM-System integrierte Lizenzmanager als Schnittstelle verwendet werden. Neben der Schnittstelle zur Lizenzverwaltung sollen im CUP über eine Anbindung an den TURM weitere Hilfsmittel des TURM-Systems zentral verwendet werden (z.B. der Konfigurationsmanager oder die Medienverwaltung).

Als Benutzerschnittstelle für den Prototypen wird ein Kommandozeilen-Interface (CLI Command Line Interface, siehe auch Kapitel 2.3.2) verwendet, womit die vorhandenen Methoden überprüft werden können.

Dementsprechend soll am Ende dieser Diplomarbeit ein funktionierender P2P-Client existieren, der auf Basis des URMs rechtsgültig handelt und somit nur Dateien mit gültigen Lizenzen austauscht. Ebenfalls wird dieser an definierten Stellen die bereits vorhandenen und die im Aufbau befindlichen Funktionen des TURMs der Forschungsgruppe „IT-Risk-Management“ nutzen und dort als Komponente integriert. In dem lizenzbewussten P2P-Netzwerk sollen sich alle Personen anmelden können, die mit anderen Personen legal digitale Medien austauschen und ihren Freunden Mediendateien mit begrenztem Weitergabestatus zur Verfügung stellen möchten.

## 1.3 Hypothese

Eine wichtige Fragestellung dieser Diplomarbeit ist, inwieweit die Datenübertragung über ein P2P-Netzwerk mit dem CUP umgesetzt werden kann. Es stellt sich die Frage, welche Möglichkeiten das gewählte P2P-Framework JXTA in dem vorgegebenen Zeitrahmen der Arbeit bietet. Diese Fragestellung ist schwierig und zu Beginn der Arbeit nur spekulativ zu beantworten, da erst während der CUP-Konzipierung das benötigte Wissen über das JXTA-Framework gesammelt werden kann. Da die Konzipierung und Implementierung einer Netzwerkkommunikation die Grundlage für darauf aufbauende Dienste darstellt, liegt die Vermutung nahe, dass die P2P-Infrastruktur des CUPs einen Schlüsselfaktor dieser Arbeit

darstellen wird.
Basierend auf der Netzwerkimplementierung stellt sich die weitere Frage, ob sich das URM-Konzept auf der P2P-Infrastruktur des CUPs umsetzen lässt. Da die Grundidee des URM-Konzepts die Verwaltung von Nutzungsrechten ist und diese in Form von Daten repräsentiert sind, wird es darauf hinauslaufen, dass lediglich weitere Daten über das P2P-Netzwerk übertragen werden müssen.

## 1.4 Forschungsmethodik und Thematische Einordnung

Folgende Forschungsmethoden werden in dieser Diplomarbeit eingesetzt (in Anlehnung an den Klassifizierungsansatz von [Braun u. a., 2004] und [Vaishnavi u. Kuechler, 2004]):

Empirische Methoden:

- Literatur- und Internetrecherche
- Anforderungsanalyse

Konstruktive Methoden:

- Design Research
- Entwicklung und Test eines Prototypen

Bei der **empirischen Forschung** wurden im Bereich der *Anforderungsanalyse* grundsätzlich die in [Grechenig u. a., 2009] beschriebenen Ansätze (u.a. Anwendungsfälle, User Stories und Geschäftsprozessmodellierung) angewandt. Dabei geht es darum, die von einem Betroffenen (Stakeholder) benötigten Funktionen und die vorhandenen Einschränkungen eines Systems herzuleiten. Für die Analyse und Beschreibung der Anforderungen wurde sich in dieser Arbeit für die Methode der Anwendungsfälle entschieden, auf welche in dem späteren Kapitel 4 detailliert eingegangen wird.

Die in dieser Diplomarbeit angewandte **konstruktive Methode** der Prototypentwicklung verfolgt den im Weiteren beschriebenen Ansatz des *Design Researchs*. Beim nutzenorientierten Ansatz des Design Researchs wird ein Produkt entwickelt, das universell für ein bestimmtes Problem steht und eine effizientere oder neuartige Lösung offeriert [Vahidov, 2006]. Das Vorgehen beim Design Research wird nach [Takeda u. a., 1990] in fünf Phasen unterteilt, welche in Abbildung 1.1 dargestellt sind und im Folgenden erläutert werden.

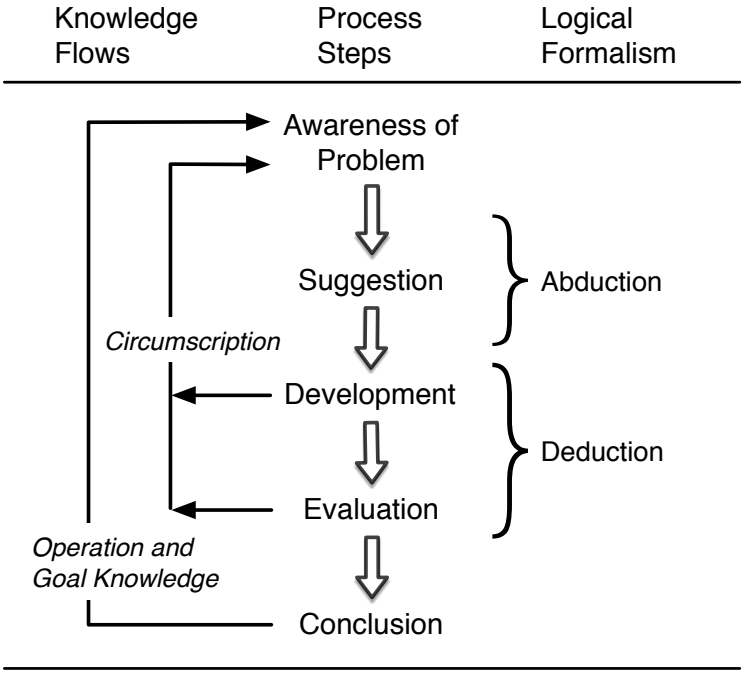


Abbildung 1.1: Allgemeine Methodik des Design Researchs
in Anlehnung an [Takeda u. a., 1990] und [Vaishnavi u. Kuechler, 2004]

Die in Abbildung 1.1 skizzierten Phasen des Design Research Ansatzes sind:

Problemerkennung ⇒ Vorschläge ⇒ Entwicklung ⇒ Bewertung ⇒ Fazit

Es wird zunächst untersucht, welches Problem vorliegt. Danach werden Lösungsvorschläge überdacht, welche die Produktentwicklung vorantreiben und das Produkt wird entwickelt bzw. hergestellt. Am Ende wird das

Produkt bewertet und es werden abschließende Schlussfolgerungen gezogen.
Die Pfeile stellen dar, dass es sowohl während der Entwicklungsphase als auch während der Bewertung des Produktes dazu kommen kann, dass die Problemstellung erweitert werden muss (da z.B. neue Probleme erkannt wurden), so dass der Prozess des Design Researchs mit der modifizierten Problemstellung neu begonnen wird. Dieser Vorgang verdeutlicht die problemlösende bzw. leistungssteigernde Charakteristik dieser Forschungsmethodik.

Die vorliegende Diplomarbeit ist thematisch als Implementierungsarbeit einzuordnen, da sich ein großer Teil der Softwareentwicklung widmet und softwaretechnische Grundprinzipien der Programmierung zugrunde liegen. Die Entwicklung eines lizenzbewussten Peer-to-Peer Clients steht ebenso im Vordergrund wie die Ausführungen und Erläuterungen zu vorhandenen Protokollen (siehe Kapitel 2.3.2) und Frameworks (siehe Kapitel 2.3.2). Die Arbeit setzt sich außerdem sowohl im theoretischen als auch im praktischen Teil mit der Übertragung von Informationen über ein Netzwerk auseinander.

## 1.5 Vorgehensweise und Struktur

Folgende Herangehensweise wird in dieser Diplomarbeit angestrebt (in Anlehnung an den Objektorientierten Softwareentwicklungsansatz von [Pichocki, 2002] und [Boles, 2005]):

**Literatur- und Internetrecherche**
Um einen Überblick über die für den P2P-Client sinnvollen Frameworks (siehe Kapitel 2.3.2) und Javatechnologien zu bekommen, werden diese zunächst durch Literatur- und Internetrecherche untersucht.

**Problembeschreibung und Anforderungsanalyse des P2P-Clients**
Im nächsten Schritt wird eine grobe Spezifikation des lizenzbewussten P2P-Clients erstellt und zusätzlich Anforderungen an den Client

analysiert. Diese Anforderungen werden in einer Liste festgehalten, so dass detailliert vorliegt, was der lizenzbewusste P2P-Client leisten soll.

**Objektorientierte Analyse**
In der Analysephase werden zunächst die Anwendungsfälle der Problembeschreibung und der Anforderungen, deren Akteure sowie ihre Beziehungen untereinander als konzeptionelles Modell herausgestellt (Analyse des Problembereichs). Das entstehende Modell wird anhand eines UML-Verhaltensdiagramms visualisiert. Für weitere Informationen zu UML siehe Kapitel 2.3.3.

**Objektorientierter Entwurf**
Das in der objektorientierten Analyse entstandene Modell wird in dieser Phase verfeinert und dient als konkrete Vorbereitung der Implementierung. Dazu werden die Anwendungsfälle um weitere Details ergänzt und anhand eines UML-Klassendiagramms die benötigten Klassen bzw. Objekte, deren Attribute und Methoden sowie ihre Beziehungen untereinander aufgezeigt. Dabei werden ebenfalls Einzelheiten hinzugefügt, die für die objektorientierte Analyse nicht relevant waren, jedoch bei der Implementierung notwendig sind, wie z.B. Klassen zur Speicherung von Objekten (Abbildung des Analysemodells auf den Lösungsraum). Weiterhin kann in dieser Phase auf sogenannte Entwurfsmuster (Design Patterns; siehe Kapitel 2.3.2) zurückgegriffen werden, um bereits etablierte Lösungen nicht neu finden zu müssen. Ziele dieser Phase sind vor allem die Wartbarkeit und Wiederverwendbarkeit. Zur Darstellung des hier entstehenden Entwurfsmodells, können alle UML-Diagrammtypen eingesetzt werden.

**Objektorientierte Implementierung**
In diesem Schritt wird das objektorientierte Entwurfsmodell in der Programmiersprache Java (siehe Kapitel 2.3.4) umgesetzt. Hier entscheidet sich z.B. welche Klassen durch vorhandene Java-Klassenbibliotheken bereits abgebildet werden und statt implementiert direkt verwendet werden

können. Weiterhin stellt sich heraus, welche Frameworks (siehe Kapitel 2.3.2) eingesetzt werden können.

**Objektorientierter Test**
Zum Testen der Funktionen des Peer-to-Peer-Clients wird ein Kommandozeilen-Interface (CLI, siehe Kapitel 2.3.2) eingesetzt, über das die vorhandenen Methoden auf ihre Funktion und Richtigkeit überprüft werden können.

Der Aufbau und die Struktur der vorliegenden Arbeit richtet sich schwerpunktmäßig nach der gerade beschriebenen Vorgehensweise. Zunächst wird nach der **Einleitung** (ab Seite 1), deren Bereiche Motivation, Zielsetzung, Hypothese und Forschungsmethodik bereits behandelt wurden, der Bereich der **Grundlagen** (ab Seite 13) beschrieben. Dieser dient zusammen mit dem darauf folgenden Abschnitt **URM** (ab Seite 46) als Basis für das weitere Verständnis. Im Hauptteil, der sich aus den Bereichen **Analyse und Entwurf**, **Implementierung** (ab Seite 83) und **Anwendungsaspekt** (ab Seite 108) zusammensetzt, wird im Detail auf die Entwicklung und den Nutzen des CUPs eingegangen. Zum Abschluss wird ab Seite 112 in einem **Fazit** die erarbeitete Arbeit zusammengefasst und in einem **Ausblick** auf die Zukunftsvarianten eingegangen.

Um eine strukturierte Softwareentwicklung für diese Diplomarbeit zu gewährleisten, wurde sich für das Wasserfallmodell [Wazeck, 2003] entschieden. Dieses Vorgehensmodell ist ein Phasenmodell, das den gesamten Entwicklungsprozess in mehrere Phasen zerlegt. Das Wasserfallmodell des CUPs ist in Abbildung 1.2 veranschaulicht.
Bei diesem Modell wird jede Phase einzeln durchlaufen, das bedeutet, erst wenn eine Phase abgeschlossen ist, wird mit der nächsten Phase begonnen. Da das „reine“ Wasserfallmodell keine Rücksprünge zu der jeweils vorherigen Phase zulässt, wird in dieser Diplomarbeit das „Wasserfallmodell mit Rückkopplung“ angewandt (siehe gestrichelte Pfeile in

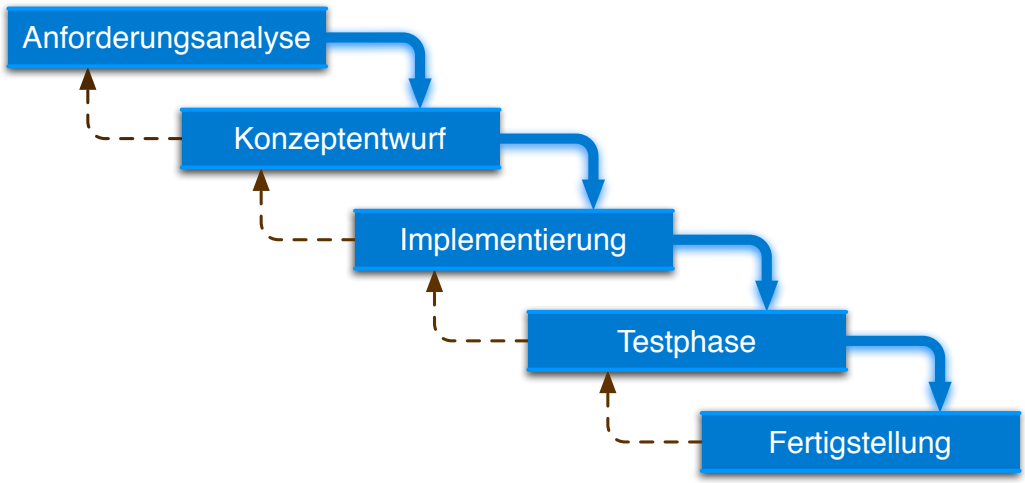


Abbildung 1.2: Wasserfallmodell des CUPs

Abbildung 1.2). Durch die Rückkopplung ist es möglich, von der aktuellen Phase zurück in eine vorherige Phase zu springen. Der Vorteil liegt darin, dass wenn z.B. erst während des Konzeptentwurfs auffällt, dass eine notwendige Anforderung nicht formuliert wurde, kann zurück in die Phase der Anforderungsanalyse gesprungen werden, um die fehlende Anforderung hinzuzufügen. Hier zeigt sich auch ein Nachteil dieser Form. Denn bei jedem Rücksprung und der jeweiligen Änderung einer bereits abgeschlossenen Phase, müssen die bereits bearbeiteten nachfolgenden Phasen dementsprechend angepasst werden. Jedoch überwiegt in diesem Vorhaben der Vorteil der Rückkopplung.

# 2 Grundlagen

In diesem Kapitel werden grundlegende, kapitelübergreifende Begriffe erläutert, die für die Betrachtung des Themas von Bedeutung sind. Es werden allgemeine Begriffsbestimmungen und Begriffe der Teilbereiche Netzwerktechnik und Softwareentwicklung ausgeführt, die wesentlich zum Verständnis der weiteren Diplomarbeit beitragen und zusammen mit dem darauffolgendem Kapitel 3 die Grundlage dieser Arbeit bilden. Die nur für einzelne Abschnitte wichtigen Begriffe werden in den entsprechenden Kapiteln behandelt.

## 2.1 Grundbegriffe

Allgemeine Bestimmungen wie digitale Inhalte, das Urheberrecht und Digital Rights Management, die für die Thematik dieser Diplomarbeit als Basiswissen vorausgesetzt sind, werden in diesem Abschnitt beschrieben.

### 2.1.1 Digitale Inhalte

Unter digitalen Inhalten werden Inhalte wie Musik, Text, Bild und Software verstanden, die digitalisiert wurden [Grimm, 2006]. In Abbildung 2.1 sind verschiedene Medienformen dargestellt und es wird im Folgenden ein Überblick über die entsprechende traditionelle und digitale Produktion und Distribution gegeben.
Musik wird traditionell auf Platten oder CDs produziert und verkauft oder auf Konzerten vorgeführt, während Musik im digitalen Sektor durch die verschiedenen vorhandenen Formate (wie z.B. MP3 oder AAC)

vertreten ist. Die Distribution von Filmen erfolgt z.B. über das Fernsehen, im Kino oder durch Videos. Im elektronischen Bereich können Filme über verschiedene Quellen (z.B. Webshops oder Tauschbörsen) einfach auf den eigenen Computer heruntergeladen werden und sind nicht immer legal. Weiterhin werden die Unterschiede bei der Vervielfältigung deutlich. Beispielsweise können digital zur Verfügung stehende Fotos durch wenig Aufwand kopiert werden, wobei die traditionelle Reproduktion unter Verwendung von Fotonegativen einen gewissen Aufwand benötigt.

**Musik**

| | Traditionell | Digital |
|---|---|---|
| Produktion | Konzert, Platten, CDs | MPEG-4, AAC, MP3 |
| Distribution | Aufführung, Notenverkauf, Unterricht | Download, Filesharing |
| Reproduktion | Noten, Hausmusik, Tonband, Kassette | Copy and Paste, CDs brennen |

**Film**

| | Traditionell | Digital |
|---|---|---|
| Produktion | Studio, Zelluloid, digital | MPEG-4, AVC |
| Distribution | Wertekette: Kino, TV, Video | Download, Filesharing |
| Reproduktion | Physische Kopiermedien | Digitalcam, Copy and Paste, Dateien |

**Bilder**

| | Traditionell | Digital |
|---|---|---|
| Produktion | Fotografie, Handzeichnung | JPEG, PNG, SVG, EPS, RAW |
| Distribution | Fotostudios | Download |
| Reproduktion | Fotonegativ, Papierkopie, Abzeichnen | Digitale Kopie |

**Bücher (E-Book)**

| | Traditionell | Digital |
|---|---|---|
| Produktion | Druck | PDF, TXT, HTML |
| Distribution | Buchläden | Amazon, P2P, Web-Direktzugang (PDF) |
| Reproduktion | Druck, Papierkopie | Digitale Kopie, Datei |

**Spiele**

| | Traditionell | Digital |
|---|---|---|
| Produktion | Kreation, Verlag, Spielkasten | Multimedia, Internetinteraktion |
| Distribution | Spielzeugläden | Download |
| Reproduktion | Physische Fabrikation | Integrierte Steuerung schwer zu kopieren |

**Software**

| | Traditionell | Digital |
|---|---|---|
| Produktion | Programmieren | Programmieren |
| Distribution | Software-Vertrieb, CDs | Download |
| Reproduktion | CD, Digitale Kopie, Datei | Digitale Kopie, Datei |

Abbildung 2.1: Digitale Inhalte
in Anlehnung an [Grimm, 2006] und [Grimm u. Nützel, 2002]

Die digitalen Inhalte ermöglichen nach [Grimm, 2006] und [Nützel, 2006] eine neue, virtuelle Struktur von Waren. Diese sogenannten virtuellen Waren besitzen die Eigenschaft, nicht mehr an ihr physikalisches Medium (CD, DVD) gebunden zu sein und ohne Verluste von einem Medium auf ein anderes übertragen werden zu können [Grimm u. Nützel, 2002]. Daraus ergibt sich für [Grimm u. Nützel, 2002], dass die Verfügbarkeit nicht mehr in der Hand des Produzenten liegt. Die Konsumenten können auf die Verfügbarkeit einwirken, da jede Kopie ein Original ist und das Vervielfältigen nicht begrenzbar ist, wie es bei analogen Inhalten durch den eintretenden Verlust (bei Vervielfältigung) der Fall ist. Weiterhin ist zu beachten, dass durch diese neue Struktur unerwünschtes Verhalten, wie das Anfertigen von rechtswidrigen Kopien, unterstützt wird.

Ein sich herausstellendes Leitbild der virtuellen Waren ist das MP3-Format, welches sich im Musikbereich durchgesetzt hat (siehe Abbildung 2.1). Auf einfache Weise können MP3-Dateien auf dem Computer kopiert und vervielfältigt werden und wegen ihrer exzellenten und trotz Kompression gleichbleibenden Qualität bei geringer Dateigröße effizient im Internet verteilt werden [Grimm, 2006]. Die virtuellen Waren stellen durch ihre günstige Reproduktion und Distribution eine ökonomische Konkurrenz zu ihren traditionellen physischen Trägern dar [Grimm, 2006].

Die Bezeichnung „digitale Inhalte“ wurde bewusst gewählt, da die nach [Grimm u. Nützel, 2002] gebräuchliche Bezeichnung „virtuelle Waren“ durch den Begriff Waren zusätzlich zu den beschriebenen Eigenschaften den Aspekt der Preiserzielung mit einschließen würde [Alisch u. a., 2005]. Damit jedoch keine unnötigen Missverständnisse auftreten, werden in der vorliegenden Arbeit die Begriffe „digitale Inhalte“, „digitale Medien“ und „virtuelle Waren“ synonym verwendet.

### 2.1.2 Deutsches Urheberrecht

Das Urheberrecht der Bundesrepublik Deutschland schützt auf der einen Seite den Urheber und Rechteinhaber von geistigen Eigentums vor

Missbrauch der eigenen Inhalte und räumt auf der anderen Seite der Allgemeinheit, also den Nutzern, Sonderrechte ein (Festlegung der Schranken (Abschnitt 6 [UrhG, 2008]). Beispielsweise dürfen für den privaten Gebrauch (§53 [UrhG, 2008]), für die Lehre (§47 [UrhG, 2008]) und das wissenschaftliche Arbeiten (§52a [UrhG, 2008]) Inhalte ohne Nachfragen verwendet werden, wodurch die Ausschließlichkeit beschränkt wird. Niedergeschrieben sind diese Rechte im „Urheberrechtsgesetz“ (UrhG; [UrhG, 2008]). Das UrhG wurde 1965 laut [Dreier u. Nolte, 2003] als Reaktion auf das Kopieren geistigen Eigentums eingesetzt, um die folgenden Punkte zu realisieren (in Anlehnung an [Lutz, 2009]):

- die Stärkung der Urheber, in dem die persönlichen Interessen wie Anerkennung und Einkommen gewahrt bleiben.
- die Gewährleistung der wirtschaftlichen Belange der Produzenten wie Gewinn, Kostenschutz und Amortisation.
- die Steuerung der Bedingungen zur Herstellung, Verbreitung und Nutzung.

Des Weiteren wurde beispielsweise die Schutzfrist auf 70 Jahre nach dem Tode des Urhebers verlängert (§64 [UrhG, 2008]). Weiterhin wurden sogenannte Pauschalabgaben eingeführt, so dass bereits beim Kauf eines Gerätes, das mit hoher Wahrscheinlichkeit für Vervielfältigungen von urheberrechtlich geschützten Werken genutzt wird, eine angemessene Vergütung mit bezahlt werden muss (siehe §54 [UrhG, 2008]).

Da der CUP in der Regel auch länderübergreifend verwendet werden kann, muss hierzu die internationale Rechtslage bezüglich des Urheberrechts bestimmt werden. Die detaillierte Betrachtung internationaler Rechtslagen ist nicht Gegenstand dieser Arbeit, weshalb sich dieser Abschnitt nur mit dem Urheberrecht der Bundesrepublik Deutschland auseinandersetzt.

### 2.1.3 Digital Rights Management - DRM

Die digitale Rechteverwaltung (DRM) beschreibt Mechanismen, durch die Vermarktungs- und Urheberrechte an digitalen Inhalten gesichert werden und sowohl die Nutzung als auch die Verbreitung dieser Medien teilweise kontrolliert werden kann [Arlt, 2006]. Anbieter haben durch Digital Rights Management Systeme (DRM-Systeme) die Möglichkeit, ihre Daten einer Nutzungskontrolle zu unterziehen. Dadurch können neue Abrechnungsverfahren eingesetzt werden, bei denen z.B. anhand von Lizenzen die Nutzungsrechte an Daten die Bezahlgrundlage bilden, anstatt sich die Daten selbst vergüten zu lassen [Gossmann, 2007].

Derzeit werden DRM-Systeme überwiegend bei Musik und Filmen verwendet. Beispiele für solche DRM-Systeme sind in folgender Tabelle 2.1 dargestellt. Dazu gehören *FairPlay* von Apple, *OMA DRM* von der Organisation Open Mobile Alliance (OMA) und *Windows Media DRM* von Microsoft.

| | **FairPlay** | **OMA DRM** | **Windows Media DRM** |
|---|---|---|---|
| **verwendet von** | iTunes | Multimedia-Handys | musicload |
| **entwickelt von** | Apple | Open Mobile Alliance (OMA) | Microsoft |
| **Rechte-definitions-sprache** | nicht bekannt | ODRL | XrML |

Tabelle 2.1: Beispiele für DRM-Systeme
nach [Schmidt u. a., 2004] und [Fechner, 2007]

FairPlay findet im Multimediaprogramm *iTunes*[1] und den Mobilgeräten iPod und iPhone Anwendung. OMA DRM wird in *Multimedia-Handys* zum

[1] iTunes; URL: http://www.apple.com/de/itunes/; Stand: 23.03.2010

einen für Bilder und Klingeltöne, zum anderen für mobile Musik- und Fernsehübertragungen verwendet. Windows Media DRM wird z.B. in dem Musikonlineshop *musicload*[2] eingesetzt. [Schmidt u. a., 2004] [Fechner, 2007]

Für die Implementierung von Nutzungsregeln in die Medien werden von den DRM-Systemen sogenannte Rechtedefinitionssprachen (REL) verwendet, die es erlauben, den Umfang der eingeräumten Rechte zu beschreiben und von Wiedergabegeräten interpretiert und ausgeführt werden können [Fechner, 2007]. Anhand dieser Sprachen können verschiedene Aspekte, wie z.B. Gültigkeitsdauer der Lizenz oder Anzahl der Abspielvorgänge definiert werden. Dabei gibt es zwei verbreitete Sprachen, ODRL (Open Digital Rights Language) und XrML (eXtensible rights Markup Language) [Nützel, 2006]. ODRL ist eine offene und standardisierte Sprache, die, wie in Tabelle 2.1 dargestellt, unter anderem von dem DRM-System OMA DRM eingesetzt wird, um eine möglichst große Kompatibilität gewährleisten zu können [Fechner, 2007]. In Kapitel 3.1 wird ODRL noch einmal genauer analysiert, da diese Sprache in der angestrebten Lösung Verwendung findet. XrML hingegen basiert auf dem XML-Standard und wird von Microsoft zur Rechtedefinition der Windows Media DRM Dateien wmv und wma verwendet [Schmidt u. a., 2004]. In Bezug auf das DRM-System FairPlay von Apple ist nicht bekannt, ob überhaupt eine Rechtebeschreibungssprache eingesetzt wird.

Um virtuelle Waren zu schützen werden verschiedene Instrumente eingesetzt, welche nach [Grimm, 2006] in rechtliche, technische und organisatorische Maßnahmen unterteilt werden können.

Der **rechtliche Bereich**, welcher DRM im weiteren Sinne widerspiegelt, wird durch das im vorherigen Abschnitt 2.1.2 beschriebene Urheberrechtsgesetz, das deutsche Grundgesetz (GG; [GG, 2009]) und das Bundesdatenschutzgesetz (BDSG; [BDSG, 2009]) vertreten [Gossmann, 2007].

[2] musicload; URL: http://www.musicload.de/; Stand: 23.03.2010

Zu den **technischen Mitteln**, die als DRM im engeren Sinne gelten, zählen die in diesem Kapitel angesprochenen „Digital Rights Management"-Verfahren. Zu diesen Verfahren gehören Kopierschutzmechanismen wie XCP (Extended Copy Protection, von Sony eingesetzt) als auch digitale Wasserzeichen, die als Bitmuster verborgen in dem entsprechenden Medienmaterial einkodiert werden. Solche digitalen Wasserzeichen dienen der Erkennung des Urhebers und es kann nachgewiesen werden, dass die digitale Ware nicht gefälscht wurde [Arlt, 2006]. Des Weiteren gewähren beispielsweise die bereits im oberen Abschnitt beschriebenen Multimediashops Apple iTunes und musicload mit ihren unterschiedlichen Nutzungsregeln dem Benutzer z.B. das Recht der freien Nutzung. Diese „freie Nutzung" hat jedoch verschiedene Einschränkungen wie z.B. Begrenzung der Anzahl der Geräte und Angabe eines bestimmten Zeitraums, welche mit Hilfe der Rechtedefinitionssprachen ODRL und XrML umgesetzt werden [Grimm, 2006].

In den Bereich der **organisatorischen Maßnahmen**, die dem DRM im weiteren Sinne angehören, ordnen sich Geschäftsmodelle ein, die mit Hilfe von Client-Tools, Kommunikationsprotokollen (siehe Kapitel 2.3.2) und integrierten DRM-Mechanismen umgesetzt werden [Grimm, 2006]. Dabei soll der Nutzer durch besondere Anreize, wie es z.B. das PotatoSystem[3] umsetzt, zum urheberkonformen Handeln bewegt werden.

Durch die verschiedenen Interessen der Anbieter und Nutzer von digitalen Inhalten kommt es gerade in Bezug auf die Steuerungsmechanismen des DRMs zum Widerspruch [Gossmann, 2007]. Die Anbieter möchten die illegale Nutzung und Verbreitung virtueller Waren durch DRM-Maßnahmen wie Kopierschutz unterbinden, jedoch werden dadurch die Nutzer legal erstandener Medien in der Verwendung eingeschränkt. Um diesem marktbelastenden Konflikt zu entgehen werden in der Musikindustrie dringend Alternativen gesucht [Grimm u. Pähler, 2009].

[3] PotatoSystem; URL: http://www.potatosystem.com; Stand: 23.03.2010

Es gibt bereits alternative Vertriebssysteme wie etwa das PotatoSystem, das kein DRM im engeren Sinne verwendet, sondern durch ein Provisionsverfahren positive Anreize bei den Nutzern setzt [Nützel u. Grimm, 2005]. Anbieter von DRM-Inhalten, wie z.B. Apple oder Amazon[4][5] haben bereits auf den Unmut der Käufer reagiert und bieten mittlerweile nur noch Versionen ohne Kopierschutz an.

## 2.2 Netzwerktechnik

In dem Kapitel der Netzwerktechnik werden bestimmte Ausdrücke konkretisiert, die für diese Diplomarbeit und vor allem in Bezug auf die verwendete P2P-Technologie von grundlegender Bedeutung sind.

### 2.2.1 Netzwerkgrundlagen

Als Netzwerk wird ein System bezeichnet, das mehrere Rechner miteinander verbindet, so dass diese Rechner Informationen bzw. Daten untereinander austauschen können [Tanenbaum, 2003]. Dabei macht es keinen Unterschied, ob die Computer im gleichen Raum oder womöglich auf zwei verschiedenen Kontinenten stehen. Die Verbindung kann sowohl drahtgebunden z.B. über Kupferdraht oder durch Glasfaserkabel als auch drahtlos z.B. durch Funktechnik oder mit Kommunikationssatelliten realisiert werden [Tanenbaum, 2003]. Ein solches Computer-Netzwerk ist in folgender Abbildung 2.2 skizziert.
Die Datenübertragung erfolgt in der Regel über das Netzwerkprotokoll „IP“ (weitere Informationen hierzu sind ausführlich in [Peterson u. Davie, 2008] dargestellt) und kann durch verschiedene Netzwerkmodelle umgesetzt werden, welche im folgenden Kapitel beschrieben werden.

[4] Amazon; URL: http://www.amazon.de/; Stand: 23.03.2010
[5] Heise-News: Amazon startet Online-Musikshop mit DRM-freien MP3-Dateien; URL: http://www.heise.de/newsticker/meldung/96523; Stand: 23.03.2010

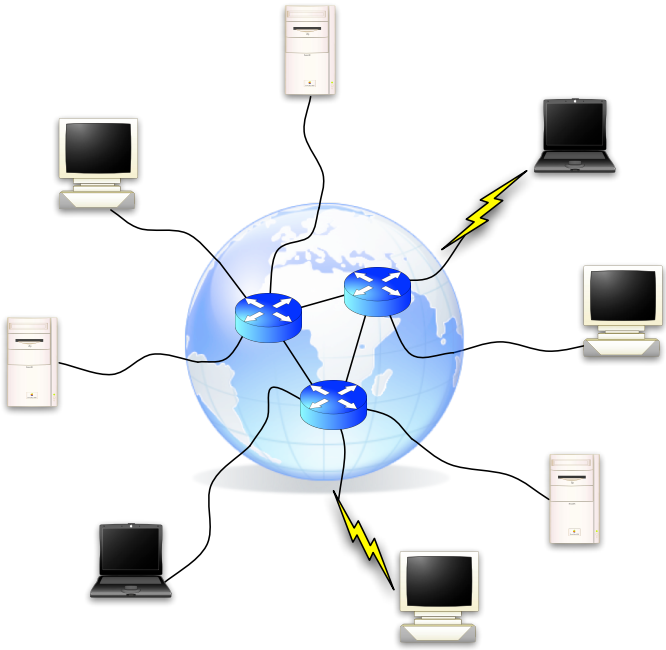

Abbildung 2.2: Darstellung eines Netzwerks

### 2.2.2 Netzwerkmodelle

Ein typisches Arrangement für den Austausch von Daten z.B. in einem Unternehmen ist nach [Tanenbaum, 2003] das **Client/Server-Modell** (siehe dazu folgende Abbildung 2.3).
Es werden leistungsstarke Computer, auch Server genannt, verwendet, um die Daten zu speichern. Diese Server stehen meist an einem zentralen Standort und müssen in der Regel von einem Systemadministrator verwaltet werden. Des Weiteren gibt es die weniger leistungsstarken Rechner, die Clients genannt werden und z.B. von den Mitarbeitern an ihren Arbeitsplätzen verwendet werden. Mit diesen Clients kann auf die entfernten Daten des Servers zugegriffen und entsprechend gearbeitet werden. Aus diesem Konzept lassen sich zwei Prozesse ableiten. Auf der einen Seite das Verfahren, das der Server anwendet, schließlich darauf zu hören, wenn eine Anfrage von einem Client ankommt und diese dann entsprechend auszuführen, in dem die angeforderten Information an den Client gesendet werden. Auf der anderen Seite findet im Client der Prozess statt,

dass beim Server nach vom Computernutzer geforderten Daten angefragt wird und der Client die Informationen empfängt. [Tanenbaum, 2003]

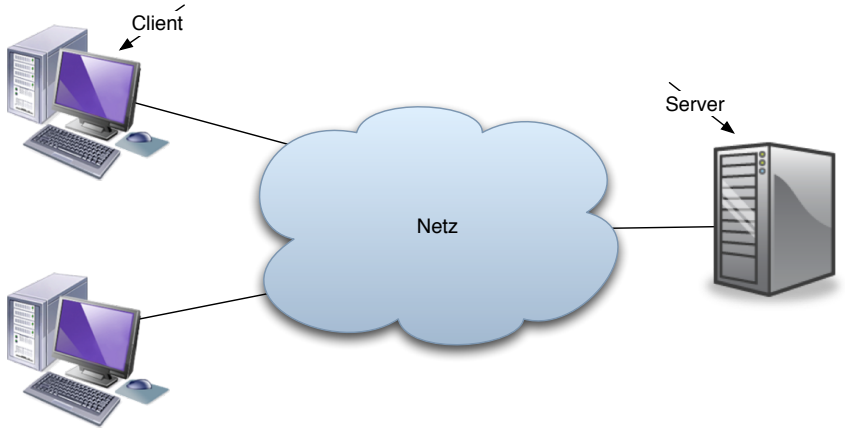


Abbildung 2.3: Ein Client/Server-Modell
in Anlehnung an [Tanenbaum, 2003]

Im Gegensatz zum Client/Server-Modell steht das **Peer-to-Peer-System**, siehe Abbildung 2.4. In diesem System gibt es keine festen Server und Clients [Tanenbaum, 2003], sondern jeder Rechner ist zueinander gleichgestellt. Das bedeutet, dass es keine feste Einordnung in Server und Clients gibt und jeder Computer mit einem oder mehreren Computern kommunizieren kann [Tanenbaum, 2003], indem jeder Rechner sowohl die Tätigkeiten eines Clients (z.B. Informationsanfrage und -speicherung) als auch die Aufgaben eines Servers (z.B. Anfragen beantworten und entsprechende Daten senden) übernehmen kann.
Meistens stehen in solchen P2P-Netzen nach [Tanenbaum, 2003] sehr viele Personen miteinander in Verbindung, die im Regelfall eine permanente Internetverbindung verwenden und Ressourcen gemeinsam nutzen. Die ersten Tauschbörsen in diesem Bereich waren Napster und KaZaA, welche jedoch ohne die Erlaubnis der Urheber deren Songs austauschten [Peterson u. Davie, 2008], so dass diese System geschlossen werden mussten [Tanenbaum, 2003]. Ein interessanter Aspekt an reinen Peer-to-Peer-Netzen ist, dass die teilnehmenden Rechner sich in keiner Hierarchie befinden und keine zentrale Steuerung wirkt [Tanenbaum, 2003].

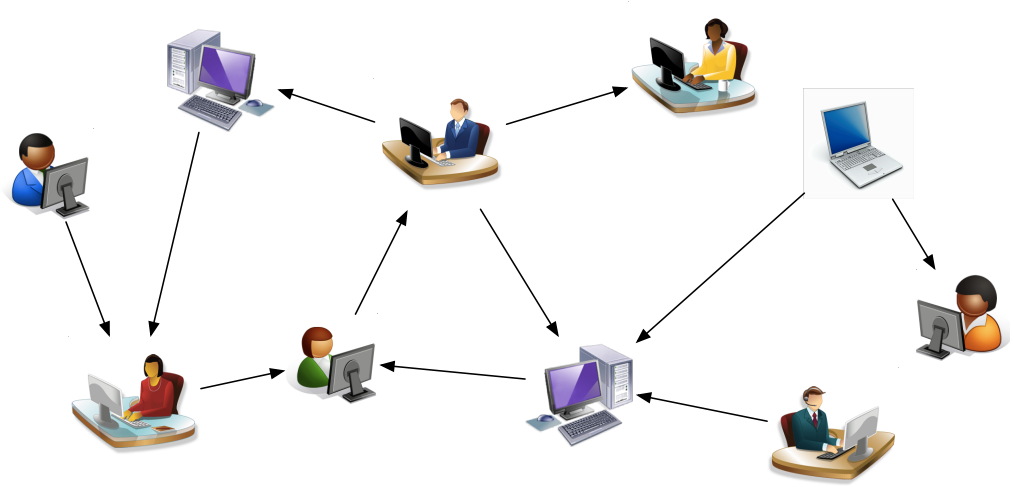

Abbildung 2.4: Ein Peer-to-Peer-System
in Anlehnung an [Tanenbaum, 2003]

Im Gegensatz zu [Tanenbaum, 2003] unterscheidet die Dissertation [Gehrke, 2004] in Übereinstimmung mit [Steinmetz u. Wehrle, 2004] und [Hong, 2001] drei verschiedene P2P Architekturen:

1. Reine P2P-Architekturen (Pure P2P)
2. P2P-Architekturen mit einer zentralen Einheit (Brokered P2P)
3. Hybride und hierarchische P2P-Architekturen

Charakteristisch für die Anwendung der P2P-Technologie ist das Unterstützen des Austauschs von digitalen Inhalten (siehe Abbildung 2.1), wie Musik oder Filme. Grundsätzlich ist es so, dass P2P-Netze sehr groß sind und sich die Benutzer untereinander nicht kennen, so dass es nicht ganz so einfach ist, ohne eine zentrale Stelle die gesuchten Daten zu finden [Tanenbaum, 2003]. Aus diesem Grund wird auch oft ein sogenanntes zentralisiertes P2P-Netzwerk genutzt, wie es z.B. auch Napster verwendet hat. Es besitzt ein zentrales Dateienregister, so dass jeder Benutzer zunächst darin sucht und bei erfolgreicher Suche an den einzelnen Rechner weitergeleitet wird, der über die gesuchte Datei verfügt und diese dann dort heruntergeladen werden kann [Peterson u. Davie, 2008].

Gehrke [Gehrke, 2004] weist auf die für reine P2P-Systeme typischen Probleme der verteilten Suche hin (z.B. exponentielle Skalierung bei Suchalgorithmen) und stellt die Gefahr von Einbußen der Performance bei Aktionen innerhalb des P2P-Systems heraus. Er bezeichnet den Datenfluss als ein *„hindurch [...] diffundieren“* [Gehrke, 2004, S. 18]. Den Nachteil bei Brokered P2P-Systemen (z.B. mit einem zentralen Dateienregister) stellt Gehrke als erhöhten Aufwand in der Verwaltung dieser Systeme heraus und verweist dabei auf *„fortgeschrittene Algorithmen“* [Gehrke, 2004, S. 20] und indirekt auf fehlende Skalierbarkeit bei offenen Systemen. Bei hybriden und hierarchischen Systemen übernehmen die im Netzwerk teilnehmenden Peers verschiedene Rollen, so dass ein Netzwerkteilnehmer entweder als klassischer Peer agiert oder die Rolle eines verwaltenden „Broker“-Knotens (in [Hong, 2001] als Supernode bezeichnet) einnimmt. Es ist auch vorgesehen, dass ein Peer beide Rollen gleichzeitig ausüben kann. Mit der Einführung von Hierarchien werden „Broker“-Knoten mit festgelegten Kommunikationsrichtlinien ausgestattet, die den Netzwerkverkehr (z.B. Suchanfragen) koordinieren. Laut [Gehrke, 2004] sollte die Hierarchie mehrfach redundant ausgelegt werden, damit der eventuelle Ausfall eines Supernodes kompensiert werden kann.

### 2.2.3 P2P-Netzwerkprotokoll JXTA

JXTA[6] ist ein kollaboratives Forschungsprojekt für P2P-Systeme. Das JXTA Projekt ist Open Source, wurde 2001 von Sun Microsystems[7] ins Leben gerufen und von Mitwirkenden aus der ganzen Welt weiterentwickelt. Der Name JXTA ist abgeleitet von dem Wort „juxtapose“, das nach [Wilson, 2002] bedeutet, dass zwei Peers nebeneinander oder in unmittelbarer Nähe existieren. Mit der Wahl dieser Bezeichnung erkannte das Entwicklungsteam von Sun an, dass P2P-Lösungen neben Client/Server- und webbasierten Lösungen eine Existenzberechtigung besitzen, ohne diese komplett zu ersetzen.

[6] JXTA Project; URL: https://jxta.dev.java.net/; Stand: 20.04.2010
[7] Sun Microsystems; URL: http://de.sun.com/; Stand: 21.04.2010

JXTA ist keine Programmierschnittstelle (API), Programmiersprache, Anwendung oder Code-Bibliothek, sondern nach [Microsystems, 2007] eine offene Netzwerkplattform für P2P-Anwendungen.

JXTA bietet ein Paket von offenen Protokollen, welche durch Open Source Referenzimplementierungen (z.B. für die Programmiersprache Java) für die Entwicklung von P2P-Anwendungen unterstützt werden. Die JXTA-Protokolle werden in einem der folgenden Abschnitte genauer beschrieben und standardisieren die Art und Weise, in der die Peers:

- sich gegenseitig entdecken
- sich in Gruppen selbst organisieren
- Netzwerkressourcen anbieten und entdecken
- miteinander kommunizieren
- sich gegenseitig überwachen

Die JXTA-Protokolle sind so konzipiert, dass sie unabhängig von der verwendeten Programmiersprache und dem verwendeten Transportprotokoll sind. Das bedeutet nach [Microsystems, 2007], dass die Protokolle mit der Programmiersprache Java, C/C++, .NET und zahlreichen anderen Sprachen implementiert werden können. Darüber hinaus können die Protokolle über TCP/IP, HTTP, Bluetooth oder andere Protokolle genutzt werden, während die allgemeine Kompatibilität die ganze Zeit aufrechterhalten bleibt.

Die von JXTA angebotenen Protokolle ermöglichen den Entwicklern die Erstellung und Bereitstellung von kompatiblen P2P-Anwendungen und -Services. Da die Protokolle nach [Oaks u. a., 2002] unabhängig von der Programmiersprache, der Plattform und dem Netzwerk sind, können heterogene Geräte mit komplett unterschiedlichen Softwarekomponenten miteinander interagieren. Mit JXTA-Technologie können laut [Microsystems, 2007] Entwickler vernetzte, kompatible Anwendungen schreiben, die folgende Aufgaben ausführen können:

- Finden von anderen Peers im Netzwerk durch dynamische Entdeckung über Firewalls und NATs.
- Einfaches Teilen von Ressourcen mit beliebigen Instanzen über das Netzwerk.
- Finden von verfügbaren Inhalten auf Internetseiten.
- Erstellen einer Gruppe von Peers, die einen Service anbieten.
- Überwachung von Peeraktivitäten aus der Ferne.
- Sichere Kommunikation mit anderen Peers über das Netzwerk.

Die folgenden Abschnitte bieten einen Einblick in die Terminologie, die Konzepte und die Architektur von JXTA und setzen diese Strukturen in den allgemeinen Rahmen von P2P-Netzwerken.

### JXTA-Konzepte

In diesem Abschnitt werden wichtige Begrifflichkeiten und Bestandteile von JXTA beschrieben.

*Peers*
Ein Peer ist nach [Microsystems, 2007] eine vernetzte Einheit, die ein oder mehrere JXTA-Protokolle implementiert. Peers können sich sowohl auf Computern als auch auf Sensoren, Handys und PDAs befinden. Jeder Peer agiert unabhängig und asynchron von allen anderen Peers und wird von JXTA eindeutig durch eine Peer-ID identifiziert. JXTA-Peers können in Anlehnung an [Microsystems, 2007] in folgende Kategorien unterteilt werden:

- *Edge-Peers*: Einfache Peers, die einem einzigen Endbenutzer dienen. Die minimalen Edge-Peers wie z.B. Sensorgeräte oder Geräte zur Haussteuerung implementieren nur die erforderlichen JXTA Kernservices und werden mit Hilfe von anderen Peers im JXTA-Netzwerk

voll integriert. Die vollständigen Edge-Peers implementieren alle Kern- und Standard-Services von JXTA und bilden laut [Microsystems, 2007] z.b. mit Computern oder Telefonen die Mehrheit der Peers in einem JXTA-Netzwerk.

- *Super-Peers*: Peers, die eine besondere Rolle beim Aufbau und der Bereitstellung des JXTA-Netzwerks einnehmen. Der Super-Peer kann von den folgenden Funktionen eine oder mehrere in kombinierter Form implementieren:
  - *Relay*: Ermöglicht die Speicherung und Weiterleitung von Nachrichten zwischen Peers, die wegen Firewalls oder NAT keine direkte Verbindung aufbauen können.
  - *Rendezvous*: Verwaltet Indexe globaler Advertisements, um die Effizienz der Advertisement-Suche zu erhöhen.
  - *Proxy*: Ist verantwortlich für die Unterstützung von minimalen Edge-Peers, die selbst nicht auf alle JXTA-Netzwerkfunktionen zugreifen können. Der Proxy-Peer übersetzt und fasst Zugriffe zusammen, beantwortet Anfragen und bietet Unterstützungsfunktionen für minimale Edge-Peers.

Die beschriebenen Kategorien umfassen die häufigsten Konfigurationen von Peers. Je nach Anwendung und Peer-Fähigkeiten kann es sinnvoll sein, die Peers mit einer Mischung aus den genannten Funktionen einzusetzen.

*Peer Groups*

Eine Peer Group ist eine Gruppe von Peers, die den gemeinsamen Interessen der gruppierten Peers dient. Beispielsweise bietet eine Peer Group spezifische Services, die nur für die Mitglieder dieser Gruppe bestimmt sind, an. Peers organisieren sich grundsätzlich selbst in Peer Groups und können darüber hinaus gleichzeitig mehreren Peer Groups angehören. Peer Groups können anhand einer PeerGroup-ID eindeutig zugeordnet werden und Richtlinien für die Mitgliedschaft definieren (z.B. offen oder

geschützt). Die sogenannte NetPeerGroup ist die Standard Peer Group eines JXTA-Netzwerks, welche standardmäßig als erste Gruppe instanziiert wird. Wie Peers eine Peer Group veröffentlichen, finden, beitreten und überwachen können wird in dem Abschnitt „JXTA-Protokolle“ beschrieben.

*Services*
In einem JXTA-Netzwerk kooperieren und kommunizieren Peers miteinander, um Netzwerk-Services zu veröffentlichen, zu entdecken und aufzurufen. Peers können mehrere Services veröffentlichen, die von anderen Peers entdeckt werden können. Die von JXTA angebotenen Protokolle identifizieren zwei Typen von Netzwerk-Services:

- *Peer Services*: Ein Peer Service ist nur auf dem Peer erreichbar, der den Service veröffentlicht hat. Wenn dieser Peer ausfällt, ist der Service nicht mehr erreichbar. Mehrere Service-Instanzen können auf verschiedenen Peers laufen, aber jede Instanz veröffentlicht sein eigenes Advertisement.

- *Peer Group Services*: Ein Peer Group Service ist eine Sammlung von Instanzen eines Services, die auf mehreren Peers der entsprechenden Peer Group laufen. Wenn ein Peer ausfällt, ist der kollektive Peer Group Service nicht betroffen, solange der Service noch von mindestens einem weiteren Peer-Mitglied angeboten wird. Peer Group Services werden als Teil eines Peer Group Advertisements veröffentlicht.

JXTA-Entwickler sind in der Lage, ihre eigenen angepassten Peer Services und Peer Group Services zu schreiben, während JXTA mehrere integrierte Peer Group Services zur Verfügung stellt. Diese beinhalten erforderliche Kernservices und optionale Services.
Die Kern Peer Group Services, die von jedem Peer implementiert werden müssen, der das JXTA-Netzwerk verwenden möchte, sind der „Endpoint Service“ und der „Resolver Service“. Der Endpoint Service wird verwendet, um Nachrichten zwischen Peers zu versenden und zu empfangen und

implementiert das Endpoint Routing Protocol. Der Resolver Service sendet generische Abfrageanforderungen an andere Peers. Peers können Anfragen definieren und austauschen um benötigte Informationen zu finden. Weitere optionale Peer Group Services sind der „Discovery“, „Membership“, „Access“, „Pipe“ und „Monitoring Service“.

*Advertisements*
Ein Advertisement in JXTA ist eine strukturierte Darstellung einer Instanz, eines Services oder einer Ressource, die von einem Peer oder einer Peer Group als Teil eines P2P-Netzwerks zur Verfügung gestellt wird. Alle in den vorherigen Absätzen beschriebenen Komponenten von JXTA, wie Peers, Peer Groups oder Services können in JXTA durch Advertisements beschrieben werden. Advertisements werden anhand eines XML-Dokuments in sprachenneutraler Struktur dargestellt. Die JXTA-Protokolle verwenden Advertisements, um die Existenz einer Peer-Ressource zu beschreiben und zu veröffentlichen. Jedes Advertisement wird mit einer Lebensdauer veröffentlicht, die die Verfügbarkeit der zugehörigen Ressource festlegt. Damit werden veraltete Ressourcen ohne jegliche zentrale Steuerung gelöscht. Um die Lebensdauer einer Ressource zu verlängern, kann ein Advertisement immer wieder erneut veröffentlicht werden. Die JXTA-Protokolle definieren verschiedene Advertisement-Typen, wie z.B. Peer, Peer Group oder Rendezvous Advertisements. Die vollständige Spezifikation der Advertisements von JXTA ist in der JXTA Protokoll Spezifikation[8] zu finden.

*JXTA-Shell*
JXTA bietet ein Kommandozeilen-Programm an, das dafür genutzt werden kann, Probleme zu beheben und JXTA Services zu testen. Befehle sind beispielsweise für das Entdecken von JXTA-Advertisements, das Erzeugen von JXTA-Ressourcen, das Beitreten und Verlassen einer Peer Group oder das Senden und Empfangen von Nachrichten vorhanden.

[8]JXTA Protokoll Spezifikation; URL: https://jxta-spec.dev.java.net/; Stand: 22.04.2010

### JXTA-Protokolle

JXTA definiert nach [Microsystems, 2007] eine Reihe von XML-Nachrichten bzw. Protokollen für die Kommunikation zwischen Peers. Diese Protokolle werden von den Peers verwendet, um einen anderen Peer zu entdecken, Netzwerk-Ressourcen zu bewerben und zu entdecken und die Kommunikation und Weiterleitung von Nachrichten zu realisieren.
Es gibt sechs Standard JXTA Protokolle, die asynchron sind und auf einem Anfrage/Antwort-Modell basieren:

- Peer Discovery Protocol (PDP)
- Peer Information Protocol (PIP)
- Peer Resolver Protocol (PRP)
- Pipe Binding Protocol (PBP)
- Endpoint Routing Protocol (ERP)
- Rendezvous Protocol (RVP)

Eine vollständige Beschreibung der JXTA-Protokolle ist in der JXTA Protokollspezifikation[9] zu finden.

### JXTA-Architektur

Die JXTA-Architektur kann in drei logische Schichten unterteilt werden, die in Abbildung 2.5 dargestellt sind. Die Kernschicht (*JXTA Core*), die Serviceschicht (*Services Layer*) und die Anwendungsschicht (*Applications Layer*). Jede Ebene baut nach [Wilson, 2002] auf den Fähigkeiten der darunter liegenden Schicht auf und fügt neue Funktionen und eine höhere Verhaltenskomplexität hinzu.

*JXTA Core*
Der Kern von JXTA enthält die Elemente, die für jede P2P-Lösung unabdingbar sind und bereits im vorherigen Abschnitt der JXTA-Konzepte

[9] JXTA Protokoll Spezifikation; URL: https://jxta-spec.dev.java.net/; Stand: 22.04.2010

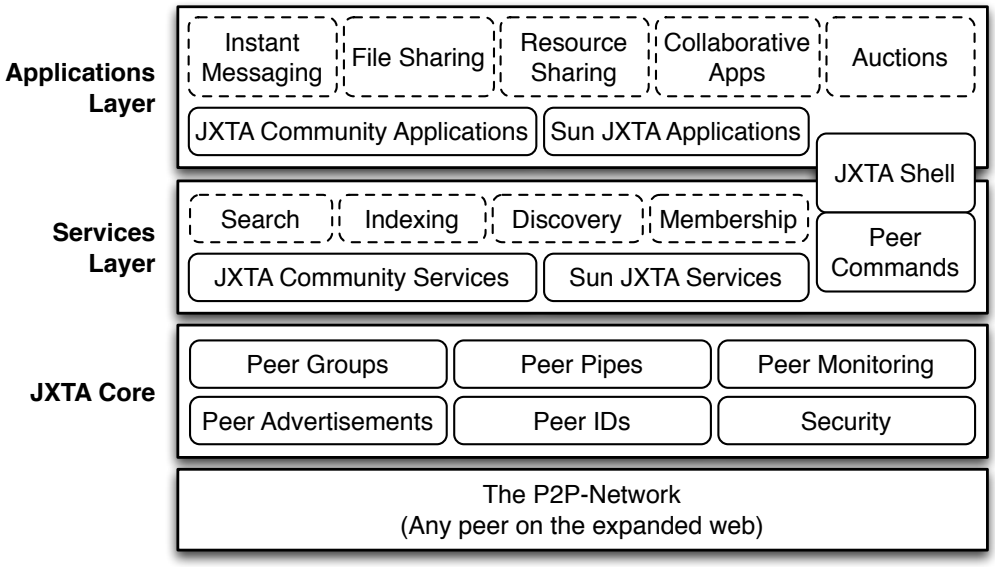


Abbildung 2.5: Die JXTA Architektur
in Anlehnung an [Templeton, 2002] und [Wilson, 2002]

erläutert wurden. Dazu gehören laut [Microsystems, 2007] die wichtigen Mechanismen der Entdeckung, der Kommunikationstransporte (einschließlich Firewall und NAT-Überquerung), der Erstellung von Peers und Peer Groups und die damit verbundenen Sicherheitsfunktionen.
Die Kernschicht integriert die sechs wichtigsten JXTA-Protokolle. Obwohl diese als Services implementiert sind, sind diese in der Kernschicht angesiedelt und werden nach [Wilson, 2002] als Kern-Services bezeichnet, um sie von den Service-Lösungen der Serviceschicht zu unterscheiden. Alle anderen Aspekte einer JXTA-P2P-Lösung in der Service- oder Anwendungsschicht bauen auf dieser Schicht auf und bieten weitere Funktionalitäten.

*Services Layer*
Die mittlere Schicht der JXTA-Architektur umfasst Netzwerkdienste, die laut [Microsystems, 2007] und [Wilson, 2002] für das Betreiben eines P2P-Netzwerks nicht unbedingt notwendig, aber in einer P2P-Umgebung gebräuchlich und wünschenswert sind. Diese Netzwerk-Services

implementieren Funktionen, welche in verschiedenen P2P-Anwendungen aufgenommen werden können, wie z.B. die Suche von Ressourcen auf einem Peer, das Teilen von Dokumenten von einem Peer oder das Durchführen einer Peer-Authentifizierung. Die Serviceschicht umfasst zusätzliche Funktionen, die durch die JXTA-Gemeinschaft (mit dem JXTA-Projekt arbeitende Open Source Entwickler) aufgebaut sind, in Ergänzung zu den Services des JXTA Projektteams. Auf der JXTA-Plattform eingesetzte Services bieten spezifische Fähigkeiten, die von einer Vielzahl von P2P-Anwendungen benötigt werden und kombiniert eine vollständige P2P-Lösung bilden.

*Applications Layer*
Die Anwendungsschicht baut auf den Fähigkeiten der Serviceschicht auf und umfasst die Implementierung von integrierten Anwendungen, wie P2P-Instant-Messaging, Dokumenten- und Ressourcenteilung, Inhaltsmanagement und -auslieferung, P2P E-Mail-Systeme, verteilte Auktionssysteme und viele andere. Die obere Schicht stellt P2P-Anwendungen der JXTA-Community, sowie Demonstrationsanwendungen, wie die durch das JXTA Projektteam aufgebaute JXTA-Shell, zur Verfügung.

Die Abgrenzung zwischen Service- und Anwendungsschicht ist nach [Microsystems, 2007] nicht als streng anzusehen. Laut [Wilson, 2002] ist es manchmal sogar schwierig zu bestimmen, was eine Anwendung und was einen Service darstellt. Eine Anwendung könnte beispielsweise nur einen einzigen Service oder insgesamt mehrere Services umfassen oder eine Kundenanwendung kann z.B. nach [Microsystems, 2007] als Service an einen anderen Kunden angesehen werden. Normalerweise zeigt das Vorhandensein irgendeiner Form von Benutzeroberfläche an, dass es sich eher um eine Anwendung als um einen Service handelt. Im Falle der im vorherigen Konzeptabschnitt beschriebenen JXTA-Shell sind die meisten Funktionen auf Peer-Befehlen aufgebaut, das bedeutet einfache Services die Kommandozeilen-Argumente von der JXTA-Shell akzeptieren. Die JXTA-Shell selbst ist ein Service, der nur eine minimale Benutzeroberfläche

bietet und sich somit auf der Grenze von Service- und Anwendungsschicht ansiedelt.
Das gesamte System ist modular aufgebaut, so dass Entwickler eine Sammlung von Services und Anwendungen wählen und zusammenstellen können, die ihren Anforderungen entsprechen.

## 2.3 Softwareentwicklung

Das Kapitel der Softwareentwicklung führt den Leser in die Grundzüge der Softwaretechnik ein und behandelt im Speziellen die im Folgenden verwendete Modellierungssprache UML (Unified Modeling Language) und die zur Realisierung des lizenzbewussten P2P-Clients eingesetzte Programmiersprache Java.

### 2.3.1 Softwarelebenszyklus

Bevor ein Softwareprojekt durchgeführt wird, ist es nach [Sommerville, 2007] angemessen, sich zunächst einen allgemeinen Softwarelebenszyklus zu betrachten. Deshalb wird nun an dieser Stelle und in Abbildung 2.6 in Anlehnung an [Sommerville, 2007] und [Heinrich, 2007] auf die verschiedenen Phasen, die bei der Softwareentwicklung durchlaufen werden können, eingegangen.

Zunächst gibt es ein Problem, welches durch ein Softwaresystem gelöst werden soll (**Problemstellung**). Im nächsten Schritt wird damit begonnen, die in der Problemstellung angesprochenen Sachverhalte zu erfassen und Entscheidungen über die Durchführbarkeit und die Erfolgsaussichten des künftigen Projekts zu treffen (*Analysieren*), woraus eine **Problemanalyse** erstellt wird. Anschließend werden Anforderungen, die zur Lösung der Problemstellung notwendig sind, definiert (*Definieren*) und in einem **Pflichtenheft** natürlichsprachlich und oft mit formalen Ergänzungen dokumentiert. Dieses Pflichtenheft stellt die Basis für die Softwareentwicklung dar und wird häufig als Vertragsgrundlage zwischen dem Auftraggeber und dem Softwarehersteller verwendet. Im Anschluss an die

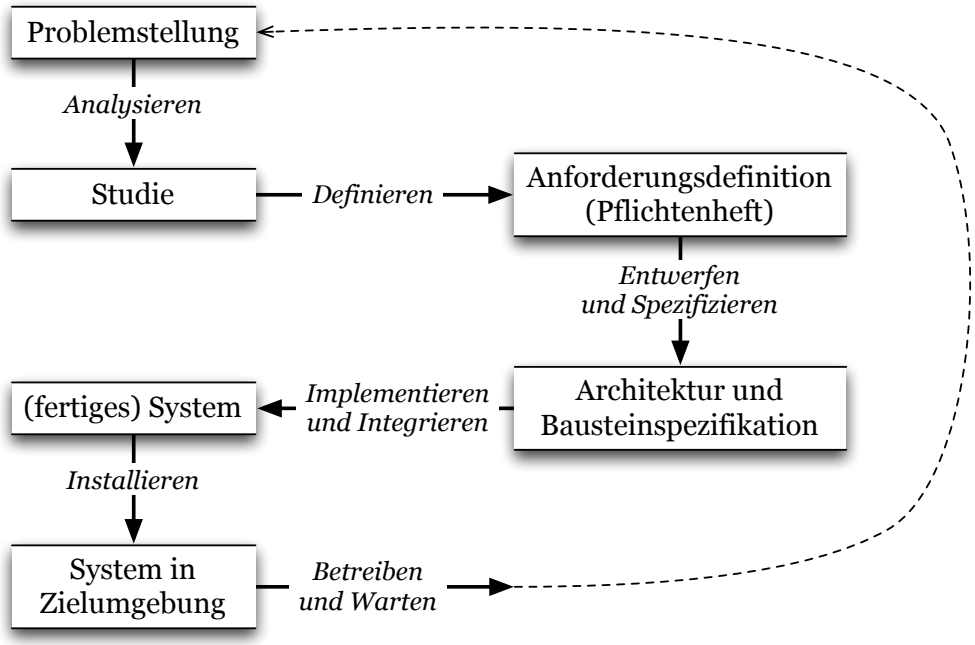


Abbildung 2.6: Der Softwarelebenszyklus
in Anlehnung an [Sommerville, 2007] und [Heinrich, 2007]

Definition der Anforderungen wird die **Architektur** durch das Konkretisieren der einzelnen Bausteine durch weitere Bausteine und deren Beziehungen (*Entwerfen*) und das Präzisieren von Anforderungen an die verschiedenen Bausteine für dessen Realisierung (*Spezifizieren*) festgelegt.
Nachdem die theoretische Grundlage gebildet ist, werden sowohl die Bausteine entworfen, programmiert und validiert (*Implementieren*) als auch anschließend zusammengefügt und gemeinsam validiert (*Integrieren*). Das vorliegende **(fertige) System**, wird in dessen Zielumgebung, beispielsweise auf den Servern des Auftraggebers, eingerichtet (*Installieren*), um es in Betrieb nehmen zu können.
Das **System in Zielumgebung** kann im Produktiveinsatz verwendet werden (*Betreiben*) und wird in den meisten Fällen währenddessen angepasst, weiterentwickelt, verbessert und auftretende Fehler werden entsprechend behoben (*Warten*).

Die beschriebenen Phasen eines Softwarelebenszyklus werden von weiteren Aktivitäten unterstützt, gelenkt und begleitet. Zu diesen phasenübergreifenden Prozessen zählen das **Projektmanagement** (Planen, Überwachen und Steuern), die **Qualitätssicherung** (z.B. Tests durchführen) und das **Dokumentieren** (z.B. Protokolle führen) [Heinrich, 2007].

### 2.3.2 Begriffsklärung im Softwarebereich

Nach dem deutlich geworden ist, welche Phasen während der Realisierung eines Softwareprodukts durchlaufen werden können, wird im folgenden Abschnitt auf grundlegende Aspekte der Softwareentwicklung eingegangen.

**Programmlogik / Benutzerschnittstelle**
Eine Software besteht zumeist aus einem Frontend und einem Backend. Das Backend ist die sogenannte Programmlogik und liegt, wie der Name vermuten lässt, versteckt im Hintergrund. Das bedeutet, dass beispielsweise der Benutzer diese nicht sehen oder direkt ansprechen kann. Damit der Benutzer die Möglichkeit hat, mit der Anwendung interagieren zu können (z.B. Eingaben tätigen, Funktionen aufrufen), gibt es das Frontend, welches auch Benutzerschnittstelle (User Interface) genannt wird und entsprechend die Verbindung zwischen der Programmlogik und dem Benutzer der Software herstellt. Eine solche Mensch-Maschine-Schnittstelle kann beispielsweise durch eine grafische Benutzeroberfläche (GUI Graphical User Interface) oder durch ein Kommandozeilen-Interface (CLI Command Line Interface) umgesetzt werden. Dabei wird die Kommandozeile oft auch gleichbedeutend mit dem generalisierenden Begriff „Shell“ verwendet, da das CLI zu Beginn der Computergeschichte die einzige Benutzerschnittstelle war [Wolf, 2007]. Bei dem CLI werden die entsprechenden Anweisungen anhand der Tastatur eingegeben, während mit der GUI umfassende Oberflächen gestaltet werden können, die gewöhnlich mit der Maus, wahlweise aber auch über andere Eingabegeräte angesteuert werden können [Chlebek, 2006].

**Open Source**

Als Open Source wird nach der Open Source Initiative[10] (OSI) eine Software bezeichnet, deren zugrunde liegender Programmcode für alle Interessenten unentgeltlich zur Verfügung steht, während die Software an sich beliebig kopiert, modifiziert und verbreitet werden darf. Dabei ist ebenfalls erlaubt, die Software in veränderter Form weiterzugeben [Asche u. a., 2008]. Weitere Informationen über dieses Themengebiet können in der Open Source Definition[11] (OSD) der OSI nachgelesen werden.

**GNU General Public License**

Bei der GNU General Public License (GPL) handelt es sich laut [Asche u. a., 2008] um eine der am meisten verbreiteten Standardlizenzen von Open-Source-Softwareprojekten, welche durch den ideologischen Hintergrund entstand, dass anhand von Software öffentliches Wissen repräsentiert wird und dessen uneingeschränkte Verbreitung gewährleistet werden sollte, in dem Software in einem quelloffenen Format vertrieben wird. Um zu verhindern, dass Software verkommerzialisiert wird, wird dies im rechtlichen Sinne durch die Einräumung von Nutzungsrechten sichergestellt, deren Anforderung es ist, jegliche Änderungen des Programmcodes auch wieder frei zur Verfügung zu stellen, so dass eine Geheimhaltung des Programmcodes auch bei zukünftigen Softwareversionen unterbleibt. [Asche u. a., 2008]

**Netzwerkprotokoll**

Ein Netzwerkprotokoll im Sinne der Informatik, in dieser Arbeit auch als Protokoll bezeichnet, regelt die Kommunikation zwischen zwei Instanzen und basiert in seiner minimalen Form auf einer Syntax (ein definiertes Format, z.B. die Wort-Reihenfolge), einer Semantik (die Bedeutung der Syntax) und Zeitbedingungen (z.B. Zeitbeschränkungen) [Peterson u. Davie, 2008].

[10] Open Source Initiative (OSI); URL: http://www.opensource.org/; Stand: 23.03.2010

[11] Open Source Definition (OSD); URL: http://www.opensource.org/docs/osd; Stand: 23.03.2010

**Socket**
Die Bezeichnung Socket ist abgeleitet von dem englischen Wort Steckdose und bezeichnet im Computerbereich ein Softwaremodul, welches sich anhand einer Computeranwendung mit einem Netzwerk verbinden und gegenwärtig Informationen mit anderen Rechnern austauschen kann. Der Socket wird dementsprechend als Endpunkt einer Netzwerkkommunikation deklariert. [Peterson u. Davie, 2008]

**Framework**
Der Begriff Framework wird häufig im Bereich der Softwareentwicklung verwendet und stellt eine Rahmenstruktur dar, die vor allem in objektorientierten Programmiersprachen verwendet werden. Das bedeutet, dass das Framework den Rahmen bereitstellt, innerhalb dessen der Softwareentwickler ein Programm entwickelt. Das Framework an sich ist also eine halbfertige, wiederverwendbare Anwendung, welche verfeinert werden kann um individuelle Anwendungen zu erstellen. Daraus ergibt sich der Vorteil, dass bei Verwendung von Frameworks sowohl die Entwicklungsdauer verkürzt als auch die Qualität erhöht werden kann, da die Kompetenz von vielen anderen Benutzern, die bereits daran gewirkt haben, mit einfließt. [Schildt, 2002]

**Entwurfsmuster**
Entwurfsmuster, im Englischen als Design Patterns bezeichnet, sind generische Musterlösungen für häufig in der Softwareentwicklung auftretende Problemstellungen. Sie sind unabhängig von der Programmiersprache und können mit Hilfe von objektorientierten Programmiersprachen umgesetzt werden. Beispiele für Design Patterns sind Singleton oder Listener. [Gamma u. a., 2009]

**Objektorientierte Programmierung**
Die objektorientierte Programmierung (OOP) basiert, wie der Name schon sagt, auf dem Konzept der Objektorientierung, die nach [Lahres u. Rayman, 2009] heutzutage ein etabliertes Verfahren ist, um die Komplexität von Softwaresystemen zu bewältigen. Dabei werden Daten und

Informationen in verschiedenen Objekten gesammelt und bearbeitet, welche beispielsweise nach außen durch entsprechende Attribute unsichtbar (private) oder sichtbar (public) gemacht werden können. Weitere Paradigmen zur objektorientierten Programmierung können in [Lahres u. Rayman, 2009] nachgelesen werden. Bekannte objektorientierte Programmiersprachen sind beispielsweise Java oder C#.

**XML**
XML ist eine erweiterbare Auszeichnungssprache, die vom World Wide Web Consortium (W3C)[12] mit Unterstützung von verschiedenen Unternehmen und Experten entwickelt und technologisch betreut wird. Anhand dieser Auszeichnungssprache können Informationen hierarchisch in einer Baumtopologie strukturiert werden. Es handelt sich um ein Textformat, dass auf Strukturregeln aufbaut. XML wird vor allem für den Datenaustausch zwischen verschiedenen Systemen eingesetzt, die Plattform- und Programmiersprachenunabhängig sind. [Vonhoegen, 2009]

**Web Services**
Ein Web Service ist ein sich selbst beschreibendes, in sich geschlossenes Softwaremodul, welches über ein Netzwerk erreichbar ist. Funktionen, wie z.B. spezifische Berechnungen durchführen oder Informationen im Internet suchen, können durchgeführt werden. Da Web Services auf gebräuchlichen Industriestandards und existierenden Technologien, wie XML und HTTP basieren, sind diese einfach einzusetzen. [Papazoglou, 2007]

### 2.3.3 Unified Modeling Language - UML

Die Unified Modeling Language (UML) ist eine komplexe Modellierungssprache, mit der Softwaresysteme analysiert und entworfen werden können. Sie ist laut [Rupp u. a., 2007] derzeit die gängigste unter den

[12]World Wide Web Consortium (W3C); URL: http://www.w3.org/; Stand: 05.05.2010

Notationen und wird entwickelt von der Object Management Group[13] (OMG). Seit dem Jahr 2005 liegt UML in Version 2 vor, welche in dieser Arbeit immer mit UML impliziert ist (UML = UML 2). Mit Hilfe der UML können umfassende Softwaresysteme modelliert, dokumentiert, spezifiziert und visualisiert werden. Die verschiedenen zur Verfügung stehenden Notationselemente ermöglichen es, die unterschiedlichsten Bereiche abzudecken, wie z.B. statische oder dynamische Modelle von Analyse, Design und Architektur und begünstigen im Besonderen die objektorientierten Methoden. Dabei gibt [Rupp u. a., 2007] jedoch zu beachten, das die UML nie allumfassend und widerspruchsfrei sein wird,

> *„[...] nicht perfekt, nicht vollständig, keine Programmiersprache, keine rein formale Sprache, nicht spezialisiert auf ein Anwendungsgebiet, kein vollständiger Ersatz für Textbeschreibung und vor allem keine Methode oder kein Vorgehensmodell [...]“ [Rupp u. a., 2007, S. 12]*

welches in einer hohen Vielseitigkeit und schnellen Fortschritten der gegenwärtigen Softwareentwicklung begründet liegt und das auch so gewollt ist. [Rupp u. a., 2007]

UML wird anhand von Diagrammsprachen eingesetzt, die sich in 13 verschiedene Diagrammtypen unterteilen und in Abbildung 2.7 illustriert sind. Es existieren sechs verschiedene Strukturdiagramme (z.B. ein Klassendiagramm), die für die Systemstatik verwendet werden können und sieben Verhaltensdiagramme (z.B. ein Aktivitäts- oder Use-Case-Diagramm), wovon wiederum vier Diagramme zur Darstellung von Interaktion bestimmt sind und ebenso grafisch wie tabellarisch notiert werden können (z.B. ein Sequenzdiagramm) [Rupp u. a., 2007].

Im Folgenden wird anhand von [Rupp u. a., 2007] entsprechend auf die für diese Ausarbeitung wichtigen und in Abbildung 2.8 skizzierten Diagramme eingegangen:

[13] Object Management Group (OMG); URL: http://www.omg.org/; Stand: 23.03.2010

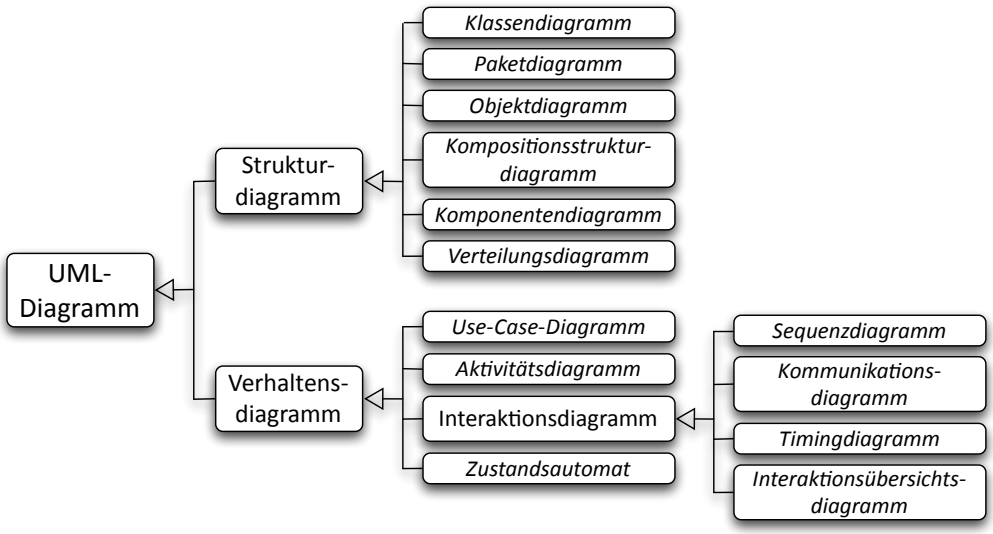


Abbildung 2.7: Diagrammtypen der UML
in Anlehnung an [Rupp u. a., 2007]

*Klassendiagramm*
Ein Klassendiagramm stellt die statische Form eines Systems dar, welches entworfen und abgebildet werden soll. Es bezieht alle wesentlichen Strukturrelationen und Datentypen mit ein und ist das Verbindungsglied zu den dynamischen Diagrammen. Das bedeutet für den Softwareentwickler, dass in diesem Diagrammtyp visualisiert wird, aus welchen Klassen das System besteht und in welcher Beziehung diese Klassen zueinander stehen.

*Use-Case-Diagramm*
Mit einem Use-Case-Diagramm, auch Anwendungsfalldiagramm genannt, wird in dynamischer Form die Sicht von außen auf eine Anwendung abgebildet, unter Verwendung von einfachen Notationsinstrumenten. Dies ist sehr hilfreich, um das Umfeld, wie z.B. Nachbarsysteme oder Stakeholder auf einem ausgeprägten Abstraktionslevel abgrenzen zu können.

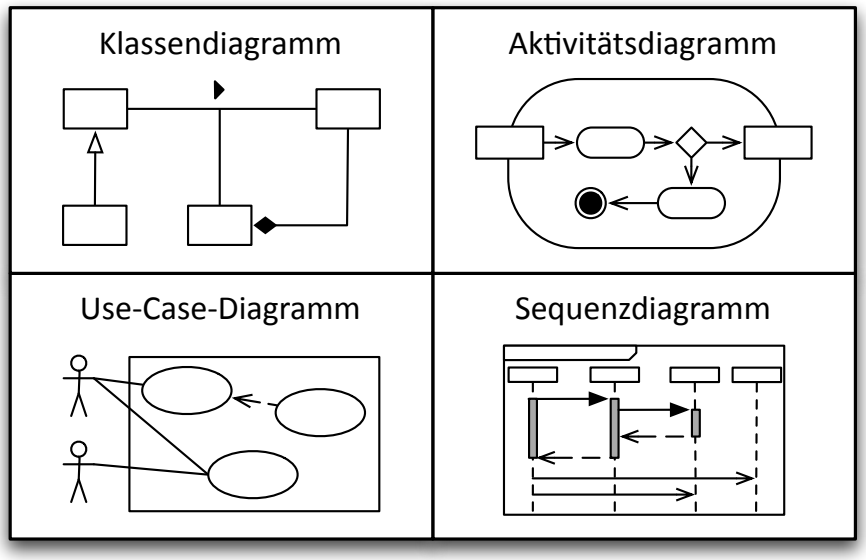


Abbildung 2.8: Beispiele für UML-Diagramme
in Anlehnung an [Rupp u. a., 2007]

*Aktivitätsdiagramm*
Ein Aktivitätsdiagramm wird verwendet, um Abläufe mit deren Bedingungen, Schleifen und Verzweigungen im Detail zu visualisieren, wobei selbst parallele und synchronisierte Abläufe dargestellt werden können. Für einen Programmierer heißt das, dass z.B. der Ablauf eines Algorithmus oder ein flussorientierter Prozess anhand dieses dynamischen Diagrammtyps skizziert werden kann.

*Sequenzdiagramm*
Welche Informationen mit welchen Akteuren und in welcher Abfolge ausgetauscht werden kann mit einem Sequenzdiagramm präsentiert werden. Das bedeutet, der Informationsaustausch zwischen zwei Kommunikationspartnern wird mit Hilfe von Schachtelungen und einer Flusssteuerung, wozu Bedingungen, Schleifen und Verzweigungen gehören, visualisiert.

### 2.3.4 Programmiersprache Java

Java gehört zu den objektorientierten Programmiersprachen (siehe auch Kapitel 2.3.2) und besitzt einige bedeutende Merkmalen (z.B. Sicherheit und Portabilität), wodurch die Sprache vielseitig einsetzbar ist und mit Hilfe derer robuste Software entwickelt werden kann [Ullenboom, 2009]. Ein hohes Maß an Sicherheit und die von Entwicklern begrüßte Portabilität tragen dazu bei. Die Sicherheit wird durch die Beschränkung einer Java-Anwendung auf die Java Virtual Machine (JVM) gewährleistet, welche den Java-Bytecode in eine Sprache übersetzt, die die zugrunde liegende Plattform versteht. Die Portabilität bezieht sich darauf, dass der ausführbare Java-Code auf alle Plattformen, die die JVM verwenden, portiert und ausgeführt werden kann.

Um zu verdeutlichen, was hinter dem Prinzip von Java steckt, ist in Abbildung 2.9 der Funktionsablauf von Java illustriert, welcher anhand von [Sierra u. Bates, 2005] im nachfolgenden Textabschnitt erläutert wird.

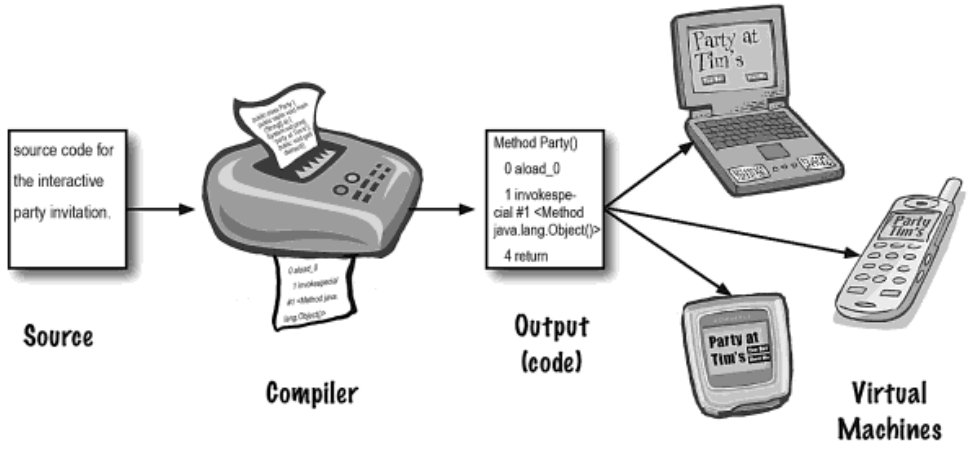


Abbildung 2.9: Der Funktionsablauf von Java
[Sierra u. Bates, 2005, S. 2]

Wie bereits angesprochen ist unter anderem ein Ziel von Java, ein Computerprogramm zu schreiben, welches auf einem breiten Spektrum von Endgeräten läuft. Dafür wird zunächst unter Verwendung der Java-Programmiersprache ein Dokument mit Quelltext erzeugt (*Source*). Über dieses

Dokument wird ein Quellcode-Compiler laufen gelassen, der den Quelltext nach Fehlern absucht und das Dokument erst kompiliert, wenn er sichergestellt hat, dass alles korrekt ausgeführt werden kann (*Compiler*). Der Compiler erzeugt nun eine neues Dokument und kodiert dieses in Java-Bytecode (*Output (code)*). Dieser Bytecode ist plattformunabhängig und kann somit nun von jedem Gerät, das fähig ist Java laufen zu lassen, in eine Datei interpretiert bzw. übersetzt werden, die von diesem Gerät ausgeführt werden kann. Zum Schluss wird also auf den Endgeräten in der sogenannten Java-Laufzeitumgebung, unter Einsatz der virtuellen Javamaschine (JVM), der Bytecode gelesen, in den entsprechenden Geräte-Code übersetzt und ausgeführt (*Virtual Machines*). [Sierra u. Bates, 2005]

Obwohl die beiden Aspekte Sicherheit und Portierbarkeit die herausragendsten Eigenschaften der Sprache Java darstellen, gibt es nach [Schildt, 2002] noch weitere wesentliche Gesichtspunkte, die im Folgenden aufgelistet sind.

- *Einfach*
  Durch die gut strukturierten Eigenschaften der Programmiersprache Java ist diese leicht zu erlernen und einsetzbar zu machen.

- *Sicher*
  Das Verfahren von Java zur Einrichtung von Applets (Anwendungen für das Internet) ist sehr sicher.

- *Portabel*
  Wie bereits thematisiert, können Java-Anwendungen überall dort ausgeführt werden, wo eine Java-Laufzeitumgebung zur Verfügung steht.

- *Objektorientiert*
  Die moderne Philosophie der Objektorientierung (siehe Kapitel 2.3.2) wird mit den typischen Aspekten, wie etwa Kapselung (Code wird mit den zu behandelnden Daten verbunden und vor Angriffen von außen geschützt), Polymorphie (eine Schnittstelle kann auf eine globale Klasse von Aktionen zugreifen) und Vererbung (ein Objekt kann

die Fähigkeiten eines anderen Objekts übernehmen) in Java umgesetzt.

- *Robust*
  Bereits während der Compilezeit überprüft Java den Quellcode auf Fehler und hält an einer strengen Typisierung fest, so dass der Entwickler von einer korrekten Programmierung ausgehen kann.

- *Multithreading*
  Unter Multithreading im Bereich der Programmierung wird die gleichzeitige Ausführbarkeit mehrerer Programmteile verstanden.

- *Architekturneutral*
  Mit Java muss der Entwickler sich nicht festlegen, auf welchem Betriebssystem oder Rechnertyp die programmierte Software später eingesetzt werden soll.

- *Interpretiert*
  Kompilierter Java-Code ist nicht direkt und unmittelbar auf der Hardware ausführbar. Es wird ein Zwischencode erzeugt, der nur von einer Zwischenschicht, der JVM, ausgeführt werden kann. Das bedeutet, dass Java-Programme von der JVM interpretiert werden können und Hardwareabhängiger Code generiert wird.

- *Leistungsstark*
  Die JVM ist für einzelne Rechnerarchitekturen optimiert, so dass der Java-Bytecode schnell ausgeführt werden kann.

- *Dynamisch*
  Java lädt nicht zwingend alle Klassen initial beim Start des Programms, sondern ist in der Lage dynamisch Klassen nachzuladen.

Im Vergleich zu anderen Programmiersprachen sticht die Eigenschaft von Java plattformunabhängig zu sein hervor. Beispielsweise ist diese Offenheit von Java in Bezug auf Plattformen nach [Ullenboom, 2009] der Hauptgrund, dass Java mit der objektorientierten Programmiersprache C#, die

von Microsoft für die .NET-Umgebung entwickelt wurde, nicht gleichzusetzen ist. Java und C# sind sich außer dieser Unterscheidung sehr ähnlich, z.B. ist das .NET-Framework vergleichbar mit den Java-Bibliotheken. In Sachen Innovativität hat sich laut [Ullenboom, 2009] die Programmiersprache von Microsoft durch eine starke Eigendynamik einen Vorsprung ausgebaut.

# 3 U*sage* R*ights* M*anagement*

In diesem Kapitel wird das Forschungsgebiet Usage Rights Management[1] (URM) der Professur Grimm an der Universität Koblenz-Landau beschrieben. Dieses kann in zwei Bereiche unterteilt werden, die theoretische Forschung und die praktische Umsetzung, welche durch das TURM-System repräsentiert wird und die theoretischen Modelle in die Praxis umsetzt. Beide Teilbereiche sind voneinander abhängig und ergänzen sich gegenseitig. Aus diesem Grund wird zum Schluss dieses Kapitels noch auf den Anwendungsbereich von URM eingegangen und ein Ausblick auf die weiteren Schritte der URM-Forschung gegeben. Das wissenschaftliche Themengebiet URM ist das Grundlagenthema dieser Arbeit und somit bedeutend für die Betrachtung der vorliegenden Arbeit.

## 3.1 Theoretische Aspekte

Das URM stellt eine abgewandelte Form des in Kapitel 2.1.3 beschriebenen Digital Rights Managements dar, da die Verwaltung der Nutzungsrechte im Fokus steht (siehe hierzu Abbildung 3.1).
URM zwingt den Nutzer nicht zur Einhaltung der Urheberrechte durch die Verschlüsselung von Mediendateien, sondern klärt den Benutzer über die entsprechenden Rechte (abspielen, weitergeben) an seinen digitalen Inhalten auf (*offenes System*). Diesem Vorgehen liegt nach [Hundacker u. a., 2009] der Gedanke zugrunde, dass es Nutzer gibt, die

[1] Usage Rights Management (URM); URL: http://urm.iwvi.uni-koblenz.de/; Stand: 23.03.2010

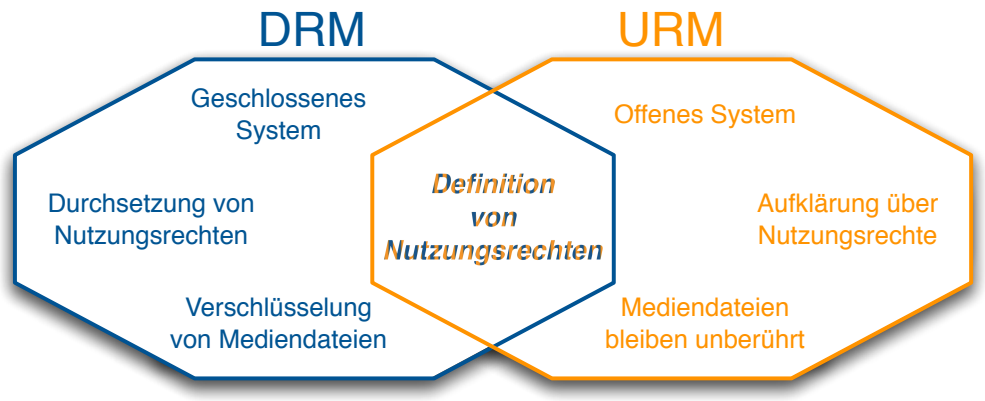


Abbildung 3.1: DRM (Kopierschutz) und URM im Vergleich
nach [Jahn, 2010]

sich legal verhalten wollen. Die Kopierschutzfunktion des reinen DRMs macht es dem Nutzer nicht offensichtlich, welche Nutzungsrechte ihm an seinen digitalen Medien zustehen (*geschlossenes System*). Weiterhin sind die virtuellen Waren durch die DRM-Kopierschutz technisch eingeschränkt, so dass z.B. das Recht einer Privatkopie nur noch sehr schwer in Anspruch genommen werden kann. [Jahn, 2010]

Die Notwendigkeit eines Aufklärungsinstruments über Nutzungsrechte wird in [Hundacker u. a., 2009] herausgestellt. Mittlerweile gibt es viele Faktoren, die sich unterschiedlich auf die Nutzungsrechte von digitalen Inhalten auswirken und damit dem unbedarften Nutzer die Transparenz nehmen. Einige Nutzungsrechte sind implizit durch das Gesetz erteilt, wie z.B. das Anfertigen einer Privatkopie, andere sind ausdrücklich durch den Urheber oder Anbieter definiert [Hundacker u. a., 2009]. Aspekte die sich auf die Nutzungsrechte von im Musikbereich vorhandenen MP3-Dateien auswirken sind nationale Unterschiede, da beispielsweise in Deutschland andere Gesetze gelten als im amerikanischen Rechtsraum [Hundacker u. a., 2009]. Dabei beschränkt sich die vorliegende Diplomarbeit auf den deutschlandweiten Raum, um die Komplexität nicht unnötig zu erhöhen.

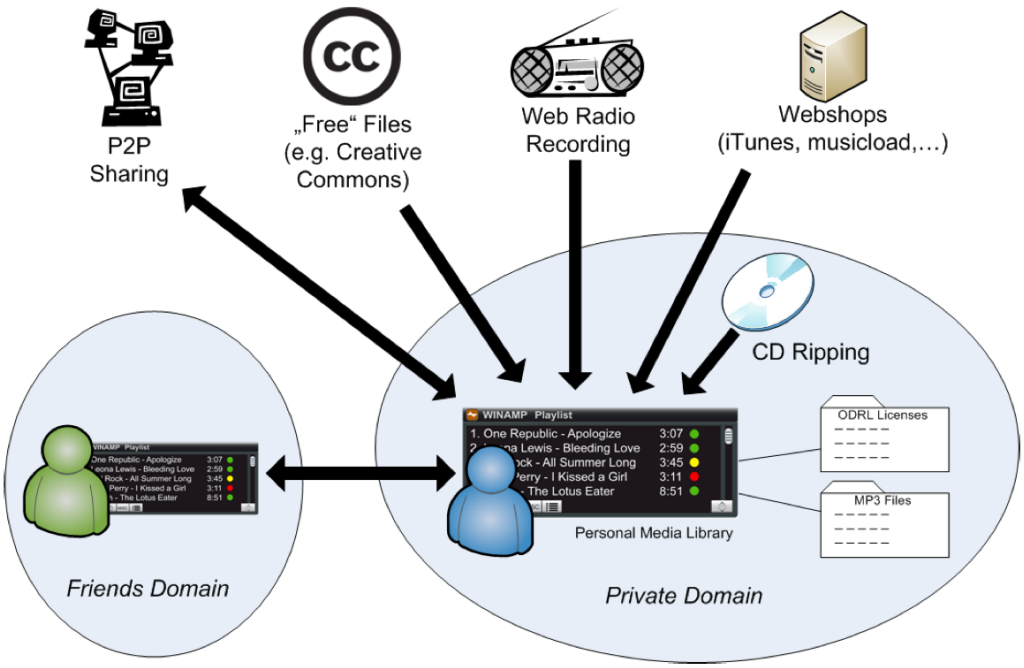


Abbildung 3.2: MP3-Datenquellen
[Hundacker u. a., 2009, S. 130]

Weiterhin wird anhand der Abbildung 3.2 der Publikation [Hundacker u. a., 2009], in der unterschiedliche Datenquellen aufgezeigt sind, deutlich, dass die Rechte durch ihre Abstammung variieren können. An einer Musikdatei, die aus einer käuflich erworbenen CD extrahiert wurde, hat der Nutzer z.B. andere Rechte, als an einem Song, der während einer Radiosendung mitgeschnitten wurde. Eine unter Creative Commons[2] (CC) stehende MP3-Datei hat andere Nutzungsrechte als die Dateien, die beispielsweise bei Webshops erworben wurden. Weiterhin setzen die verschiedenen Musikonlineshops unterschiedliche, verkäuferspezifische Konditionen durch [Hundacker u. a., 2009].

Die Grundidee von URM ist nach [Hundacker u. a., 2009], die Nutzungsrechte virtueller Waren anhand eines nutzerfreundlichen Lizenzsystems zu verwalten. Das bedeutet, dass für jede Mediendatei, wie z.B. eine

[2]Creative Commons (CC): Vergabe von vordefinierten Lizenzmodellen; URL: http://de.creativecommons.org/; Stand: 23.03.2010

MP3-Datei, eine Lizenz existiert, in der die genauen Rechte des Nutzers festgeschrieben sind, damit der Nutzer sich daran halten kann. Die lizenzierten Nutzungsmöglichkeiten einer Datei werden im Fall der URM anhand der Rechtedefinitionssprache ODRL (siehe dazu auch Kapitel 2.1.3) beschrieben und bilden die Lizenz. Ein Beispiel für eine mit ODRL beschriebene URM-Lizenz wird in Abbildung 3.3 vorgestellt.

```
<?xml version="1.0" encoding="utf-8"?>
<rights>
    <agreement>
        <context>
            <date><fixed>2009-04-20T22:33:51</fixed></date>
        </context>
        <asset>
            <context>
                <uid>c32969acdd6ab7415bf7a9e564afa4f37c974bb</uid>
                <name>
                    Queen - 1994 - Greatest Hits I - Bohemian Rhapsody
                </name>
                <dlocation>
                    file://C:\local_music_directory\01_Bohemian_Rhapsody.mp3
                </dlocation>
                <plocation>Helge's storage rack</plocation>
            </context>
        </asset>
        <permission>
            <play />
            <duplicate>
                <constraint>
                    <count>3</count>
                    <spatial><context>
                        <uid>iso3166:DE</uid>
                    </spatial></context>
                </constraint>
            </duplicate>
        </permission>
        <party>
            <context>
                <uid>hundacker@uni-koblenz.de</uid>
            </context>
        </party>
    </agreement>
</rights>
```

Abbildung 3.3: Beispiel einer URM-Lizenz
in Anlehnung an [Hundacker u. a., 2009]

Eine URM-Lizenz wird von den Attributen *rights* und *agreement* eingebunden und enthält darin die Attribute *context*, *asset*, *permission* und *party*. Folgende Informationen sind in einer Lizenz gespeichert: das Ausstellungsdatum der Lizenz (*date*), der gehashte Wert der Datei-Payload (genaueres dazu im folgenden Abschnitt) mit der entsprechenden Hashfunktion (hier SHA-1) als Präfix (*uid*), der Titelname der Datei (*name*), das Verzeichnis, in dem die Datei gespeichert ist (*dlocation*) und der Ort an dem das Medium abgelegt ist (*plocation*). Weiterhin sind im *permission*-Attribut die Rechte an der Datei, wie abspielen und weitergeben (*permission*) zusammen mit der Anzahl der zulässigen Kopien der Mediendatei (*count*) und dem Ländercode (*spatial/context -> uid*) festgelegt und im *party*-Attribut die E-Mail-Adresse des Eigentümers (*uid*) eingetragen. Dementsprechend ist die Lizenz des URM-Konzepts eine Erlaubnis (*permission*), für eine Person (*party*) in Bezug auf ein Objekt (*asset*) [Hundacker u. a., 2009]. Die Aufgabe, die in der Lizenz eingetragenen Werte aktuell und synchron zu halten (z.B. wenn eine Musikdatei in einen anderen Ordner verschoben wird, den Eintrag der *dlocation* entsprechend anzupassen) liegt in der Verantwortung des Benutzers bzw. kann vom TURM-System automatisch aktualisiert werden.

Die gerade beschriebene URM-Lizenz stellt eine sogenannte Erstlizenz dar, den Lizenztyp eines Erstkäufers. Das bedeutet, dass der Eigentümer dieser Mediendatei (*party -> uid* der URM-Lizenz) diese rechtmäßig erworben hat (beispielsweise über eine in Abbildung 3.2 dargestellte Datenquelle). Wenn dieser Eigentümer nun das Recht hat, diese Mediendatei an eine weitere Person weitergeben zu dürfen (*permission -> duplicate*), wird für diese Person eine neue Lizenz erzeugt. Diese Lizenz ist von einem abweichenden Lizenztyp, da es sich um eine Weitergabe handelt und nachvollziehbar sein soll, von welcher Person die Lizenz weitergegeben wurde. Eine solche Lizenz wird als Exportlizenz bezeichnet. Des Weiteren kann anhand der Exportlizenzen des Lizenzausstellers festgestellt werden, wie viele Lizenzen bereits für eine Musikdatei vergeben wurden. Bei einer Mediendatei mit begrenztem Recht zur Weitergabe kann also anhand der

Anzahl der Exportlizenzen und dem Begrenzungswert errechnet werden, wie viele Dateien bzw. Lizenzen noch legal weitergegeben werden dürfen. Die Exportlizenz unterscheidet sich nur wenig von der normalen Lizenz, wie in Abbildung 3.4 deutlich wird.

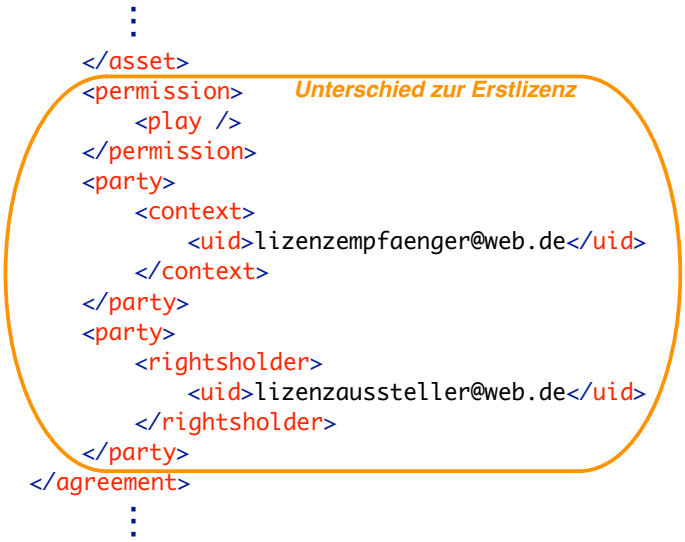


Abbildung 3.4: Beispiel einer URM-Exportlizenz
in Anlehnung an [Jahn, 2010]

Während der Bereich *asset* der Exportlizenz mit der Erstlizenz identisch ist, stellt sich der *permission*- und der *party*-Teil in abgeänderter Form dar. Hier bekommt der Lizenzempfänger kein *duplicate*-Recht mehr. Weiterhin gibt es nun zwei *party*-Bereiche. In dem *party*-Teil, der ebenfalls in Abbildung 3.3 vorhanden ist, ist die E-Mail-Adresse des Lizenzempfängers eingetragen. Das hinzugekommene *party*-Element beinhaltet in einem *rightsholder* die E-Mail-Adresse des Lizenzausstellers, so dass zusätzlich zu dem Eigentümer erkenntlich ist, von wem die Datei weitergegeben wurde.

Um die freie Verfügbarkeit von Mediendateien zu gewährleisten, wird die Lizenz getrennt von der Mediendatei abgespeichert [Jahn, 2010]. Das bedeutet, dass es eine Mediendatei und eine Lizenzdatei gibt, welche in unterschiedlichen Verzeichnissen zu finden und durch ein bestimmtes Verfahren miteinander verbunden sind.

Um zu einer Musikdatei die entsprechende Lizenz zu assoziieren (Musikdatei –> Lizenz) wird aus den MP3-Daten (Payload) einer MP3-Datei ein Hashwert gebildet, welcher sowohl den Lizenznamen (siehe auch Abbildung 3.5) bildet als auch im Lizenzattribut *asset* beschrieben ist und dementsprechend in dem vordefinierten Lizenzverzeichnis nachgeschlagen werden kann [Hundacker u. a., 2009].
Um wiederum von der Lizenz auf die Musikdatei schließen zu können (Lizenz –> Musikdatei), wird entweder der Dateipfad aus dem entsprechenden *dlocation*-Element oder der gehashte Wert aus dem *asset*-Attribut der Lizenz ausgelesen, woraus eindeutig die dazugehörige Mediendatei bestimmt werden kann [Hundacker u. a., 2009].

Wie eine URM-Verzeichnisstruktur aussehen kann, ist in Abbildung 3.5 dargestellt, einschließlich der verschiedenen Lizenztypen. Die Erstlizenzen liegen in dieser Darstellung direkt in dem Ordner *meinelizenzen* als ODRL-Dateien vor während die Exportlizenzen in dem Ordner *exportlizenzen* in dem jeweiligen Verzeichnis des Lizenzempfängers abgelegt sind.

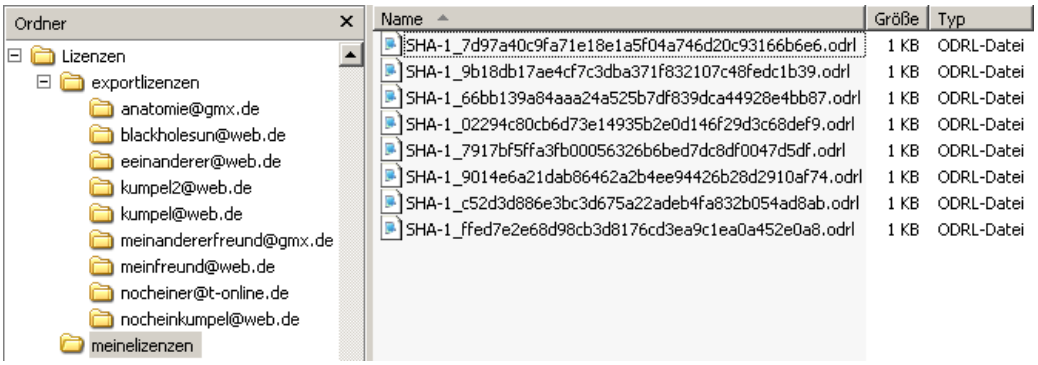


Abbildung 3.5: Verzeichnisstruktur der Lizenzdateien
[Jahn, 2010, S. 16]

Es wird deutlich, dass die Lizenzdateien in eine festgelegte Struktur eingebunden sind, wohingegen der Nutzer seine Mediendateien grundsätzlich dort ablegen kann, wo er es für sinnvoll erachtet. Dies ist bewusst so umgesetzt, da die Lizenzdateien in den meisten Fällen über entsprechende

Werkzeuge auf der Festplatte abgelegt werden und dies selten manuell vom Nutzer ausgeführt wird, währenddessen die Mediendateien des Öfteren in Eigenregie des Nutzers abgespeichert werden.

## 3.2 TURM-System

Als Startprojekt für die Umsetzung und Unterstützung von URM dient der aus einer Studienarbeit entstandene Prototyp einer ODRL-Mediathek, siehe [Jahn, 2010]. Dieses in der .NET-Programmiersprache C# umgesetzte Musikverwaltungsprogramm wurde zunächst dazu verwendet, anhand einer funktionsfähigen Plattform zu zeigen, wie Aspekte des URMs in Bezug auf Musikdateien in der Praxis verwirklicht werden können.

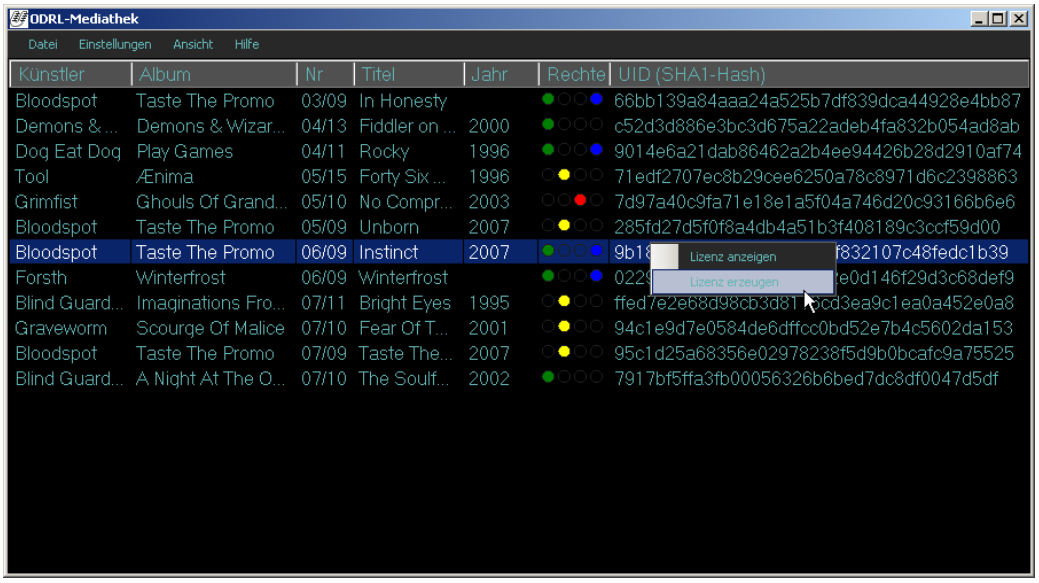


Abbildung 3.6: ODRL-Mediathek
in Anlehnung an [Jahn, 2010]

Eine beispielhafte Ansicht der ODRL-Mediathek wird in Abbildung 3.6 vorgestellt, in der eine Auflistung verschiedener Musikdateien, mit gebräuchlichen Informationen wie Künstlername, Albumtitel oder Songtitel zu sehen ist. Zwei Spalten fallen jedoch gegenüber bisherigen Musikprogrammen auf, die Information über die Nutzungsrechte und die An-

zeige des SHA1-Hashwerts, wodurch die URM-Umsetzung vorliegt. Die Nutzungsrechte für jede Musikdatei werden über ein Punktsystem mit verschiedenen Farben gekennzeichnet. Ein *grüner Punkt* bedeutet, dass der Nutzer diese Datei abspielen darf, während der *blaue Punkt* anzeigt, dass diese Mediendatei an Freunde weitergegeben werden darf. Mit einem *gelben Punkt* wird angezeigt, dass zu dieser Musikdatei keine valide Lizenz existiert und unklar ist, ob diese abgespielt oder weitergegeben werden darf. Ein *roter Punkt* signalisiert, dass diese Datei von einer illegalen Quelle stammt und der empfohlene Lösungsweg das Beziehen einer legalen Version ist [Hundacker u. a., 2009].

Aus verschiedenen Gründen, wie beispielsweise die Plattformunabhängigkeit, wurde nach der Erfahrung mit dem ODRL-Prototyp entschieden, ein neues System aufzubauen, in dem die Funktionalität der ODRL-Mediathek eine Teilmenge bilden soll. Dieses als Toolkit for Usage Rights Management (TURM) benannte System ist derzeit in der Entstehung und wird sowohl durch die Mitarbeiter der Professur Grimm der Universität in Koblenz als auch durch Studierende der Universität in Koblenz anhand von verschiedenen Qualifikationsarbeiten, darunter fallen sowohl Studien- und Bachelorarbeiten als auch Diplom- und Masterarbeiten, konzipiert und implementiert. Grundlage dieses Tools bildet die objektorientierte Programmiersprache Java, die bereits in Kapitel 2.3.4 genauer beschrieben wurde, um eine Plattformunabhängigkeit bieten zu können.

Das TURM-System wird derzeit mit den in Abbildung 3.7 aufgezeigten Komponenten ausgestattet, die teilweise noch in der Entwicklungsphase sind und im nächsten Abschnitt beschrieben werden.
Zu den Bestandteilen des TURMs gehören der Kern des Turms (*Turm (core)*), der *TurmFeatureManager*, der *FileAnalyzer*, der *MediaManager*, der *LicenseManager*, der *SignatureManager* und die Komponenten *CUP* in der Funktion eines P2P-Clients und das *UserInterface* als Benutzerschnittstelle.

*Turm (core)*
Der Kern des Turms beinhaltet die Hauptklasse, welche alle anderen

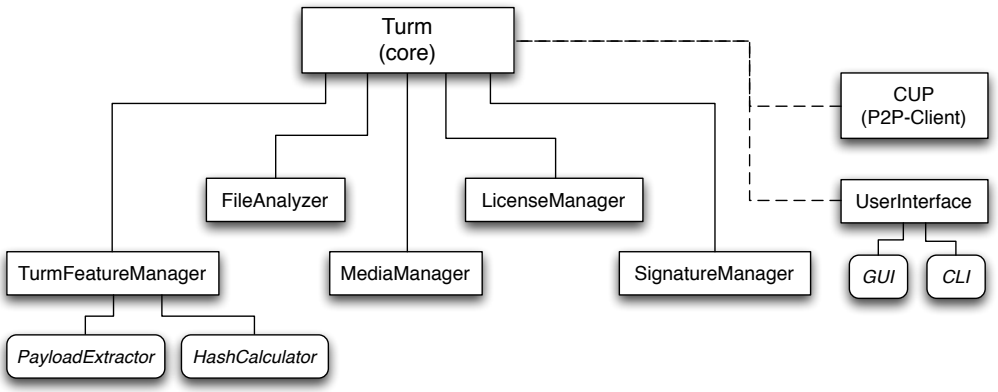


Abbildung 3.7: Komponenten des TURM-Systems

Klassen bzw. Bestandteile verwaltet. Das bedeutet, dass der Turmkern eine zentrale Stelle bildet, über die jede Komponente aufgerufen werden kann, so dass eine andere Klasse nicht immer eine neue Instanz erzeugen muss. Vor allem agiert der Turmkern als Hauptschnittstelle nach außen.

*TurmFeatureManager*
Diese Komponente versteht sich als generischer Funktionsmanager, der es ermöglicht, Funktionalitäten während der Laufzeit hinzuzufügen und zu entfernen. Zu den bereits implementierten Funktionen gehört ein *PayloadExtractor* und ein *HashCalculator*. Diese Bestandteile sind beispielsweise dafür zuständig, die MP3-Daten (Payload) aus einer MP3-Datei zu extrahieren (–> *PayloadExtractor*) und von dieser Payload den Hashwert zu berechnen (–> *HashCalculator*). Mit dem berechneten Hashwert kann die UID erstellt und das Musikstück identifiziert werden.

*FileAnalyzer*
Der FileAnalyzer soll in einem nächsten Projekt umgesetzt werden. Dabei geht es darum, Dateien, deren Rechte dem Nutzer unbekannt sind, überprüfen zu lassen, in dem z.B. die Datei auf Informationen über den Urheber durchsucht wird oder in einer Datenbasis überprüft wird, ob zu dieser Datei eine allgemeingültige Lizenz existiert.

*MediaManager und LicenseManager*
Die Zuständigkeit der Module MediaManager und LicenseManager liegt darin, die Dateien und Lizenzen mit ihren Informationen, wie ID3-Tags oder Lizenzinformationen, zu verwalten und anderen Komponenten des TURM-Systems zur Verfügung zu stellen. Dabei ist es möglich, über den MediaManager in den vorhanden MP3-Dateien und ID3-Tags nach einem bestimmten „Text“ zu suchen, derweil der LicenseManager für das Auslesen der Lizenzdaten, das Erzeugen neuer Lizenzen und deren Speicherung zuständig ist. Während der LicenseManager gerade im Rahmen einer Studienarbeit entwickelt wird und kurz vor der Fertigstellung ist, wird der MediaManager in naher Zukunft als Bachelorarbeit umgesetzt.

*SignatureManager*
Ein bereits fest eingeplanter Bestandteil des Toolkits for URM ist der SignatureManager, der später (siehe Kapitel 3.4) die digitalen Signaturen der Lizenzen verwalten und in naher Zukunft realisiert werden soll.

*CUP (P2P-Client)*
Der CUP ist das Hauptthema dieser Arbeit und wird sowohl in den nächsten Kapiteln beschrieben als auch praktisch umgesetzt.

*UserInterface*
Mit dem UserInterface werden Benutzerschnittstellen zur Verfügung gestellt, die es dem Benutzer ermöglichen beispielsweise den CUP zu steuern. Dabei gibt es ein Kommandozeilen-Interface (CLI Command Line Interface) für den CUP und ein noch in der Planung befindliches Projekt, welches sich die Umsetzung einer grafischen Benutzeroberfläche (GUI Graphical User Interface) zur Aufgabe macht. Für weitere Informationen über Benutzerschnittstellen siehe Kapitel 2.3.2.

Ein weiteres Projekt ist ein Web Service zum Überprüfen von digitalen Wasserzeichen, welcher bereits in Kapitel 2.1.3 als technische Umsetzung des DRMs vorgestellt wurde. Dieser Web Service wurde während einer Qualifikationsarbeit umgesetzt und kann später vom TURM-System,

z.B. als Teil des *FileAnalyzers*, sinnvoll eingesetzt werden. Weitere Informationen sind in [Hundacker, 2008] und in Kapitel 3.4 dargestellt. Der Benutzer spricht den Web Service mit Angabe der zu überprüfenden Datei an und bekommt entsprechend zurück, ob ein Wasserzeichen existiert. Ist dies der Fall, werden dem Benutzer die als Wasserzeichen gespeicherten Informationen angezeigt. Dieses Programm kann bisher nur Bilddateien verarbeiten, jedoch wurde die Erweiterung auf andere Medientypen bereits ins Auge gefasst.

## 3.3 URM-Anwendungsbereich

Wie bereits in Kapitel 2.1.3 thematisiert, nimmt seit einiger Zeit die Musikindustrie Abstand von DRM-Kopierschutzmechanismen. Durch diese Maßnahme wird der Nutzwert von digitalen Musikdateien der Nutzer entsprechend erhöht. Dies bedeutet nicht, dass diese mit Ihren Dateien beliebig verfahren dürfen.

Es stellt sich dem Nutzer die Frage, was dieser mit seinen erworbenen Musikdateien machen darf, ohne geltende Rechtsgrundlagen zu verletzen. Diese Frage lässt sich anhand der Geschäftsbedingungen des entsprechenden Anbieters (z.B. iTunes oder musicload) und des Urheberrechtsgesetzes (siehe Kapitel 2.1.2) definieren. Dabei ist es allerdings für einen unbedarften Nutzer nicht leicht, über diese Bestimmungen den Überblick zu behalten, um die eigenen Rechte konkret einordnen zu können.[3] An dieser Stelle greift das Usage Rights Management (URM) ein, dessen Grundkonzept auf von Nutzern selbstverwalteten Lizenzen beruht. Die Nutzer von digitalen Inhalten werden durch diverse Werkzeuge (z.B. die ODRL-Mediathek oder der CUP; siehe Kapitel 3.2) dabei unterstützt, den Überblick über die Rechte zu den einzelnen digitalen Mediendateien zu behalten. Passend zu diesem Anwendungsbereich wurde das auf der CeBIT

[3]URM-Pressemitteilung der CeBIT 2010 in Hannover; URL: http://donar.messe.de/exhibitor/007/120/159396/products/7460/links/20100211121944-URM_pressemitteilung.pdf; Stand: 16.01.2010

2010 in Hannover von der Forschungsgruppe „IT-Risk-Management“ der Universität Koblenz-Landau präsentierte URM-Plakat erstellt, welches als Ausschnitt in Abbildung 3.8 illustriert ist.


Abbildung 3.8: Ausschnitt aus dem URM-Plakat der CeBIT 2010
[Forschungsgruppe „IT-Risk-Management“, Universität Koblenz-Landau]

Das in Abbildung 3.8 dargestellte Musikverwaltungsprogramm stellt eine URM-Mediathek dar. Dies erkennt man an der Visualisierung (mittels Farbcode) der jeweiligen Rechte, die hinter den Musikstücken dargestellt ist. Diese Visualisierung wurde bereits in Kapitel 3.2 mit der ODRL-Mediathek beschrieben. Damit ist es dem Nutzer möglich, seine Rechte (z.B. darf abgespielt und weitergegeben werden) auf einen Blick erkennen zu können und sich auf der sicheren Seite zu bewegen (*URM-Feel Safe!*). Die Rechteinformationen werden in selbst generierten Lizenzen, welche vom Nutzer selbst verwaltet werden, anhand der Sprache ODRL abgespeichert und sind in einer offenen URM-Struktur eingebettet.

## 3.4 URM-Ausblick

Als ein weiterer Schritt in dem Forschungsgebiet URM ist geplant (siehe [Hundacker u. a., 2009]), dass URM nicht nur auf MP3-Dateien beschränkt bleibt wie es zum Zeitpunkt der Erstellung der Arbeit der Fall ist, sondern auf weitere digitale Inhalte (siehe Abbildung 2.1 in Kapitel 2.1.1) anwendbar wird. Dafür müssen die verschiedenen Medientypen analysiert werden, da es doch Unterschiede gibt. Beispielsweise sind die unterschiedliche Nutzung (hören, ansehen, lesen oder spielen bzw. statisch oder dynamisch), die differenzierenden Rechtsarten, welche für jeden Medientyp ein entsprechendes Rechtsprofil (ODRL-Profil) erfordern und die Akzeptanz bei den Nutzern des jeweiligen Mediums [Hundacker u. a., 2009] zu beachten. Außerdem gibt es bei den unterschiedlichen Medienarten verschieden starke Ausprägungen, wie erfolgreich hartes DRM (im Moment) ist.

Des Weiteren wird das TURM-System durch zusätzliche Funktionalitäten und die Integration von Tools ausgebaut und soll sich im Open Source Bereich unter der GNU General Public License (GPL) etablieren. Ein sich in Planung befindliche Erweiterung für das TURM-System befasst sich beispielsweise mit der Rechteverifikation von Musik. Das bedeutet, dass beispielsweise über einen Web Service eine Datenbasis angefragt wird, ob zu einer dem Anfrager gehörenden Musikdatei eine allgemeingültige Lizenz (z.B. Creative Commons) vorhanden ist und damit die Rechte des Anfragers bestimmt werden können. Eine Vorstufe zu der gerade angesprochenen Rechteverifikation kann die Anbindung des Web Services zur Überprüfung von digitalen Wasserzeichen an das TURM-System leisten. Dazu wird zunächst untersucht, ob ein digitales Wasserzeichen vorhanden ist und im nächsten Schritt die Ergebnisdaten der Wasserzeichen-Verifikation auf Hinweise über Rechte des Nutzers überprüft.

Eine andere mögliche Erweiterung des URMs ist es, Lizenzen von z.B. Musikshops digital signieren zu lassen, so dass eine solche signierte Lizenz als echter Beweis für die Urheberschaft der entsprechenden

Multimediadatei, selbst vor Gericht, eingesetzt werden kann [Hundacker u. a., 2009]. In gewisser Weise sind die vorhandenen Lizenzen bereits gerichtsfähig. Beispielsweise durch eine glaubhafte Darlegung vor dem Richter, dass die Lizenzen alle gewissenhaft (und korrekt) erstellt wurden. Jedoch ist die Beweiskraft geringer, als die der möglichen Lizenzen mit digitaler Signatur. Auch wenn sich URM zunächst ausschließlich als Aufklärungsinstrument versteht, ist es auf lange Sicht laut [Hundacker u. a., 2009] möglich, die Verwaltung der Nutzungsrechte an sinnvollen Stellen um stärkere DRM-Maßnahmen zu ergänzen [Hundacker, 2008].

# 4 Analyse und Entwurf

In folgendem Kapitel werden zunächst, gemäß des in Abbildung 1.2 dargestellten Wasserfallmodells, die Anforderungen betrachtet, die an den „Client for URM based P2P filesharing“ (CUP) gestellt werden (siehe Abschnitt 4.1) und in Abschnitt 4.2 auf das verwendete Netzwerkmodell eingegangen. Im Weiteren wird das dem CUP zugrunde liegende Konzept vorgestellt. Dieses ergibt sich aus den Phasen Softwareanalyse, Softwarearchitektur und Feindesign und wird anhand von verschiedenen UML-Diagrammtypen verdeutlicht. Des Weiteren wird sich in diesem Konzept mit den Schnittstellen zu den unterschiedlichen TURM-Komponenten auseinandersetzt (siehe Abschnitt 4.3).

## 4.1 Anforderungsanalyse

Da die vorliegende Diplomarbeit dem Bereich der Implementierungsarbeiten zugeordnet ist, spielen die softwaretechnischen Anforderungen an den CUP eine wichtige Rolle. Anhand einer Anforderungsanalyse (siehe Wasserfallmodell in Abbildung 1.2) wird, vor den Phasen Konzeptentwurf und Implementierung eindeutig festgelegt, welche Erfordernisse das zu entwickelnde System später erfüllen muss, kann und soll. Dementsprechend befinden wir uns im Softwarelebenszyklus, der in Kapitel 2.3.1 beschrieben und in Abbildung 2.6 dargestellt ist, in der Phase, die Problemstellung des CUPs zu analysieren und die Anforderungsdefinition abzugrenzen.

Die in dieser Diplomarbeit erstellte Anforderungsliste wurde in Anlehnung an [IEEE, 1998], dem IEEE-Standard zur Spezifikation von Software entwickelt. Einige Bestandteile dieser Vereinbarung wurden jedoch in soweit angepasst, dass es den Gegebenheiten dieser Diplomarbeit gerecht wird. Dies betrifft beispielsweise die vorgeschlagene Gliederung, die für diese Ausarbeitung nicht übernommen wurde um dem Rahmen dieser Ausarbeitung gerecht zu werden, da diese aus Sicht des Autors hauptsächlich für größere Projekte vorgesehen ist.
Die komplette Anforderungsspezifikation der vorliegenden Diplomarbeit ist im Anhang in Tabelle A.1 auf Seite 118 zu finden und zur besseren Übersicht in Funktionstypen unterteilt, worauf im weiteren Verlauf dieses Abschnitts genauer eingegangen wird.
In diesem Kapitel wird primär auf die wichtigsten Bestandteile der Anforderungen eingegangen, da die meisten Anforderungen selbst erklärend sind.
Die Grundlage der einzelnen Anforderungen bilden sowohl die Richtlinien des, der Diplomarbeit zugrunde liegenden, Usage Rights Managements (URM) als auch die verschiedenen geführten Gespräche des Autors mit den Betreuern der vorliegenden Ausarbeitung.

Im Zuge dieser wissenschaftlichen Arbeit ist eine Java-Anwendung zu entwickeln, die es den Benutzern ermöglicht, Dateien mit URM-Lizenzen untereinander auszutauschen und sich somit legal verhalten zu können. Gemäß den Vorgaben ist diese Anwendung als eine Komponente zu implementieren, welche in das TURM-System integriert wird und damit die Funktionalität des TURMs um einen „Client for URM based P2P filesharing“ (CUP) erweitert (siehe *Anforderung 01*).
Wie in den *Anforderungen 02 - 05* beschrieben, wird der CUP-Benutzer über die Kommandozeile die Anwendung starten können, wobei der Anwender verschiedene Informationen, wie z.B. Benutzername und ob der CUP als Client oder Server gestartet werden soll, angeben muss. Sind die vom Benutzer getätigten Eingaben korrekt, soll der CUP gestartet werden, andernfalls muss dem Benutzer angezeigt werden, dass die

eingegebenen Parameter nicht korrekt sind und der CUP nicht startet. Die korrekt übergebenen Eingaben der Kommandozeile werden, wie in *Anforderung 06* dargelegt, zur späteren Verwendung vom CUP in dem sogenannten ConfigurationManager des TURMs abgespeichert. Dieser „ConfigurationManager“ fungiert als Datenbasis und wird im späteren Kapitel 5.2 noch genauer erörtert.
Die vom CUP benötigten Netzwerkfunktionen sollen nach *Anforderung 7* mit einem Framework namens JXTA, welches in Kapitel 5 ausgeführt ist, implementiert werden, um auf bereits geschriebenen und etablierten Code zurückzugreifen und die in die Erstellung des Frameworks eingegangene Kompetenz zu nutzen. Weiterhin wird, siehe *Anforderung 08*, der in dieser Diplomarbeit zu entwickelnde CUP zunächst entweder als Client oder als Server gestartet. Das bedeutet, wenn der CUP als Client gestartet ist, werden nur die Funktionalitäten eines Clients, wie Dateien suchen und mit Lizenz downloaden, ausgeführt, während der CUP handelnd als Server nur Funktionalitäten, wie Dateien mit Lizenz zur Verfügung stellen und den Weitergabestatus der Dateien zu prüfen, ausführt. Um einen Überblick über die Anforderungsliste in Tabelle A.1 zu bieten, wurde jede einzelne Anforderung in der letzten Spalte mit der entsprechenden Erscheinungsform „als Client handelnd“, „als Server handelnd“ oder „Allgemein“ gekennzeichnet.
Wie bereits im vorherigen Absatz angedeutet und in den *Anforderungen 09 - 13* spezifiziert, muss der Benutzer Dateien anderer Peers sowohl suchen als auch zusammen mit der Lizenz herunterladen können. Die vorhandenen Dateien müssen auf das Recht der Weitergabe überprüft werden. Nur Dateien mit erlaubter Weitergabe dürfen dem Benutzer zum Suchen und zum Download mit der zugehörigen Lizenz (Exportlizenz) bereitgestellt werden.
Der CUP kann nicht nur, wie weiter oben erläutert und in *Anforderung 03* spezifiziert, als Client oder Server gestartet werden, sondern auch als sogenannte Freund-Funktion, einschließlich einer E-Mail-Adresse, aufgerufen werden. Dieser Aufruf bewirkt, dass der angegebene Benutzer anhand seiner E-Mail-Adresse in die Freundesliste des CUP-Benutzers

eingetragen wird (siehe *Anforderung 14*).
Wie in *Anforderung 15* beschrieben, wird der in dieser Diplomarbeit zu entwickelnde CUP zunächst nur ein verbreitetes Medienformat (das MP3-Format) unterstützen, während die restlichen digitalen Inhalte, wie AAC, E-Books oder Bilder, durch ein späteres Projekt hinzugefügt werden.

## 4.2 CUP-Aufbau

Der CUP wird mit Hilfe der objektorientierten Programmiersprache Java entwickelt, da diese Programmiersprache für die Erstellung von größeren Systemen, wie es der TURM ist, die plattformunabhängig sein sollen prädestiniert ist und bereits von dem entstehenden TURM erfolgreich verwendet wird. Alternativen zu Java werden an dieser Stelle nicht untersucht, da seitens der Forschungsgruppe „IT-Risk-Management“ die Programmiersprache Java als strategische Plattform festgelegt wurde.
Nach Besprechung mit den entsprechenden Betreuern dieser wissenschaftlichen Arbeit wurde herausgestellt, dass der CUP als Komponente für den TURM entwickelt werden soll. Entsprechend kann der CUP nur mit dem TURM existieren und wird benötigte Funktionalitäten, die der TURM anbietet, durch Schnittstellen zu den einzelnen TURM-Komponenten einbinden.

Da der zu implementierende Prototyp über das Netzwerk kommunizieren muss, wird im Folgenden das dem CUP zugrunde liegende Netzwerkmodell dargelegt. Dieses ist das bereits in Kapitel 2.2.2 beschriebene Peer-to-Peer-System (hybrid). Der CUP soll dafür ausgelegt werden, dass sehr viele Personen miteinander in Verbindung stehen können. Digitale Inhalte wie Musik- oder Bilddateien sollen effizient ausgetauscht werden können. Der CUP wird nicht als zentralisiertes P2P-Netzwerk eingesetzt, sondern bildet ein reines P2P-System, das bedeutet, dass es keine zentrale Instanz (Server) gibt und somit keine Serveradministration durchgeführt werden muss.
Für den Aufbau der P2P-Anwendung des CUP-Systems wird die in

Kapitel 2.2.3 beschriebene JXTA-Technologie eingesetzt. Diese bietet nach [Oaks u. a., 2002] eine gemeinsame Plattform, die das einfache Implementieren von P2P-Anwendungen, die miteinander auch auf unterschiedlichen Endgeräten kompatibel sind und Daten austauschen können, ermöglicht.

## 4.3 Konzept

Dieser Abschnitt behandelt den Konzeptentwurf, der im Wasserfallmodell des CUPs (siehe Abbildung 1.2 auf Seite 12) als zweiter Schritt eingeordnet ist und umfasst die Aufgaben des Entwerfens und Spezifizierens der Architektur bzw. Bausteinspezifikation des Softwarelebenszyklus (siehe Abbildung 2.6 auf Seite 34).

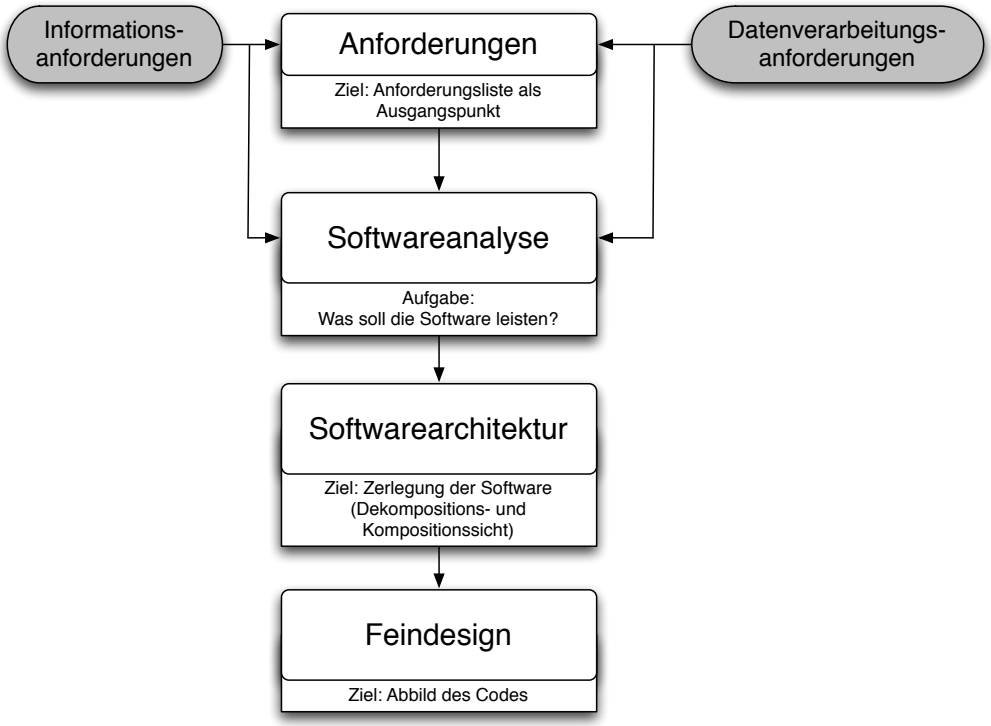


Abbildung 4.1: Phasen der Softwareentwicklung
in Anlehnung an [Rupp u. a., 2007]

Nach [Rupp u. a., 2007] kann der Konzeptentwurf in drei Phasen der Softwareentwicklung unterteilt werden, die für die Konzepterstellung des CUPs durchlaufen werden, bevor die eigentliche Implementierung begonnen wird. Dies sind die Softwareanalyse, die Softwarearchitektur und das Feindesign, welche in der Übersicht in Abbildung 4.1 dargestellt sind.
Eine notwendige Voraussetzung dieser Schritte ist nach [Rupp u. a., 2007] eine vorliegende Anforderungsliste, die für den CUP bereits im vorherigen Kapitel 4.1 beschrieben wurde und im Anhang auf Seite 118 zu finden ist. Mit diesen Anforderungen als Ausgangsbasis wird mit der Softwareentwicklung begonnen.

In den folgenden drei Kapiteln wird auf die Phasen der Softwareentwicklung nach [Rupp u. a., 2007] eingegangen und dargelegt, welche Rolle die verschiedenen Schritte für den entstehenden CUP einnehmen. Die einzelnen Schritte werden vom Autor bewusst nicht als Unterkapitel des Konzept-Kapitels gegliedert, sondern wegen des hohen Stellenwertes und des hohen Umfangs in Hauptkapiteln ausgeführt.

## 4.4 Softwareanalyse

Die Aufgabe der Softwareanalyse ist es, zu untersuchen, was die Software nach Fertigstellung leisten soll. Dabei wird nur die Software betrachtet die intern realisiert werden soll, während Applikationen die extern entwickelt werden oder zugekauft sind in dieser Phase irrelevant sind. Für den CUP bedeutet es, dass die komplette Software zu betrachten ist, da das komplette CUP-System intern realisiert wird.
Weiterhin können verschiedene Teile der Software entweder zusammen oder durch mehrere Softwareanalysen untersucht werden. An dieser Stelle weist [Rupp u. a., 2007] darauf hin, dass eine gemeinsame Betrachtung vorzuziehen ist, da hier die Vorteile, wie z.B. die einfachere und effizientere Zusammenarbeit der Analytiker, überwiegen. Da der CUP im Rahmen dieser Diplomarbeit lediglich von einer Person entwickelt wird, wird

deshalb nach [Rupp u. a., 2007] für den CUP nur eine Softwareanalyse durchgeführt wird.

Die Ziele der Softwareanalyse sind zum einen, die Anforderungen des Kunden zu untersuchen, um ihm Auskunft über die Machbarkeit geben zu können und zum anderen, tiefergehende und ausführlichere Richtlinien für die weitere Entwicklung aufzustellen. Da das zugrunde liegende CUP-Projekt eine Diplomarbeit ist und der Autor, gleichzeitig Entwickler, aus der gegebenen Aufgabenstellung die Anforderungen selbständig erarbeitet hat, gibt es im klassischen Sinne der Softwareanalyse keinen Kunden. Im weiteren Sinne können jedoch die Betreuer dieser Arbeit bzw. die Forschungsgruppe „IT-Risk-Management“ als Auftraggeber angesehen werden. Hinzu kommt, dass bereits beim Erstellen der Anforderungen das System auf Umsetzung überprüft wurde und detaillierte Vorgaben gemacht wurden (siehe dazu Kapitel 4.1).

Im Weiteren wird auf wichtige Aspekte der Softwareanalyse des CUPs eingegangen. Das Resultat dieser Analyse, ein Use-Case-Diagramm, welches in Abbildung 4.2 visualisiert ist, wird vorgestellt und eine detaillierte Verhaltensmodellierung wird beschrieben.

### 4.4.1 Use-Case-Analyse

Während der Durchführung der Analyse des CUPs findet zunächst die **Definition der Systemgrenzen** statt. Dazu wird die Sicht von außen auf das System untersucht, indem die Akteure des CUPs anhand der in Kapitel 4.1 beschriebenen Anforderungsliste (siehe Anhang Seite 118) identifiziert werden. Dies sind auf der einen Seite der *Nutzer* des CUPs und auf der anderen Seite die vorhandenen *Peers* im Netzwerk. Der Akteur Nutzer in Rolle einer Person wurde bewusst für die linke Seite des Use-Case-Diagramms gewählt, wenngleich dort ein „Client“, als Repräsentation der entsprechenden Clientfunktion, hätte als Akteur eingesetzt werden können, um dem unbedarften Leser einen besseren Überblick zu bieten. Auf der rechten Seite werden die verschiedenen Rechner mit gestartetem CUP

durch drei Peers repräsentiert, wobei jeder einzelne Peer erwartungsgemäß auch die Funktion Client beinhaltet, diese jedoch nicht in dieses Diagramm einmodelliert ist, damit es nicht unübersichtlich wird.

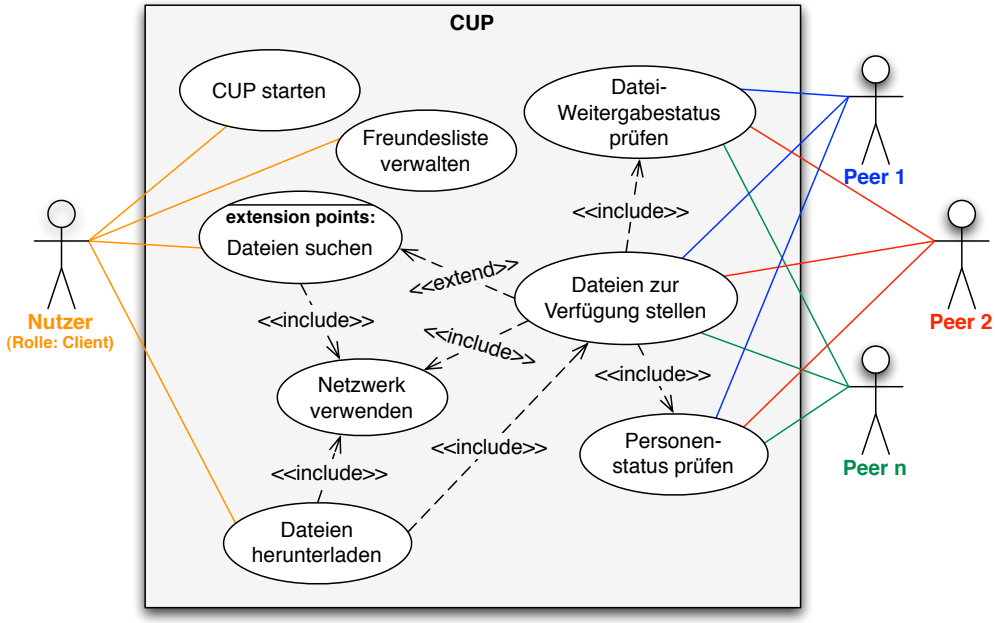


Abbildung 4.2: Use-Case-Diagramm des CUPs

Des Weiteren spielt in der CUP-Softwareanalyse das **Beschreiben der Funktionalitäten** eine wichtige Rolle. Das Use-Case-Diagramm in Abbildung 4.2 stellt nicht nur dar, welche Benutzer (Akteure) mit dem CUP interagieren, sondern bildet weiterhin die wesentlichen Funktionen (in Form der Anwendungsfälle bzw. Use Cases) ab, die das CUP-System den Benutzern bieten soll. Außerdem gibt das Diagramm durch die Zugehörigkeit der einzelnen Akteure zu den verschiedenen Funktionen eine Vorgabe für die Beziehungen untereinander. Die Systemgrenze umschließt das ganze CUP-System, während außerhalb dieser Grenze die im vorherigen Absatz identifizierten Akteure abgebildet sind. Innerhalb des CUPs befinden sich die ermittelten Use-Cases, die anhand der Anforderungen in Tabelle A.1 auf Seite 118 abgeleitet werden konnten. Diese sind die vom Nutzer beeinflussbaren Anwendungsfälle *CUP starten*,

*Dateien suchen*, *Freundesliste verwalten* und *Dateien herunterladen* und die von den mit den Peers in Verbindung stehenden Use-Cases *Datei-Weitergabestatus prüfen*, *Personenstatus prüfen* und *Dateien zur Verfügung stellen*. Der Use-Case *Netzwerk verwenden* steht in keiner Beziehung zu einem Akteur, sondern tritt mit den entsprechend dargestellten Anwendungsfällen in Beziehung. Mittels der «*include*»-Beziehungen wird visualisiert, dass beispielsweise der Use-Case *Dateien herunterladen* das Verhalten des Use-Cases *Dateien zur Verfügung stellen* importiert und *Datei-Weitergabestatus prüfen* von *Dateien zur Verfügung stellen* importiert wird. Unter Zuhilfenahme der «*extend*»-Beziehung wird verdeutlicht, dass der Anwendungsfall *Dateien suchen* durch den Anwendungsfall *Dateien zur Verfügung stellen* erweitert werden kann, aber nicht muss. Eine Verfeinerung der genannten Use-Cases wird im Feindesign (siehe Kapitel 4.6) durchgeführt und erläutert.

Für die Sicht von außen auf das CUP-System und die Beschreibung der Funktionalitäten des CUP-Systems wurde der Diagrammtyp Use-Case verwendet, da diese Form der Darstellung selbst von Personen die bisher noch nichts mit UML zu tun hatten zu verstehen ist. Auch wenn Use-Case-Diagramme nur einen sehr groben Überblick über ein System bieten, kann diese Darstellung als Grundlage zur Anregung von Diskussionen und Aufdeckung von Problemen genutzt werden. Das in Abbildung 4.2 dargestellte Use-Case-Diagramm wurde auf Basis von geführten Interviews mit Experten im Bereich der Informatik erstellt und verfeinert.

### 4.4.2 Verhaltensmodellierung

Um das Verhalten für ausgewählte Anwendungsfälle aus dem Use-Case-Diagramm detailliert beschreiben zu können, können nach [Rupp u. a., 2007] an dieser Stelle weitere Abbildungen herangezogen werden. Dabei richtet sich die Auswahl der Detaillierung nach der Wichtigkeit für die Analyse. Muss zu einem Anwendungsfall viel an Wissen dokumentiert werden, ist nach [Rupp u. a., 2007] eine detaillierte Modellierung des Verhaltens sinnvoll.

Darauf aufbauend wird im Folgenden der Anwendungsfall *Netzwerk verwenden* genauer betrachtet.

Der Use-Case *Netzwerk verwenden* steht für das zur Verfügung stellen der Kommunikation über ein P2P-Netzwerk. Die zu implementierende P2P-Anwendung wird anhand der bereits in Kapitel 2.2.3 beschriebenen JXTA-Technologie aufgebaut und umgesetzt. Das in Abbildung 4.3 visualisierte P2P-Konzept des CUPs stellt die für diese Diplomarbeit konzipierte Funktionsweise des Netzwerks über JXTA dar. Ein mögliches Szenario wird im Weiteren erörtert.

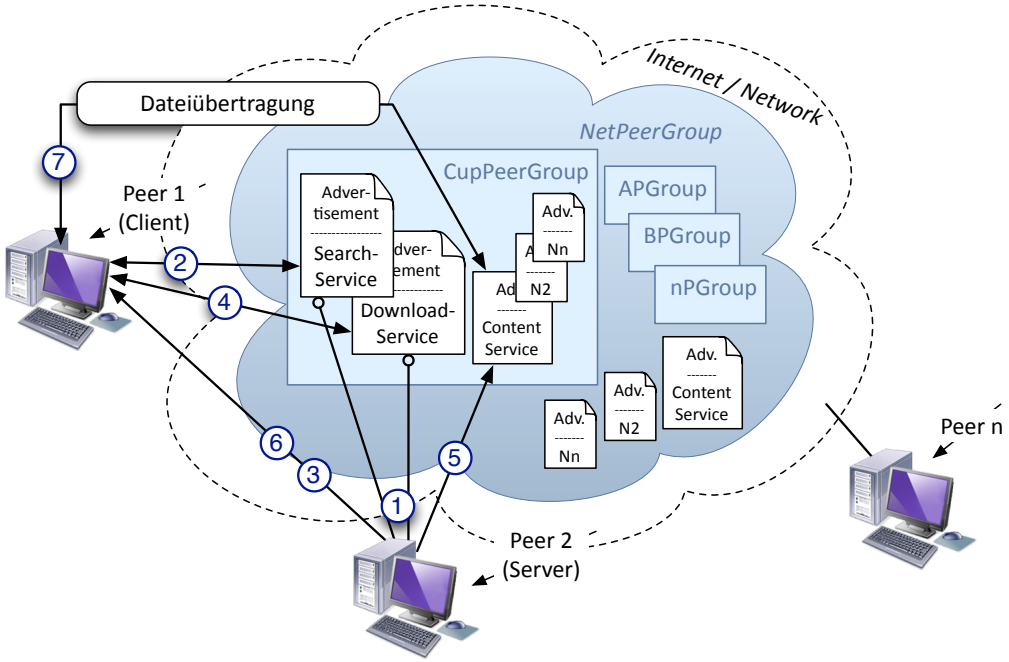


Abbildung 4.3: P2P-Konzept des CUPs

Die in Abbildung 4.3 anhand von Rechnern dargestellten *Peers 1, 2 und n* stellen die JXTA-Peers dar, die sich im Internet bzw. im Netzwerk befinden. Peer 1 handelt als Client, Peer 2 übernimmt die Funktionen des Servers und Peer n stellt die weiteren Peers dar, die sich im Netzwerk befinden und deren Rollen sowohl Clients als auch Server sein können.
In ① instanziiert Peer 2 zunächst die *NetPeerGroup*, bevor er die

*CupPeerGroup* erzeugt. Peer 2 befindet sich also nun sowohl in der NetPeerGroup, als auch in der CupPeerGroup, wobei jedoch die CupPeerGroup die momentan aktive PeerGroup ist. Dann veröffentlicht Peer 2 den *SearchService* und den *DownloadService* in der CupPeerGroup, jeweils durch ein entsprechendes JXTA-Advertisement. Die Services werden nun beide von Peer 2 überwacht, um auf Anfragen reagieren zu können. Es ist auch möglich, dass noch andere Peers dieselben Services anbieten und in dem Fall auf die gleichen Service-Anfragen wie Peer 2 reagieren können.
In (2) instanziiert Peer 1 sowohl die NetPeerGroup als auch die CupPeerGroup und sucht in der CupPeerGroup nach dem SearchService. Peer 1 findet das von Peer 2 angebotene SearchService-Advertisement und lädt sich die Informationen des Advertisements herunter. Anhand der heruntergeladenen Beschreibung erstellt Peer 1 ein SearchService-Objekt und stellt an dieses Objekt eine Anfrage mit dem entsprechenden Suchtext. Diese Anfrage wird automatisch von der SearchService-Instanz an das Netzwerk bzw. die CupPeerGroup weitergeleitet.
Alle Peers, die den SearchService am überwachen sind, bekommen nun diese Anfrage übermittelt. In unserem Szenario ist es nur Peer 2, welcher anhand des Suchtextes eine entsprechende Ergebnisliste erstellt, falls passende Dateien gefunden werden. Diese Liste wird in (3) über die entsprechenden JXTA-Protokolle direkt an Peer 1 gesendet.
Ist eine passende Datei vorhanden, sucht Peer 1 in (4) zunächst in der CupPeerGroup das Advertisement des DownloadServices und lädt sich wiederum die entsprechenden Informationen des Advertisements. Peer 1 sendet eine DownloadService-Anfrage mit der aus der Antwortliste extrahierten Datei-ID (Hashwert) an den DownloadService, welche Peer 2 übermittelt bekommt.
Peer 2 ermittelt nun die gewünschte Datei und erzeugt eine entsprechende Exportlizenz. Dann werden in (5) beide Dateien von Peer 2 in dem vom JXTA-Framework zur Verfügung gestellten ContentService zum Download bereitgestellt. Peer 1 bekommt nun von Peer 2 in (6) die notwendigen Informationen, um die Dateien vom ContentService herunterladen zu können.

Im letzten Schritt (7) wendet sich Peer 1 an den ContentService, der direkte Verweise zu den auf Peer 2 lokalisierten Daten enthält, und lädt die Datei und die Exportlizenz herunter.

An dem vorgestellten JXTA-Konzept ist zu beachten, dass bereits die NetPeerGroup eigene Services bereitstellt, wie z.B. der ResolverService oder der DiscoveryService, die für verschiedene Aktionen, wie das Entdecken der Advertisements oder der anderen Peers verantwortlich sind. Diese Services werden in dem P2P-Konzept des CUPs für die CupPeerGroup direkt übernommen, damit diese Services genutzt werden können.

## 4.5 Softwarearchitektur

Basierend auf den Ergebnissen der Softwareanalyse (Kapitel 4.4) wird in der Phase der Softwarearchitektur (siehe Abbildung 4.1 auf Seite 65) die zu betrachtende Software in kleinere Teile zerlegt. Das Ziel einer solchen Zerlegung ist es, die Software so zu unterteilen, dass ihre Komponenten im weiteren Vorgehen überwiegend unabhängig voneinander implementiert werden können und jegliche Funktionalität nur einmal implementiert werden muss. Dieser Vorgang findet in der sogenannten „Dekompositionsschicht“ statt. Das Gegenstück dazu ist die „Kompositionsschicht“, in der die einzelnen Komponenten wieder zu einer lauffähigen Software zusammengesetzt werden. Der CUP in seiner Rolle als Plugin ist bereits als eine Komponente des bestehenden Systems TURM zu sehen, stellt an sich durch die Größe des Projekts keine hohe Komplexität dar und wird von einer Person realisiert. Diese Voraussetzungen sind ausschlaggebend dafür, dass die hier beschriebene Phase der Softwarearchitektur in dieser Diplomarbeit keine Anwendung findet und der gesamte CUP als eine Komponente gesehen wird, die nicht in weitere Subkomponenten zerlegt werden sollte. Die weitere Aufteilung und die Verfeinerung der Use-Cases aus Kapitel 4.4 wird dem nachfolgenden Schritt, dem Feindesign überlassen.

## 4.6 Feindesign

Der letzte Schritt vor der Implementierung des CUPs ist das Feindesign (siehe Abbildung 4.1 auf Seite 65). Die entstehende Software bzw. die in der Softwarearchitektur gefundenen Komponenten werden so beschrieben, dass der Übergang zur Implementierung möglichst klein ist. Das Ziel dieses Verfahrens ist es, einen leichteren Einstieg in das Verstehen des Codes zu bekommen und bereits jetzt Entscheidungen treffen zu können, die später bei der Bewertung einer direkten Codierung mehr Aufwand bedeuten würden.

Waren die bis jetzt beschriebenen Phasen der Softwareentwicklung größtenteils unabhängig von der Programmiersprache, die verwendet werden soll, spielt in dieser Vorstufe der Implementierung diese Randbedingung eine entscheidende Rolle. Vor allem die Unterscheidung von „objektorientiert“ und „nicht-objektorientiert“ tritt hier in den Vordergrund. Bei beiden Programmierstilen wird von [Rupp u. a., 2007] für das Feindesign empfohlen, eine unterschiedliche Vorgehensweise anzuwenden, um ein optimales Ergebnis zu erreichen und somit die Lücke zwischen Feindesign und Code so gering wie möglich zu halten. Da die für den CUP zu verwendende Programmiersprache Java objektorientiert ist, wird das Vorgehen des Feindesigns für eine objektorientierte Implementierung genutzt, indem die verschiedenen Verantwortlichkeiten zu Objekten bzw. Klassen zusammengefasst werden und somit die differenzierenden Funktionalitäten des CUPs und vereinzelt des TURMs betrachtet werden können. Es werden alle Komponenten, die nicht weiter zerlegt werden sollen einzeln betrachtet, das bedeutet für diese Diplomarbeit, dass dieser Vorgang nur einmal durchlaufen wird, da der CUP als eine Komponente zur Verfügung steht.
Dazu werden im ersten Teil des CUP-Feindesigns die in Kapitel 4.4 modellierten und in Abbildung 4.2 dargestellten Anwendungsfälle anhand von Aktivitäts- und Sequenzdiagrammen verfeinert und im nächsten Teil die

Objekte bzw. Klassen und deren Beziehungen zueinander mit Hilfe eines Klassendiagramms abgebildet.

### 4.6.1 Verfeinerung der Anwendungsfälle

Nachdem in der CUP-Softwareanalyse (siehe Kapitel 4.4) die Use-Cases identifiziert und die beteiligten Akteure zugewiesen wurden, wird nun jeder der Use-Cases einzeln betrachtet. Um den Anforderungen eines Feindesigns zu entsprechen, wird vorerst ein Aktivitätsdiagramm entwickelt. Dieses spiegelt den Ablauf der CUP-Implementierung wider und reflektiert somit zusätzlich zu den einzelnen Anwendungsfällen des Use-Case-Diagramms in Abbildung 4.2 die verschiedenen Aktionen (Einzelschritte, in denen etwas getan wird) bzw. Objektknoten (symbolisieren beteiligte Daten) in der entsprechenden Reihenfolge. Das modellierte Aktivitätsdiagramm des CUP-Programmierablaufs findet sich in Abbildung 4.4 und wird im Folgenden näher erörtert.
Alle in schwarz gezeichneten Elemente und Linien entstammen der Sprache UML und stellen im Gesamten ein nach UML spezifiziertes Aktivitätsdiagramm dar. Anhand des Aktivitätsdiagramms lässt sich zwar der Programmierablauf erkennen, jedoch fehlt die Übersetzung zu den bereits identifizierten Use-Cases, da die Programmier-Reihenfolge nicht genau nacheinander die Anwendungsfälle abarbeitet. Aus diesem Grund gibt es die in Farbe gezeichneten, gestrichelten Linien und Bezeichnungen. Diese stellen gewissermaßen eine vom Autor erweiterte UML-Notation dar, um den Bezug zwischen den einzelnen Anwendungsfällen des Use-Case-Diagramms und der Reihenfolge der verschiedenen Aktivitäten herzustellen, die während der Implementierung ausgeführt werden, gehören jedoch nicht zum standardisierten UML-Sprachraum.

Um die jeweils aus mehreren Aktionen bestehenden Use-Cases der Abbildung 4.4 nachvollziehen zu können, werden im Folgenden die einzelnen Aktionen bzw. Objektknoten betrachtet, wobei die mit einem hohen Detailierungsgrad weiterhin in Anhang B ab Seite 119 durch Aktivitäts- und Sequenzdiagramme visualisiert sind.

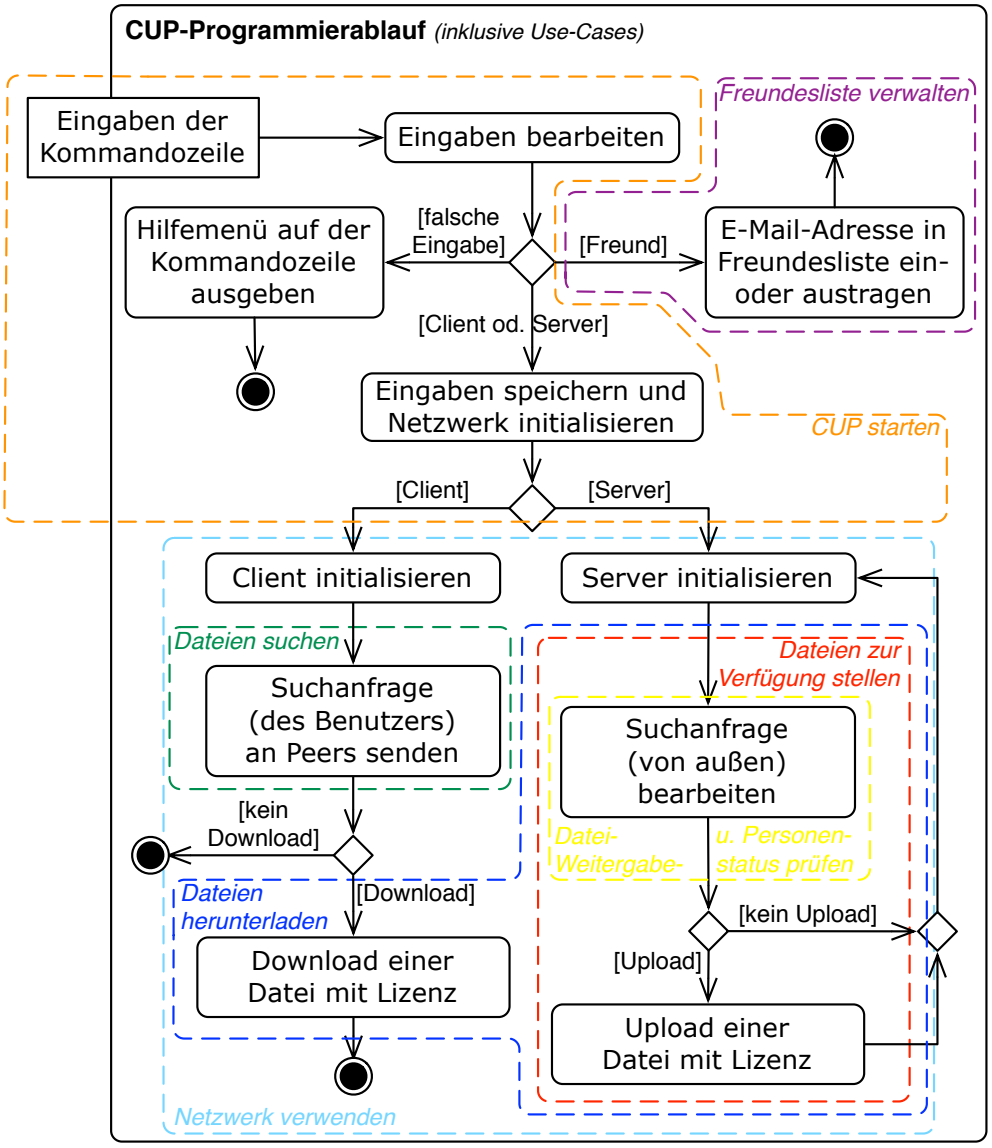


Abbildung 4.4: Aktivitätsdiagramm des CUP-Programmierablaufs

Um die Übersicht des Lesers zu fördern wurde, wie in Abbildung 4.5 gezeigt, der Grundriss des Haupt-Aktivitätsdiagramms mit dem Programmierablauf des CUPs anders gewählt (abgerundete Ecken, keine umrandete und mit Präfix vorgesetzte Bezeichnung) als das Format der Diagramme, die eine Verfeinerung von einzelnen Aktionen abbilden (Ecken nicht

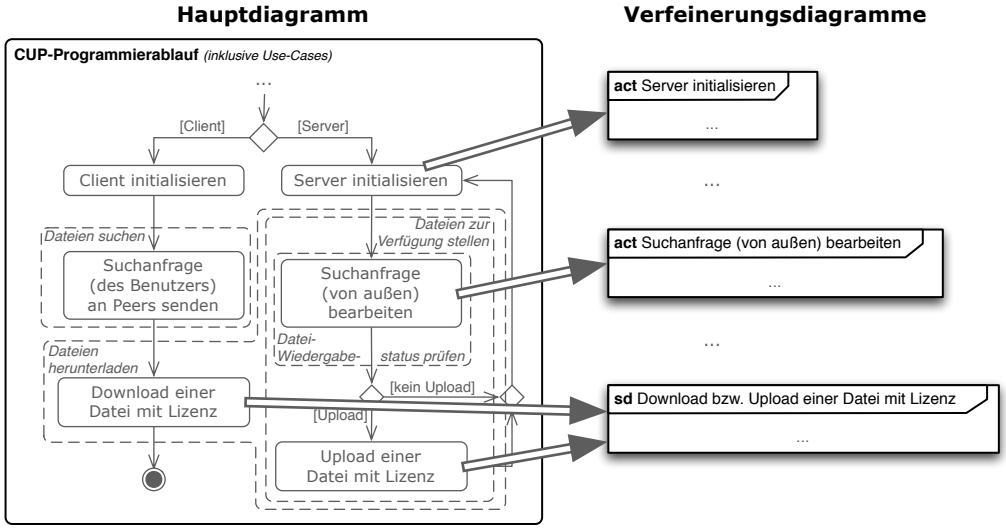


Abbildung 4.5: Formatierung der UML-Diagramme

abgerundet, Bezeichnung mit Präfix *act* und spezieller Umrandung).
In manchen Aktionen eines Aktivitätsdiagramms wurden ebenfalls für eine bessere Übersicht kleine Sequenzdiagramme eingebaut, welche wiederum eine Erweiterung der UML-Notation durch den Autor darstellt. Darüber hinaus ist zu beachten, dass die Reihenfolge der im folgenden beschriebenen Aktionen bewusst so gewählt ist, vor allem das die Client- und Serverfunktionen ab dem Splittungsvorgang abwechselnd erläutert werden, denn es müssen beide Seiten (Client- und Serverseite) existieren, da immer mindestens zwei laufende CUP-Anwendungen miteinander in einer bestimmten Abfolge kommunizieren (siehe dazu auch Abbildungen B.6 und B.9 in Anhang B).

*Eingaben der Kommandozeile*
Der Programmablauf beginnt mit dem einzigen Objektknoten des in Abbildung 4.4 dargestellten Aktivitätsdiagramms (Eingaben der Kommandozeile). Dieser Objektknoten repräsentiert die in Abbildung 4.6 gezeigten Parameter, die auf der Kommandozeile im CUP-Hilfemenü zu finden sind. Das Hilfemenü enthält alle zum Start des CUPs relevanten

Informationen bezüglich der Programmparameter (wie z.B. Benutzername oder E-Mail-Adresse des Benutzers).

**obj** Eingaben der Kommandozeile

```
usage: Cup
 -u, --user <arg>                          ->the username. (required)
 -i, --incomingFilesPath <arg>             ->the path for the incoming files where
                                           the general licenses saved. (required)
 -l, --licensePath <arg>                   ->the path for the incoming licenses
                                           where the general licenses saved.
                                           (required)

 -af, --addFriend <arg>                    ->the e-mail address of the friend, who
                                           add to the friendlist.
 -df, --deleteFriend <arg>                 ->the e-mail address of the friend, who
                                           delete from the friendlist.

 -c, --client                              ->cup behave like a client. (either
                                           client or server can be selected)
 -s, --server                              ->cup behave like a server. (either
                                           client or server can be selected)

 -e, --email <arg>                         ->the user e-mail address. (required)
 -a, --search  <arg>                       ->the searchstring. (required, if client
                                           is selected)

 -rlip, --relayIPAddresses <arg>           ->a list of noted relay peers with value
                                           separator ','.
 -rnip, --rendezvousIPAddresses <arg>      ->a list of noted rendezvous peers with
                                           value separator ','.
 -rl, --relayPeer                          ->the own computer is a relay peer.
 -rn, --rendezvousPeer                     ->the own computer is a rendezvous peer.

 -h, --help                                ->shows this help.
```

Abbildung 4.6: CUP-Hilfemenü der Kommandozeile

*Eingaben bearbeiten*

Nachdem der Benutzer die Parameter in der Kommandozeile eingegeben hat, werden diese auf Korrektheit überprüft und entsprechend gedeutet. Das heißt, wenn die Eingaben fehlerbehaftet sind, wird das CUP-Hilfemenü auf der Kommandozeile ausgegeben und der CUP nicht gestartet. Wurde die E-Mail-Adresse eines Freundes eingegeben, wird diese in der Freundesliste des CUPs je nach Aufrufparameter abgespeichert oder aus dieser entfernt. Wurde die Funktion Client oder Server

ausgewählt, werden die Eingaben gespeichert und das Netzwerk initialisiert, was in folgendem Absatz genauer erörtert wird.

*Eingaben speichern und Netzwerk initialisieren*
Die über die Kommandozeile übergebenen Parameter werden in einer entsprechenden Komponente des TURMs (benannt als ConfigurationManager und im späteren Verlauf dieser Arbeit noch genauer spezifiziert) für eine anschließende Nutzung abgespeichert. Anhand der Kommandozeilen-Parameter wird ermittelt, ob die Client- oder Serverfunktion gewählt wurde und die entsprechende Funktion aufgerufen. Dieses Vorgehen ist in Abbildung B.1 auf Seite 120 als Aktivitätsdiagramm dargestellt.

*Client initialisieren*
Damit die Clientfunktionen *Suchanfrage (des Benutzers) an Peers senden* und *Download einer Datei mit Lizenz* ausgeführt werden können, werden an dieser Stelle die erforderlichen Informationen aus dem bereits angesprochenen ConfigurationManager ausgelesen. Das sind beispielsweise der Suchtext, die E-Mail-Adresse des Benutzers, der Datei- oder Lizenzpfad, wie in Abbildung B.2 im Anhang B abgebildet. Danach wird das JXTA-Netzwerk initialisiert, in dem unter anderem die NetPeerGroup und die CupPeerGroup instanziiert werden.

*Server initialisieren*
Wie beim Client müssen beim Server ebenfalls die erforderlichen Informationen wie Typ des Servers aus dem ConfigurationManager des TURM-Systems ausgelesen und das JXTA-Netzwerk initialisiert werden. Anschließend werden die Beschreibungen der Dienste SearchService und DownloadService in der CupPeerGroup veröffentlicht und der Server beginnt damit diese zu überwachen. Siehe dazu auch das Aktivitätsdiagramm in Abbildung B.3 auf Seite 122.

*Suchanfrage (des Benutzers) an Peers senden*
Bevor die nächste Aktion des Servers beschrieben wird, muss zunächst der als Client handelnde CUP das Advertisement des SearchServices in der

CupPeerGroup suchen und bei Erfolg das gefundene Advertisement herunterladen. Anhand der Beschreibung wird eine SearchService-Anfrage mit dem Suchtext erstellt und an den SearchService gesendet. Von den Peers die den SearchService überwachen, bekommt der Client nun bei einer erfolgreichen Suche Ergebnislisten mit entsprechenden Dateinamen zur Auswahl zurück (siehe dazu in Anhang B sowohl das Aktivitätsdiagramm in Abbildung B.4 als auch das Sequenzdiagramm in Abbildung B.6).

*Suchanfrage (von außen) bearbeiten*
In diesem Schritt bearbeitet der CUP in der Funktion als Server die Suchanfrage eines anderen Peers, indem die in Abbildung B.6 visualisierten Sequenzen ausgeführt werden und bei erfolgreicher Suche dem Client eine Ergebnisliste zurückgeliefert wird (siehe auch das Aktivitätsdiagramm in Abbildung B.5). Dabei empfängt der CUP-Server zunächst eine Anfrage des SearchServices und sucht alle Dateien, die dem mitgesendeten Suchtext entsprechen. Danach werden die Dateien, die laut Lizenz kein Recht auf Weitergabe besitzen herausgefiltert, und wenn es sich um keinen Freund handelt, werden zusätzlich die Dateien, die nur begrenzt weitergegeben werden dürfen, entfernt. Sind nun noch passende Dateien vorhanden, wird dem angefragten CUP-Client die entsprechende Ergebnisliste gesendet.

*Download einer Datei mit Lizenz*
In dieser Aktion wird, wie in Abbildung B.7 dargestellt, zunächst das Advertisement des DownloadServices in der CupPeerGroup gesucht und bei Erfolg das gesuchte Advertisement heruntergeladen. Daraufhin wird eine DownloadService-Anfrage mit dem Hashwert der herunterzuladenden Datei erstellt und an den DownloadService gesendet. Wenn eine Antwort zustande kommt, empfängt der CUP-Client die benötigten Download-Informationen, womit die angeforderte Datei (inklusive der zugehörigen Exportlizenz) über den in der CupPeerGroup implementierten Content-Service heruntergeladen werden kann. Die geladenen Dateien werden in

dem konfigurierten Datei- und Lizenzpfad abgespeichert. In dem Sequenzdiagramm in Abbildung B.9 auf Seite 128 lässt sich erkennen, welche Ereignisse zum Download bzw. Upload ausgeführt werden müssen.

*Upload einer Datei mit Lizenz*
Der von der Serverfunktion eines CUPs ausgeführte Upload einer Datei (inklusive der zugehörigen Exportlizenz) wird in Abbildung B.8 als Aktivitätsdiagramm gezeigt und im bereits beim Download angesprochenen Sequenzdiagramm in Abbildung B.9 genauer spezifiziert. Zunächst wird eine DownloadService-Anfrage empfangen, wodurch anhand des übermittelten Hashwerts dargelegt wird, welche Datei übertragen werden soll. Anhand dieser Datei-ID wird die entsprechende Datei herausgesucht und zur Sicherheit „erneut“ überprüft, ob der Weitergabe- und Personenstatus gewährleistet ist. Ist dies der Fall wird zu dieser Datei eine Exportlizenz erzeugt und beide Dateien über den ContentService zur Verfügung gestellt. Damit nun der CUP-Client diese Dateien herunterladen kann, bekommt dieser die entsprechenden Download-Informationen übermittelt.

Nachdem die Einzelaktionen des CUP-Programmablaufs verdeutlicht wurden, wird im Weiteren auf die verschiedenen Klassen und deren Beziehungen untereinander eingegangen.

### 4.6.2 Aufbau des CUP-Klassendiagramms

Im folgenden Teil der Arbeit werden programmierkonforme Bezeichner wie z.B. Klassen- oder Interfacenamen verwendet, die im Quellcode des CUPs eingesetzt werden. Damit der Überblick bewahrt bleibt, werden solche Bezeichner im Weiteren beispielsweise wie folgt notiert: `Cup`.

Wie in Abbildung 4.7 zu sehen, fungiert die `Cup`-Klasse als Hauptklasse des CUP-Systems. Die `Cup`-Klasse verwaltet als Kern die Objekte des CUP-Systems und stellt die Verbindung zum TURM-System über die `Turm`-Klasse her. In Verbindung mit dem TURM sind in dem abgebildeten Klassendiagramm nur die Klassen dargestellt, die vom CUP benötigt

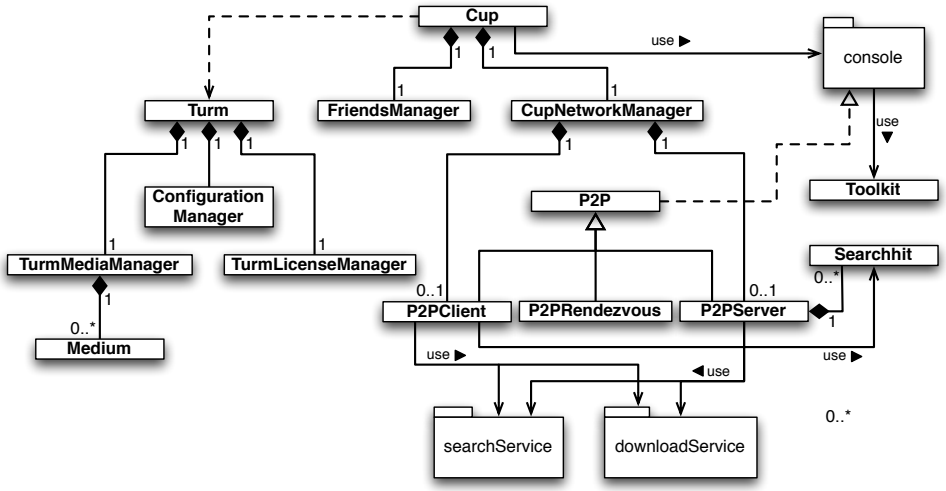


Abbildung 4.7: Klassendiagramm des CUPs

werden. Dazu gehört die Klasse `ConfigurationManager`, die dem gesamten TURM als Datenbasis für allgemeine und individuelle Konfigurationen dient. Dort speichert der CUP die verschiedenen Einstellungen, die zu Beginn über die Kommandozeile übergeben werden, wie beispielsweise Benutzername, E-Mail-Adresse des Benutzers oder Dateipfad. Die Verwaltung der Mediendateien wird von der Klasse `TurmMediaManager` übernommen, die eine entsprechende Suchfunktion für Dateien anhand eines Suchtextes oder eines Hashwerts anbietet. Von den gefundenen Dateien werden `Medium`-Objekte, mit Informationen wie Name, Pfad und Hashwert der Datei erstellt. Die `TurmLicenseManager`-Klasse verwaltet analog zur Klasse `TurmMediaManager` die Lizenzen der Dateien. Diese wird vom CUP verwendet, um herauszufinden welchen Weitergabestatus eine bestimmte Datei hat und eine Exportlizenz erzeugen zu lassen.
Die CUP-Klasse `FriendsManager` verwaltet die Liste, in der die mit besseren Konditionen ausgestatteten Freunde gespeichert sind. Diese Klasse erlaubt es, Freunde anhand deren E-Mail-Adresse der Freundesliste hinzuzufügen oder zu entfernen. Die meisten Aktionen des CUPs finden in der Klasse `CupNetworkManager` und den assoziierten Klassen statt. Die `CupNetworkManager`-Klasse startet entsprechend den Kommadozeilen-Eingaben den Client oder den Server. Für den Client wird ein Objekt der

Klasse `P2PClient` erzeugt und für den Server analog ein Objekt der Klasse `P2PServer`. Beide Klassen sind Spezialisierungen der `P2P`-Klasse und verwenden die Klasse `Searchhit` um zu den Angaben eines `Medium`-Objekts noch den Weitergabestatus zu jedem Medium abspeichern zu können. Die Klasse `P2PRendezvous` steht für einen Rendezvous-Server, der im späteren Verlauf des CUP-Projekts für die Kommunikation im Internet eingesetzt wird. Die Packages `searchService` und `downloadService` beinhalten mehrere Klassen und Interfaces, die der Übersicht halber in diesem Klassendiagramm nicht einzeln dargestellt sind, jedoch in Kapitel 5 noch näher erläutert werden. Sowohl das Package `searchService` als auch das Package `downloadService` werden von den beiden Klassen `P2PClient` und `P2PServer` verwendet, um die Kommunikation über das Netzwerk zu ermöglichen. Ein weiteres Paket ist das `console`-Package, welches ebenfalls aus mehreren Klassen besteht und die Konsolensteuerung beim Starten des CUPs und während der Netzwerkkommunikation übernimmt. Die Klasse `Toolkit` stellt allgemeine Funktionen wie z.B. das Überprüfen einer IP-Adresse auf Validität zur Verfügung.

Auf Grundlage der UML-Diagramme (siehe dieses Kapitel, Kapitel 4.6.1 und Anhang B ab Seite 119) wird das Feindesign der CUP-Komponente so spezifiziert, dass der Code nachvollziehbar ist und damit verständlich wird. So ist es möglich, dass der CUP-Code in der Implementierungsphase aus den hier beschriebenen Modellen generiert werden kann. Alle Diagramme zusammengenommen legen damit das Gesamtverhalten des CUP-Systems fest.

# 5 Implementierung

In diesem Kapitel wird darauf eingegangen, wie die im vorherigen Kapitel beschriebenen Konzepte in die Praxis umgesetzt werden. Die im Wasserfallmodell in Abbildung 1.2 gezeigten Phasen der Implementierung, Testphase und Fertigstellung werden berücksichtigt. Gleichzeitig werden die Tätigkeiten des in Abbildung 2.6 gezeigten Softwarelebenszyklus vom Implementieren und Integrieren bis zum Betreiben und Warten einbezogen.

Die Implementierung wird damit begonnen, ein Java-Projekt mit den in Abbildung 4.7 skizzierten Klassen anzulegen. Anschließend werden die Aktionen des in Abbildung 4.4 visualisierten Aktivitätsdiagramms der UML in den entsprechenden Klassen eingebunden. Dazu gehört beispielsweise das Einrichten und Überprüfen der Kommandozeilen-Eingaben, das Abspeichern oder Entfernen von Freunden, das Verwenden von Funktionalitäten des TURM-Systems oder das Implementieren des P2P-Netzwerks. Um die Übersicht zu bewahren, wird in diesem Kapitel nicht der vollständige Programmcode des CUP-Systems beschrieben, sondern wichtige Teilstücke herausgegriffen und erläutert. Der Autor hat für dieses Kapitel versucht einen Weg zu finden, einerseits technisch nicht zu detailliert zu werden, andererseits ausreichende Informationen für zukünftige Entwickler des CUPs zur Verfügung zu stellen. Aus diesem Grund bleibt es für das Kapitel der Implementierung nicht aus, einen gewissen technischen Grad beizubehalten, um wichtige programmiertechnische Zusammenhänge vermitteln zu können. Des Weiteren wird das bereits in Kapitel 4.6 eingeführte Textlayout, wie z.B. `Cup`, für Bezeichner des CUP-Quellcodes der Übersicht halber verwendet.

## 5.1 Allgemeine Programmlogik

Ein Ausschnitt der allgemeineren CUP-Klassen mit den wichtigsten Methoden sind im Anhang C.1 ab Seite 130 aufgeführt. Das im Klassendiagramm auf Seite 81 dargestellte `console`-Package stellt Funktionen für die Steuerung der Kommandozeile bereit. Sowohl die Eingaben beim Starten des CUPs (siehe Abbildung 4.6 auf Seite 77) als auch das während der P2P-Kommunikation angebotene Menü wird von den Klassen und dem Interface des `console`-Packages bereitgestellt. Einen weiteren Einblick gibt das in Kapitel 5.3.1 beschriebene Interaktionsszenario.
Als Hauptklasse des CUP-Systems besitzt die Klasse `Cup` die Methoden `getConfigurationManager()`, `getTurmMediaManager()` und `getTurmLicenseManager()` und stellt auf diese Weise die Schnittstelle zum TURM-System her. Alle anderen CUP-Klassen, die Funktionen des TURMS benötigen, verwenden diese Methoden, um auf entsprechende TURM-Komponenten wie z.B. das Objekt des `ConfigurationManagers` zuzugreifen. Des Weiteren werden durch die Methode `saveArguments(args)` die über die Kommandozeile eingegebenen Werte im `ConfigurationManager` gespeichert. Mit der Methode `updateFriendsList(args)` wird die Freundesliste entsprechend der Eingaben verwaltet.
Von dem `Cup`-Objekt wird ein Objekt der Klasse `CupNetworkManager` erzeugt. Dieses startet mit der Methode `startNetworkManager()` den gewählten P2P-Typ Client (siehe `P2PClient`-Klasse) oder Server (siehe `P2PServer`-Klasse).
Die Klassen `P2PClient` und `P2PServer` halten Funktionen bereit, die zur Kommunikation über das Netzwerk benötigt werden. Das Objekt des `P2P-Clients` stellt beispielsweise den Suchtext und die E-Mail Adresse des Benutzers zur Verfügung, während das Objekt des `P2PServers` unter Verwendung des Suchtextes eine Ergebnisliste erstellt. Beide Klassen stellen die Schnittstellen zum P2P-Netzwerk her und werden in Kapitel 5.3.3 genauer betrachtet.

## 5.2 Schnittstellen zum TURM-System

Die Schnittstellen des CUPs zum TURM-System stellen einen wichtigen Teil der Implementierung dar. Die Funktionalitäten des URM-Konzepts, wie Medien- und Lizendateien zu verwalten, werden vom TURM-System umgesetzt und können somit direkt vom CUP verwendet werden. Durch dieses Vorgehen werden Redundanzen vermieden und eine Effizienzsteigerung erreicht. Im Folgenden werden die verschiedenen Methoden der verwendeten TURM-Klassen anhand der Tabelle 5.1 betrachtet.

| Klasse | Methoden |
|---|---|
| Turm | getConfigurationManager() : ConfigurationManager<br>getTurmMediaManager() : TurmMediaManager<br>getTurmLicenseManager() : TurmLicenseManager |
| ConfigurationManager | setConfigurationValue(key, value) : void<br>getConfigurationValue(key) : value |
| TurmMediaManager | searchMedium(string) : ArrayList<Medium><br>getMedium(hashvalue) : Medium |
| Medium | getName() : String, setName() : void<br>getFiledir() : String, setFiledir() : void<br>getHashvalue() : String, setHashvalue() : void |
| TurmLicenseManager | getPropagationState(hashvalue) : int<br>getExportLicense(hashvalue, email) : File |

Tabelle 5.1: CUP-Schnittstellen zum TURM-System

Die Einbindung der im Klassendiagramm auf Seite 81 vorgestellten TURM-Klassen in das CUP-System erfolgt über den Kern des TURMs, die `Turm`-Klasse.
Diese bietet die Funktionen `getConfigurationManager()`, `getTurmMediaManager()` und `getTurmLicenseManager()`, womit

das jeweilige Objekt der entsprechenden Klasse zurückgegeben wird (siehe Beispielaufruf in Listing 5.1).

Listing 5.1: Beispielaufrufe einer Methode der Klasse Turm

```
1 Turm turm = new Turm();
2 ConfigurationManager cm;
3 cm = turm.getConfigurationManager();
```

Die Klasse `ConfigurationManager` verwaltet verschiedene Einstellungen der TURM-Komponenten. Anhand der Methode `setConfigurationValue(key, value)` kann wie im Listing 5.2 gezeigt ein Schlüssel/Werte-Paar gesetzt werden. Die im Objekt des `ConfigurationManagers` abgespeicherten Werte können wiederum anhand der Methode `getConfigurationValue(key)` mit Angabe des jeweiligen Schlüssels zurückgegeben werden (siehe Listing 5.2, Zeile 8-11).

Listing 5.2: Beispielaufrufe der Klasse ConfigurationManager

```
1  Turm turm = new Turm();
2  ConfigurationManager cm;
3  cm = turm.getConfigurationManager();
4
5  cm.setConfigurationValue("core:licensePath", "/Users/
       vliesen/licenses/");
6  cm.setConfigurationValue("cup:user", "vliesen");
7
8  String licensePath;
9  licensePath = cm.getConfigurationValue("core:
       licensePath");
10 String user;
11 user = cm.getConfigurationValue("cup:user");
```

Die `TurmMediaManager`-Klasse wird erst nach Fertigstellung dieser Diplomarbeit durch eine andere Qualifikationsarbeit vollständig entwickelt. Aus diesem Grund wurden die für den CUP benötigten Methoden

`searchMedium(string)` und `getMedium(hashvalue)` vorläufig vom Autor prototypisch implementiert.

Listing 5.3: Beispielaufrufe der Klasse TurmMediaManager

```
1  Turm turm = new Turm();
2  TurmMediaManager mm;
3 mm = turm.getTurmMediaManager();
4  String hashvalue;
5  hashvalue = "06be65baebd4f13f3dc8313ad5b4f1f5f3938d";
6
7  ArrayList<Medium> mL = new ArrayList<Medium>();
8 mL = mm.searchMedium("Bryan Adams");
9
10 Medium m = new Medium ();
11 m = mm.getMedium(hashvalue);
```

Wie Listing 5.3 zeigt, leitet der CUP die Suche nach Dateien weiter, indem dieser mit Angabe des Suchtextes die Methode `searchMedium(string)` des `TurmMediaManagers` aufruft. Siehe dazu auch die Sequenzdiagramme auf Seite 125 und 128. Daraufhin gibt das Objekt des `TurmMediaManagers` dem CUP eine Liste mit `Medium`-Objekten zurück (siehe Zeile 7 u. 8). Die Methode `getMedium(hashvalue)` wird dazu verwendet, anhand des Hashwerts der Datei-Payload das `Medium`-Objekt mit den zugehörigen Dateiinformationen zu bekommen.

Um verschiedene Informationen zu einer Datei sammeln zu können gibt es die `Medium`-Klasse. Diese ist ein Teil der Klasse `TurmMediaManager` und verwaltet in den Objekten die Dateiinformationen: Name, Verzeichnis und Hashwert. Diese Informationen sind vorläufig und werden in der späteren Version der Klasse `TurmMediaManager` erweitert. Durch getter- und setter-Methoden können die einzelnen Werte gesetzt oder abgefragt werden.

Mit Hilfe der Klasse `TurmLicenseManager` ist es dem CUP möglich verschiedene Lizenzaktionen durchzuführen, wie in Listing 5.4 und in den Sequenzdiagrammen auf Seite 125 und 128 dargestellt. Mit der

Methode `getPropagationState(hashvalue)` kann der CUP erfragen, welchen Weitergabestatus (z.B. 0 = keine, 1 = begrenzt, 2 = unbegrenzt) die angegebene Datei besitzt. Weiterhin kann durch die Methode `getExportLicense(hashvalue, email)` das Erzeugen einer Exportlizenz angestoßen werden und die erstellte Lizenzdatei wird als sogenanntes `File`-Objekt[1] zurückgegeben. Der im Listing 5.4 dargestellte Programmcode wird nach Fertigstellung des `TurmLicenseManagers` im CUP implementiert.

Listing 5.4: Beispielaufrufe der Klasse TurmLicenseManager

```
1  Turm turm = new Turm();
2  TurmLicenseManager lm;
3  lm = turm.getTurmLicenseManager();
4  String hashvalue;
5  hashvalue = "06be65baebd4f13f3dc8313ad5b4f1f5f3938d";
6
7  int propState;
8  propState = lm.getPropagationState(hashvalue);
9
10 File licenseFile = new File ();
11 licenseFile = lm.getExportLicense(hashvalue, "
     empf@enger.de");
```

## 5.3 P2P-Netzwerk

Die P2P-Funktionalität des CUPs wird in diesem Abschnitt noch etwas detaillierter beschrieben, da diese einen Großteil der Entwicklung des CUP-Systems in Anspruch nimmt. Ein wichtiger Aspekt sind in diesem Zusammenhang die JXTA-Services. Wie bereits in Kapitel 2.2.3 beschrieben sind einige solcher Dienste, die schwerpunktmäßig die Netzwerkkommunikation unter den Peers regeln, bereits Bestandteil des JXTA-Frameworks

[1] Java-Klasse `File`; URL: http://java.sun.com/j2se/1.4.2/docs/api/java/io/File.html; Stand: 05.05.2010

bzw. dessen Referenzimplementierung für Java. Darüber hinaus wurden für das CUP-System zwei Services mit den entsprechenden Kommunikationsprotokollen entwickelt, die die speziellen Anforderungen der CUP-Netzwerkübertragung erfüllen. Dazu gehören die im P2P-Konzept in Abbildung 4.3 dargestellten Services *SearchService* und *DownloadService*. Diese werden in folgendem Kapitel erläutert, bevor in Kapitel 5.3.2 auf die Kommunikationsprotokolle der Services eingegangen wird. Abschließend wird in Kapitel 5.3.3 ein Szenario der verschiedenen Interaktionen beschrieben.

### 5.3.1 CUP-Services

Aufbauend auf dem CUP-Klassendiagramm in Abbildung 4.7 auf Seite 81 werden im Weiteren die dort abgebildeten Packages `searchService` und `downloadService` genauer betrachtet. In diesen Paketen befinden sich die Klassen und Interfaces, die die CUP-spezifischen Netzwerkfunktionen umsetzen und in Abbildung 5.1 entsprechend skizziert sind.

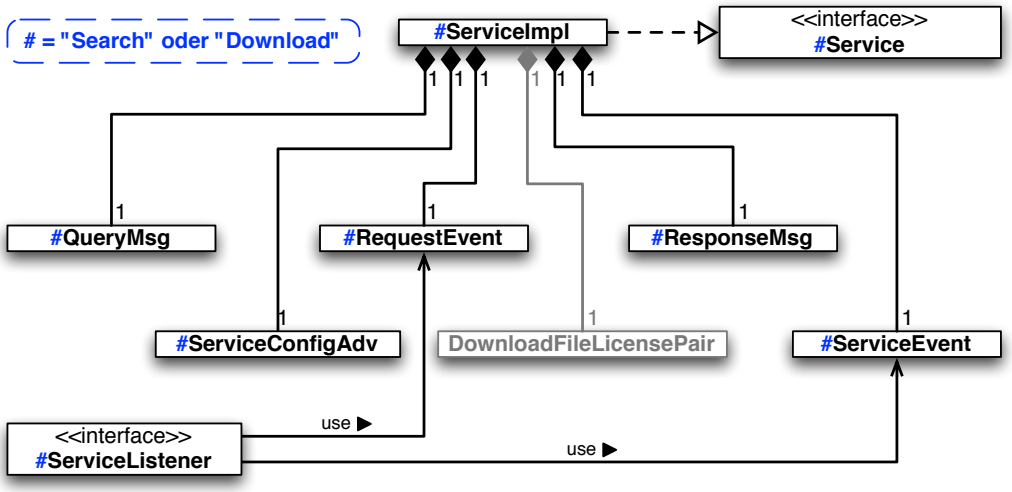


Abbildung 5.1: Klassendiagramm der Service-Packages

Ausgehend von dem CUP-Klassendiagramm werden die CUP-Services sowohl von der `P2PClient`-Klasse als auch von der `P2PServer`-Klasse

verwendet. Das Klassendiagramm der Service-Packages des CUPs zeigt, dass die beiden Services eine fast äquivalente Struktur aufweisen. Lediglich der DownloadService besitzt eine zusätzliche Klasse `DownloadFileLicensePair`, die die Medium- und Lizenzdatei als Typ-Objekt abbildet und als Rückgabewert verwendet werden kann.
Die Interfaces `Search`- und `DownloadServiceListener` werden von den Klassen `P2PClient` und `P2PServer` implementiert. Die implementierten Listener-Methoden erhalten über die jeweiligen Services die Objekte der `RequestEvent`- und `ServiceEvent`-Klassen als Parameter. Somit kann die Instanz eines Services Mitteilungen an den jeweiligen Listener versenden, der dann auf die eingehenden Ereignisse reagieren kann (z.B. eine Suche durchführen).
Die Hauptfunktionen in der Serviceimplementierung sind in der Klasse `SearchServiceImpl` bzw. `DownloadServiceImpl` realisiert. Hier laufen alle Aktionen des entsprechenden Services zusammen und gleichzeitig werden an dieser Stelle die Klassen `P2PClient` und `P2PServer` bedient. Um dies zu leisten werden die weiteren Klassen und Interfaces der Services benötigt, welche im Folgenden erläutert werden.

*`Service`-Interfaces*
Die Interfaces `SearchService` und `DownloadService` werden von der entsprechenden `ServiceImpl`-Klasse implementiert. Diese stellen die Methoden `add()` und `remove()` zur Verwaltung der `ServiceListener` (`P2PClient` und `P2PServer`) zur Verfügung und eine Servicespezifische Methode `search(searchString, email)` bzw. `download(dlHash, email)`.

*`ServiceConfigAdv`-Klassen*
Die `ServiceConfigAdv`-Klassen werden für die Konfiguration der Advertisements während der Initialisierung des SearchServices und des DownloadServices benötigt. Anhand eines `ServiceConfigAdv`-Objekts kann beispielsweise geprüft werden, ob bereits ein Peer ein Advertisement des entsprechenden Services in der `CupPeerGroup` anbietet.

*QueryMsg-Klassen*
Die QueryMsg-Klassen verpacken die zu übertragenden Informationen, wie z.B. Suchtext oder Hashwert zu einer Anfrage (Client). Diese Anfrage wird mit Hilfe des JXTA-Frameworks über das P2P-Netzwerk weitergeleitet. Siehe dazu auch Abbildung 5.5 auf Seite 102.

*RequestEvent-Klassen*
Ein Objekt der Klasse SearchRequestEvent oder der Klasse DownloadRequestEvent signalisiert die Ankunft einer neuen Anfragenachricht (QueryMsg -> Server) am Service.

*ResponseMsg-Klassen*
Die Klassen SearchResponseMsg und DownloadResponseMsg erzeugen zu einer Anfrage eine Antwortnachricht (Server). Eine solche Antwort enthält beispielsweise beim SearchService eine Ergebnisliste oder beim DownloadService Dateiinformationen der Medien- und Lizenzdatei. Siehe dazu auch das P2P-Konzept auf Seite 70.

*ServiceEvent-Klassen*
Ein Objekt der Klasse Search- oder DownloadServiceEvent signalisiert die Ankunft einer neuen Antwortnachricht (ResponseMsg -> Client) am Service.

Die verwendeten Service-Klassen für den CUP mit den wichtigsten Methoden sind im Anhang C.2 ab Seite 131 aufgeführt.

### 5.3.2 CUP-Protokolle

Für die Verwendung des SearchServices und des DownloadServices ist ein Protokoll notwendig, das die Kommunikation zwischen den Nutzern der Services, den Client- und Server-Peers, regelt. Der Aufbau und die Art der zwischen den Peers auszutauschenden Nachrichten wird von diesen Kommunikationsprotokollen beschrieben und festgelegt. Die Protokolle wurden speziell für die Anforderungen der beiden Services angepasst und entwickelt und werden im Folgenden spezifiziert.

**SearchService-Protokoll**

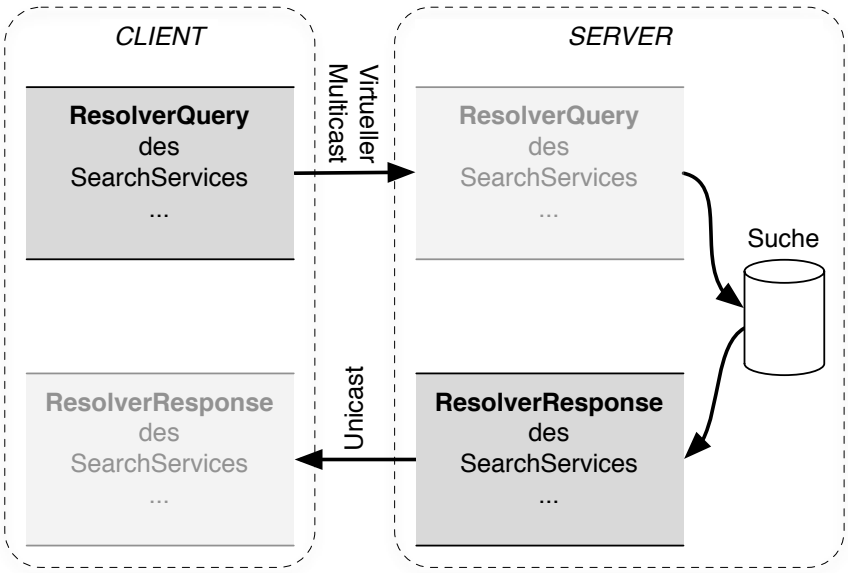


Abbildung 5.2: Protokoll des SearchServices

Das Protokoll des SearchServices ist in Abbildung 5.2 dargestellt. Es ist so aufgebaut, dass von dem als Client handelnden CUP eine `ResolverQuery` an die lokale Instanz des SearchServices gesendet wird. Die Instanz des SearchServices leitet die `ResolverQuery` an alle Peers der `CupPeerGroup`, die den SearchService instanziiert haben, weiter (*virtueller Multicast*). Das Verteilen einer `ResolverQuery` ist ein Dienst des `ResolverServices` der von JXTA bereitgestellt wird (siehe Kapitel 2.2.3). Die `ResolverQuery` beinhaltet z.B. den Suchtext und die E-Mail Adresse des Benutzers (siehe Listing 5.5). Die gesendete `ResolverQuery` wird von den lauschenden Servern empfangen, die daraufhin eine Suche durchführen. Sind Suchtreffer vorhanden wird je eine `ResolverResponse` erzeugt (siehe Listing 5.6) und an den Client zurück gesendet (*Unicast*). Das Verteilen einer `ResolverResponse` ist ebenfalls ein Dienst des `ResolverServices` der von JXTA bereitgestellt wird.

Nachdem die Semantik des SearchService-Protokolls festgelegt ist, wird im Folgenden anhand von Listings die Syntax des Protokolls

herausgestellt. Die Nachrichten `ResolverQuery` und `ResolverResponse` sind in einem XML-Format spezifiziert, welches in JXTA als Übertragungsformat festgelegt ist. Der Aufbau der äußeren Struktur ist bei allen JXTA-Nachrichten identisch. In der ersten Zeile wird durch eine XML-Deklaration die XML-Version mit *1.0* und die Zeichenkodierung mit *UTF-8* angegeben. Anschließend wird der Dokumenttyp mit dem Namen *jxta:ResolverQuery* bzw. *jxta:ResolverResponse* deklariert. In der dritten Zeile wird mit dem Hauptelement begonnen, welches mit *jxta:ResolverQuery* bzw. *jxta:ResolverResponse* benannt ist und das Präfix *jxta* an den JXTA-Namensraum *http://jxta.org* bindet. Im weiteren werden mehrere allgemeine und spezifische Elemente von dem Hauptelement eingeschlossen. Die Elemente *HandlerName* und *QueryID* sind in beiden Nachrichtenformaten enthalten. Der HandlerName beinhaltet einen eindeutigen Namen des JXTA-ResolverServices auf dem Clientpeer. Die QueryID stellt eine eindeutige Anfragenummer zur Identifizierung der Anfrage dar. Weiterhin existiert jeweils ein *Query*- oder *Response*-Element, welches die Anfrage oder Antwort enthält, die an die anderen Peers gesendet wird.

Die clientseitig erzeugbare `ResolverQuery` des Listings 5.5 auf Seite 95 beinhaltet als `ResolverQuery`-Typ die zwei zusätzlichen JXTA-Elemente *HC* und *SrcPeerID*. HC ist die Abkürzung für „Hop Count“ und legt die Anzahl der besuchten Knoten fest, während SrcPeerID die eindeutige Nummer des Servers ist. Die speziell für den CUP entwickelte Syntax wird in dem *Query*-Element verwendet. Dieses enthält die für die spätere Verarbeitung wichtigen Informationen in Form einer XML-Datei. Die Deklaration *p2p:SearchQuery* ist gleichzeitig die Bezeichnung des Hauptelements. Dadurch ist der Bezug zum `SearchService` hergestellt. Die beiden Unterelemente enthalten den Suchtext und die E-Mail Adresse des Client-Benutzers.

Die im Listing 5.6 auf Seite 96 dargestellte `ResolverResponse` kann von einem CUP-Server erzeugt werden und enthält zusätzlich zur XML-Struktur von JXTA keine weiteren JXTA-Elemente. In dem Element *Response* wird eine für den CUP spezifizierte XML-Syntax mit den für die

Weiterverarbeitung notwendigen Informationen verwendet. Der Aufbau ist ähnlich dem im Query-Element der `ResolverQuery`. Die Bezeichnung *p2p:SearchResponse* gibt den Dokumenttyp an und bezeichnet das Hauptelement. Als Informationen sind der Suchtext und die Ergebnisliste (<*answer* />) angegeben. Diese Ergebnisliste liegt in Form einer serialisierten `ArrayList`[2] vor. Das bedeutet, dass das `ArrayList`-Objekt auf die Darstellungsform `String` abgebildet wurde, der die verwendeten Packages und die Suchtreffer mit Dateinamen und Hashwert enthält.

[2]Java-Klasse `ArrayList`; URL: http://java.sun.com/j2se/1.4.2/docs/api/java/util/ArrayList.html; Stand: 12.05.2010

Listing 5.5: ResolverQuery des SearchServices

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:ResolverQuery>
<jxta:ResolverQuery xmlns:jxta="http://jxta.org">
    <HandlerName>
        urn:jxta:uuid-4CD1574ABA614A5FA242B613D8BAA30F05
    </HandlerName>
    <QueryID>
        7
    </QueryID>
    <HC>
        0
    </HC>
    <SrcPeerID>
        urn:jxta:uuid-59616261646162614E504720503250337
            D9D24A3A2F1439AA0C7DAC52ED64B8C03
    </SrcPeerID>
    <Query>
        <?xml version="1.0"?>
        <!DOCTYPE p2p:SearchQuery>
        <p2p:SearchQuery>
            <searchString>
                Bryan Adams
            </searchString>
            <email>
                vliesen@uni-koblenz.de
            </email>
        </p2p:SearchQuery>
    </Query>
</jxta:ResolverQuery>
```

Listing 5.6: ResolverResponse des SearchServices

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:ResolverResponse>
<jxta:ResolverResponse xmlns:jxta="http://jxta.org">
    <HandlerName>
        urn:jxta:uuid-4CD1574ABA614A5FA242B613D8BAA30F05
    </HandlerName>
    <QueryID>
        7
    </QueryID>
    <Response>
        <?xml version="1.0"?>
        <!DOCTYPE p2p:SearchResponse>
        <p2p:SearchResponse>
            <searchString>
                Bryan Adams
            </searchString>
            <answer>
                ------------------------------------------------
                (Zum Zweck der Lesbarkeit wird hier statt
                 des Bytestreams, die entsprechenden In-
                 formationen textuell dargestellt.)

                SERIALISIERTE ArrayList<Searchhit>:
                Packages e.g.
                    java.util.ArrayList
                    de.uni_koblenz.aggrimm.turm.cup.
                        Searchhit
                Files, e.g.
                    Feist_1234.mp3,
                    d6966e5fe2e14d954cb35b51ac96cdbc87b09eq
                    Sunrise-Avenue_Fairytale-Gone-Bad.mp3,
                    e84327b6ff764f0d982bfbceba1fd679cc733f7
                ------------------------------------------------
            </answer>
        </p2p:SearchResponse>
    </Response>
</jxta:ResolverResponse>
```

**DownloadService-Protokoll**

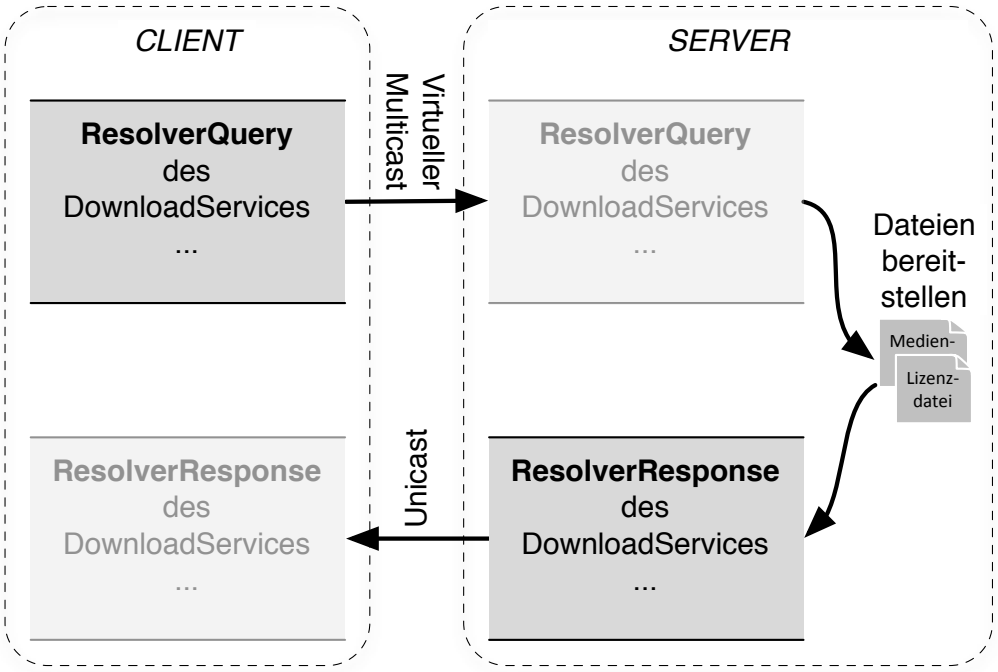


Abbildung 5.3: Protokoll des DownloadServices

Das DownloadService-Protokoll ist ähnlich dem Protokoll des Search-Services aufgebaut und in Abbildung 5.3 veranschaulicht. Die Clientseite übermittelt eine `ResolverQuery` an die lokale Instanz des Download-Services. Die Instanz des DownloadServices leitet die `ResolverQuery` an alle Peers der `CupPeerGroup`, die den DownloadService instanziiert haben, weiter (*virtueller Multicast*). Die `ResolverQuery` beinhaltet z.B. den Hashwert der gewählten Datei und die E-Mail Adresse des Benutzers (siehe Listing 5.7). Die gesendete `ResolverQuery` wird von dem Server empfangen, der daraufhin die entsprechende Medien- und Lizenzdatei bereitstellt. Eine `ResolverResponse` wird erzeugt (siehe Listing 5.8) und an den Client zurück gesendet (*Unicast*).

Die Syntax des Downloadprotokolls ist genauso aufgebaut wie beim Protokoll des SearchServices. Sowohl die `ResolverQuery` als auch die `ResolverResponse` des DownloadServices sind JXTA-Nachrichten und

bauen dementsprechend auf der im vorherigen Abschnitt beschriebenen Grundstruktur auf.

Die `ResolverQuery` des DownloadServices (siehe Listing 5.7 auf Seite 95) umfasst dem Typ entsprechend die zusätzlichen JXTA-Elemente *HC* und *SrcPeerID*. Die speziell für den CUP entwickelte Syntax wird auch hier in dem *Query*-Element verwendet, deren Inhalt als XML-Datei dargestellt ist. Das Hauptelement und die Deklaration ist mit *p2p:DownloadQuery* beschrieben, wodurch auch hier deutlich wird, dass dies eine Nachricht des DownloadServices ist. Die beiden Unterelemente enthalten den Hashwert der gewählten Datei und die E-Mail Adresse des Client-Benutzers.

Die im Listing 5.8 auf Seite 100 abgebildete `ResolverResponse` kann von einem CUP-Server erzeugt werden und enthält zusätzlich zur JXTA-XML-Struktur keine weiteren JXTA-Elemente. In dem Element *Response* wird eine für den CUP spezifizierte XML-Syntax mit den für die Weiterverarbeitung notwendigen Informationen verwendet. Der Aufbau ist ähnlich dem im *Query*-Element der `ResolverQuery`. Die Bezeichnung *p2p:DownloadResponse* gibt den Dokumenttyp an und bezeichnet das Hauptelement. Als Daten sind der Datei-Hashwert, die Download-Informationen der Medien- und Lizenzdatei und der Dateiname des Mediums aufgeführt. Die Download-Informationen sind von JXTA generierte Werte, die im JXTA-Netzwerk eindeutig sind.

Listing 5.7: ResolverQuery des DownloadServices

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:ResolverQuery>
<jxta:ResolverQuery xmlns:jxta="http://jxta.org">
    <HandlerName>
        urn:jxta:uuid-A24E77A0C2414EC28ADFE5D34CF8BCBD05
    </HandlerName>
    <QueryID>
        4
    </QueryID>
    <HC>
        0
    </HC>
    <SrcPeerID>
        urn:jxta:uuid-59616261646162614E504720503250337
            D9D24A3A2F1439AA0C7DAC52ED64B8C03
    </SrcPeerID>
    <Query>
        <?xml version="1.0"?>
        <!DOCTYPE p2p:DownloadQuery>
        <p2p:DownloadQuery>
            <dlHash>
                06be65baebd4f13f3dc8313ad5b4f1f5f3938d
            </dlHash>
            <email>
                vliesen@uni-koblenz.de
            </email>
        </p2p:DownloadQuery>
    </Query>
</jxta:ResolverQuery>
```

Listing 5.8: ResolverResponse des DownloadServices

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jxta:ResolverResponse>
<jxta:ResolverResponse xmlns:jxta="http://jxta.org">
    <HandlerName>
        urn:jxta:uuid-A24E77A0C2414EC28ADFE5D34CF8BCBD05
    </HandlerName>
    <QueryID>
        4
    </QueryID>
    <Response>
        <?xml version="1.0"?>
        <!DOCTYPE p2p:SearchResponse>
        <p2p:SearchResponse>
            <dlHash>
                06be65baebd4f13f3dc8313ad5b4f1f5f3938d
            </dlHash>
            <answerFile>
                urn:jxta:uuid-67
                    F39F9FF4C64FFB954AED6246ADB1CED2BCF
                    0609D2B4674AA24BC5FDD2294D207
            </answerFile>
            <answerFileName>
                Aladdin_A-Whole-New-World.mp3
            </answerFileName>
            <answerLicense>
                urn:jxta:uuid-67
                    F39F9FF4C64FFB954AED6246ADB1CE663D0036
                    E1F14872B4CBAFBC44897B0307
            </answerLicense>
        </p2p:SearchResponse>
    </Response>
</jxta:ResolverResponse>
```

### 5.3.3 Interaktionsszenario

Das folgende Szenario gibt einen Überblick über die Reihenfolge wichtiger CUP-interner Aufrufe. Dabei werden die beiden CUP-Services fokussiert und der Bereich vom Start der Suche über die Kommandozeile bis hin zur fertig heruntergeladenen Mediendatei mit Lizenz dargestellt. Die Funktionsaufrufe mit entsprechenden Beschreibungen sind in den Abbildungen 5.6, 5.7, 5.8 und 5.9 ab Seite 104 skizziert. Diese Abbildungen basieren auf keiner expliziten Notation wie z.B. UML, sondern wurden vom Autor so strukturiert, dass ein guter Überblick über die technischen Inhalte entsteht.

Ist der CUP gestartet, wird auf der Kommandozeile eine Liste der gefunden Peers (`Requester` -> Client bzw. `Resolver` -> Server) angezeigt und mit der `ENTER`-Taste kann ein Menü aufgerufen werden. Das Standardmenü ist in Abbildung 5.4 rechts beim Server dargestellt und bietet beispielsweise eine Funktion `quit` (Aufruf durch `q` + `ENTER`), mit der der gestartete CUP beendet werden kann. Das Clientmenü besitzt die weitere Funktion `search`, mit der die Suche gestartet werden kann.

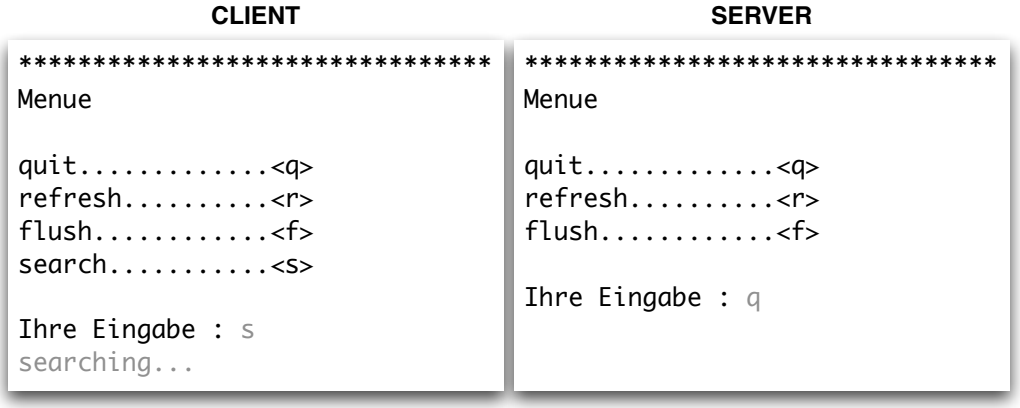


Abbildung 5.4: P2P-Menü des CUPs

Wurde durch die Eingabe `s` + `ENTER` die Suche gestartet, teilt die am laufende Methode `run()` der Klasse `ConsoleInputProcessor` dem `ConsoleCommandListener` dies mit. Entsprechend wird die Methode

`consoleCommandreceived()` der Klasse `P2PClient` aufgerufen. Nun wird mit der Methode `searchFor()` des `P2PClients` der SearchService initialisiert und über die `search()`-Funktion der Klasse `SearchServiceImpl` wird der Suchtext und die E-Mail Adresse des Benutzers für die Übertragung an das P2P-Netzwerk vorbereitet.

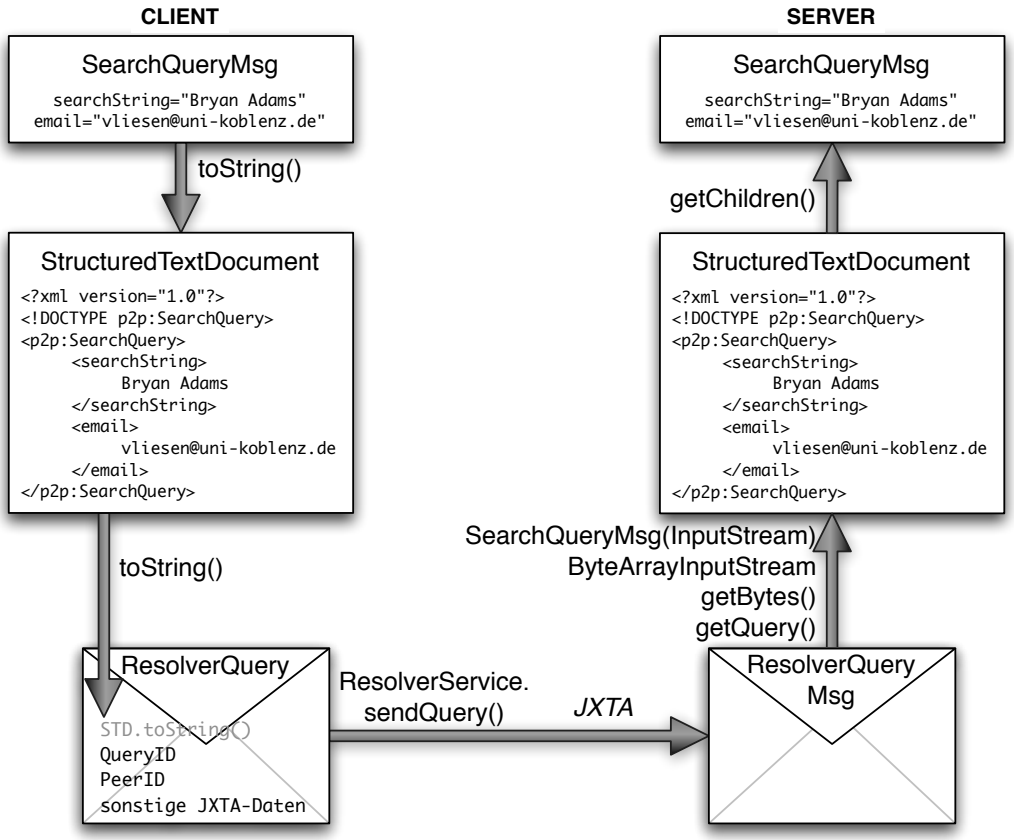


Abbildung 5.5: Beispiel-Datenübertragung mit JXTA

Wie in Abbildung 5.5 gezeigt wird zunächst eine `SearchQueryMsg` erzeugt und diese mit Hilfe der überschriebenen Methode `toString()` in den JXTA-Typ `StructuredTextDocument` (STD) umgewandelt. Dieses Objekt wird zusammen mit weiteren übertragungsnotwendigen Informationen wie der AnfrageID und der PeerID des Clients in das erzeugte Objekt der `ResolverQuery` gespeichert (siehe Listing 5.5). Anschließend wird das Objekt der `ResolverQuery` über den `ResolverService`

von JXTA an den SearchService gesendet. Bei dem abhörenden Server kommt nun ein Objekt der ResolverQueryMsg an, welches durch verschiedene Schritte wieder zu einem Objekt der SearchQueryMsg umgewandelt wird. Beispielsweise wird ein ByteArrayInputStream[3] aus dem angekommenen Objekt erzeugt, woraus die XML-Struktur des STDs und die benötigten Informationen herausgefiltert werden können.
Das entstandene SearchQueryMsg-Objekt wird anhand eines SearchRequestEvents an die SearchServiceListener übergeben. Dazu gehört der P2PServer, dessen Methode processSearch-Request() anhand des empfangenen Suchtextes eine Ergebnisliste erzeugt. Diese wird in ein Objekt der SearchResponseMsg gespeichert, welches fast analog zu dem in Abbildung 5.5 dargestellten Ablauf verpackt und gesendet wird. In diesem Fall wird jedoch ein ResolverResponse-Objekt erzeugt (siehe Listing 5.6), welches direkt an den anfragenden Client gesendet wird. Dieser empfängt wie in Abbildung 5.7 auf Seite 105 zu erkennen ein Objekt der ResolverResponseMsg.

Nachdem der Client ein Objekt des SearchServiceEvents erstellt hat wird anhand des SearchServiceListeners die Methode processAnswer() des P2PClients aufgerufen. Die Ergebnisliste wird ausgelesen und die erste Datei dieser wird zum Download ausgewählt. Mit der Methode askForDownload() wird nun ein Objekt der DownloadServiceImpl erstellt und der DownloadService wird verwendet um letztendlich die ausgewählte Datei herunterladen zu können. Die Aufrufe sind ähnlich dem gerade beschriebenen SearchService-Szenario und in Abbildung 5.8 und Abbildung 5.9 ab Seite 106 skizziert.

[3]Java-Klasse ByteArrayInputStream; URL: http://java.sun.com/j2se/1.4.2/docs/api/java/io/ByteArrayInputStream.html; Stand: 09.05.2010

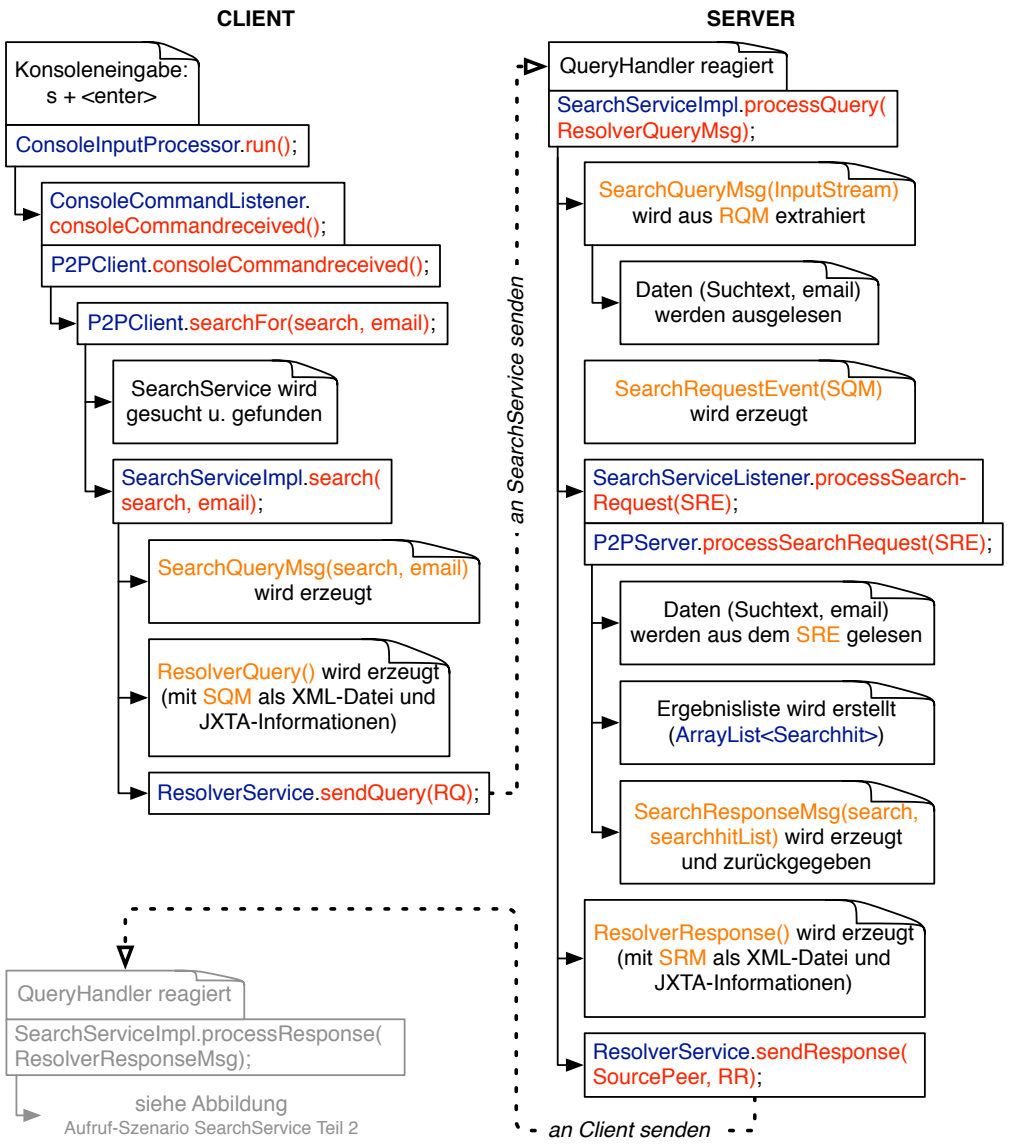


Abbildung 5.6: Aufrufszenario SearchService Teil 1

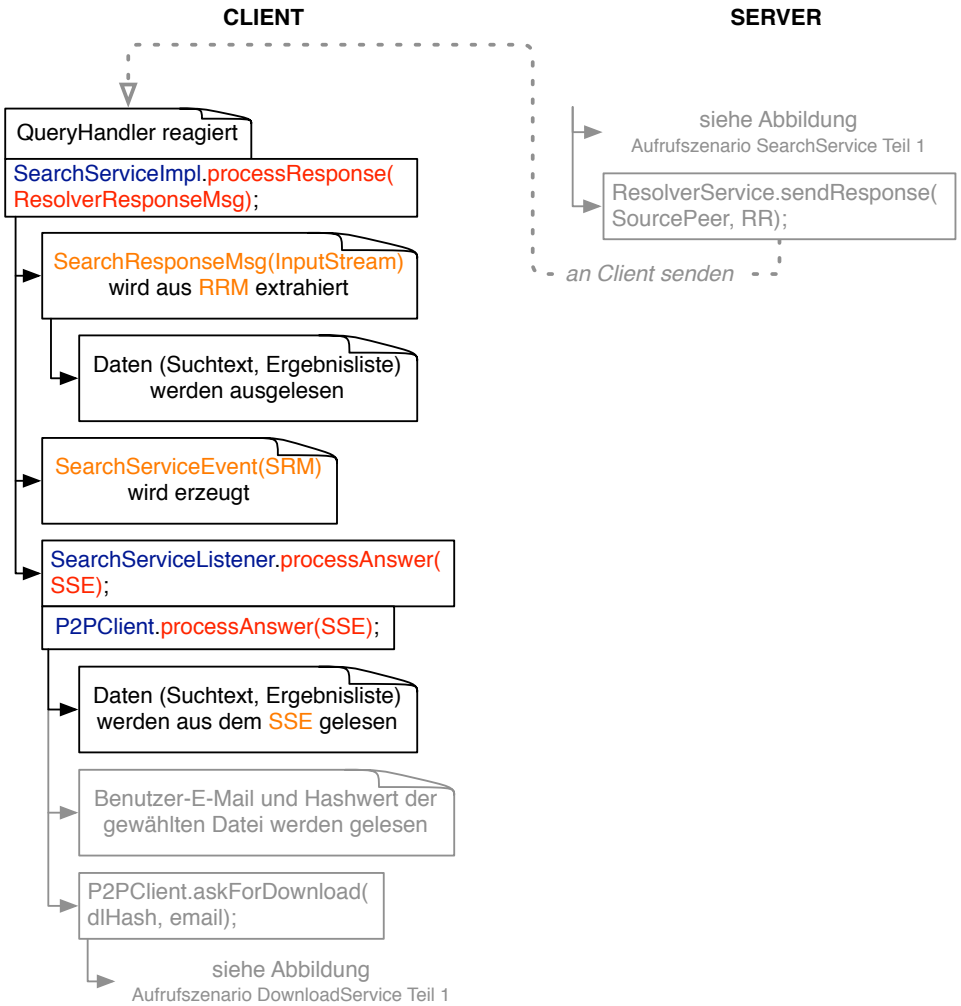


Abbildung 5.7: Aufrufszenario SearchService Teil 2

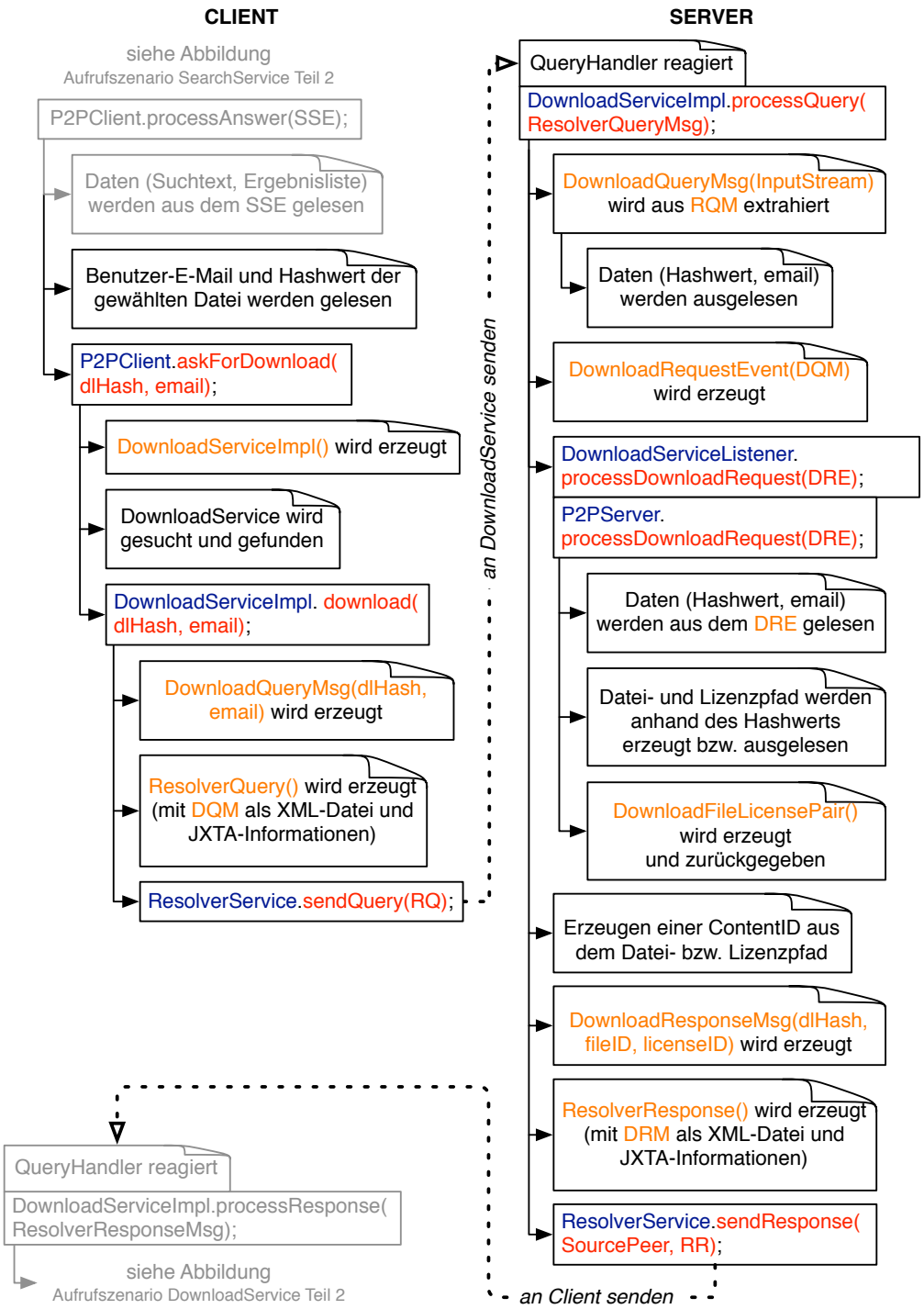


Abbildung 5.8: Aufrufszenario DownloadService Teil 1

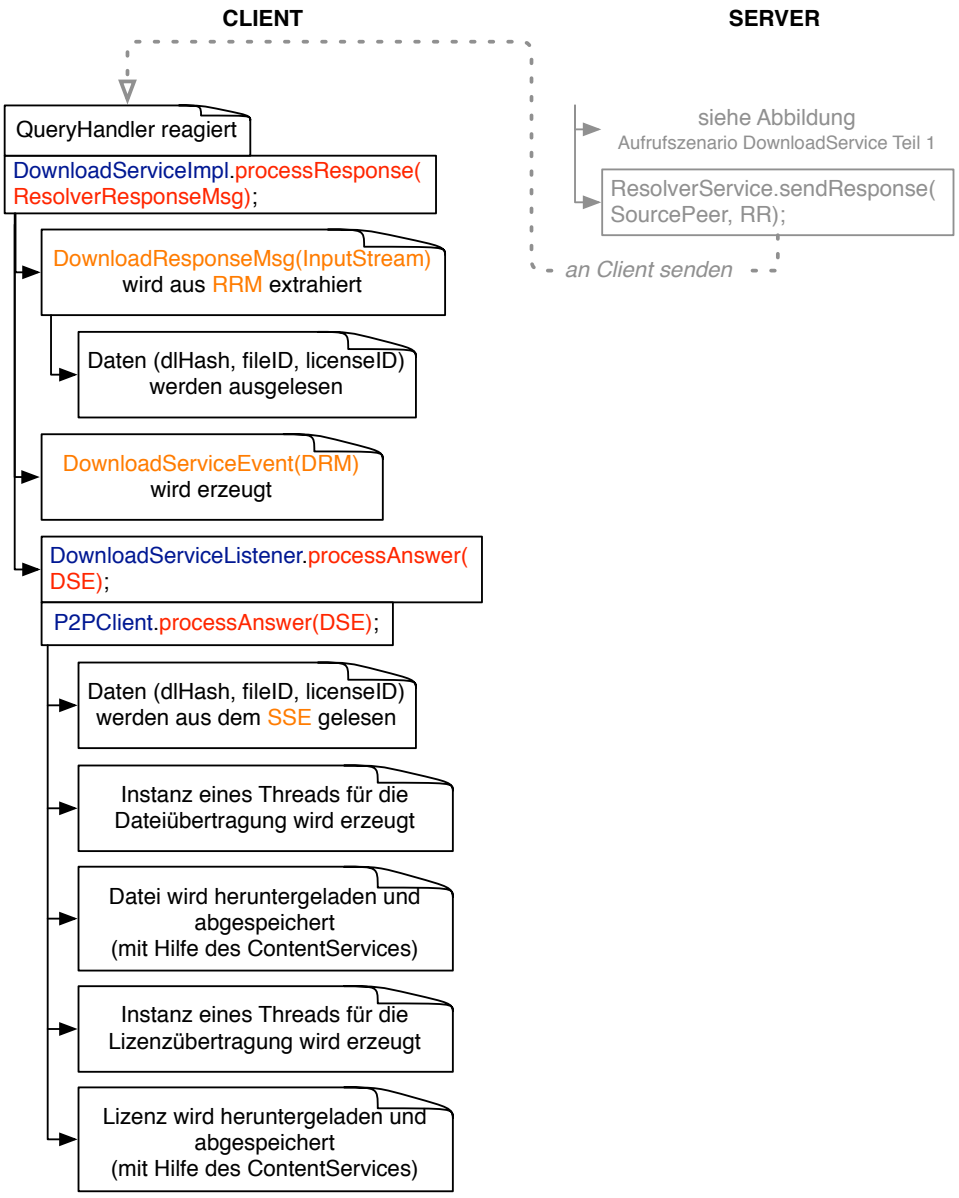


Abbildung 5.9: Aufrufszenario DownloadService Teil 2

# 6 Anwendungsaspekt

Nachdem in den vorherigen Kapiteln die Entwicklung des CUPs beschrieben wurde, wird in diesem Kapitel ein weiterer wichtiger Gesichtspunkt dieser Arbeit herausgestellt, der Anwendungsaspekt. Der in dieser Diplomarbeit entwickelte CUP stellt einen Teil des gesamten TURM-Systems dar, weshalb zunächst auf die Anwendbarkeit des gesamten URM-Bereichs, also dem TURM-System eingegangen wird. Danach werden die Anwendungsmöglichkeiten des CUPs als Erweiterung des TURM-Systems genauer beschrieben.

## 6.1 URM im Allgemeinen

In Abbildung 6.1 ist ein Szenario dargestellt, wie das Vorgehen in Bezug auf die URM-Mediathek im Einzelnen funktioniert.

Der Nutzer kann mit Hilfe der URM-Mediathek seine Musikdateien verwalten und die zugehörigen Rechte anhand der Ampelfarbenstruktur einsehen. Weiterhin kann der Nutzer sich durch Rechtsklick zu jeder Datei die Lizenzinformationen anzeigen lassen oder eine neue Lizenz (mit Informationen über ihre Rechte, abhängig von der Datei-Herkunft) erzeugen. Dabei werden die Lizenzdateien zu den entsprechenden Musikdateien von der Mediathek gelesen und ausgewertet.

Zusammenfassend können folgende Anwendungen für den Nutzer in Betracht gezogen werden:

- Informieren über seine Rechte. *(Prototyp vorhanden)*

- CD-Rippen mit Erzeugung von Lizenzen. *(Prototyp vorhanden)*
- Analyse vorhandener Dateien. *(in Planung)*
- Web-Download mit Erzeugung von Lizenzen. *(in Planung)*

Wie in der vorangegangenen Auflistung ersichtlich, sind bereits einige Anwendungen im frühen Stadium realisiert.

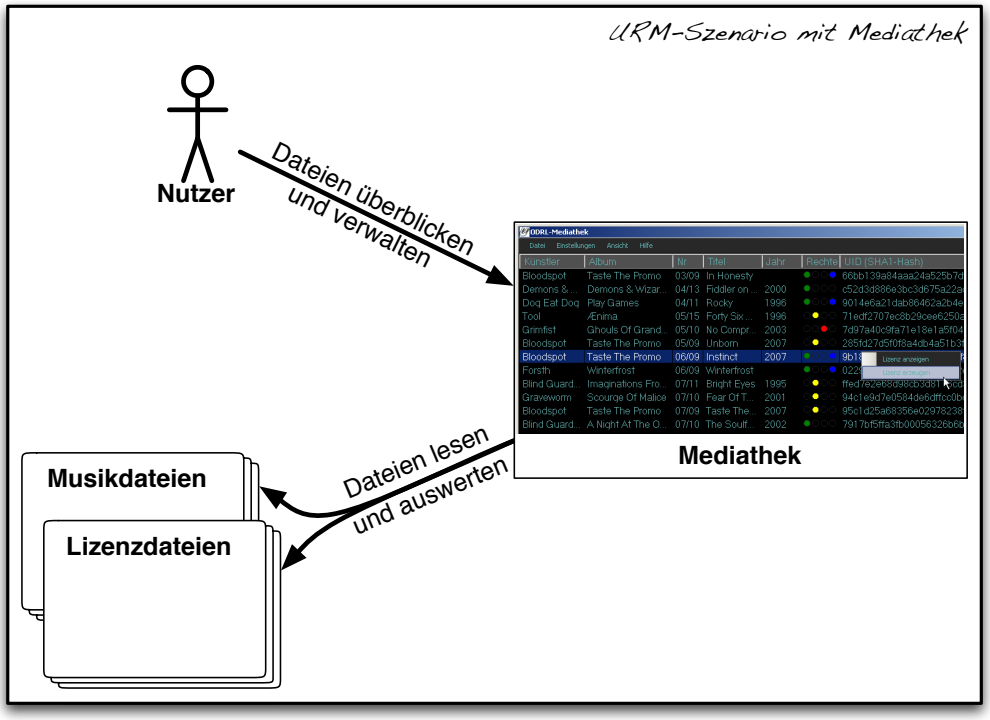


Abbildung 6.1: URM-Szenario mit Mediathek

Das URM-Konzept ist eine Lösung für alle Nutzer, die sich in Bezug auf digitale Rechte rechtskonform verhalten möchten. Weiterhin ist es darauf ausgelegt, dass nicht nur Musikdateien unterstützt werden können, sondern eine Erweiterung auf beliebige Dateiformate von digitalen Inhalten, wie z.B. Videos oder Bilder, möglich ist. Der offene Aufbau von URM unterstützt ein breites Spektrum an Erweiterungen, beispielsweise CD-Rip-Funktionen oder P2P-Netzwerke. Ein solches ist der CUP, dessen Anwendungsbereich in dem folgenden Kapitel beleuchtet wird.

## 6.2 CUP im Speziellen

Der CUP basiert auf der P2P-Technologie, welche nach [Hundacker u. a., 2009] sowohl für legale als auch für illegale Zwecke eingesetzt werden kann. Dabei liegt es in der Hand des Benutzers zu entscheiden, ob eine Datei heruntergeladen werden soll und zu hoffen, dass die entsprechenden Datei legal ist. Dies ist beim CUP nicht der Fall, denn dieser bietet den Downloadern nur Dateien an, die entsprechend der Lizenzinformationen weitergegeben werden dürfen, womit der Downloader legale Dateien (inklusive der bestätigenden Lizenz) erhält.

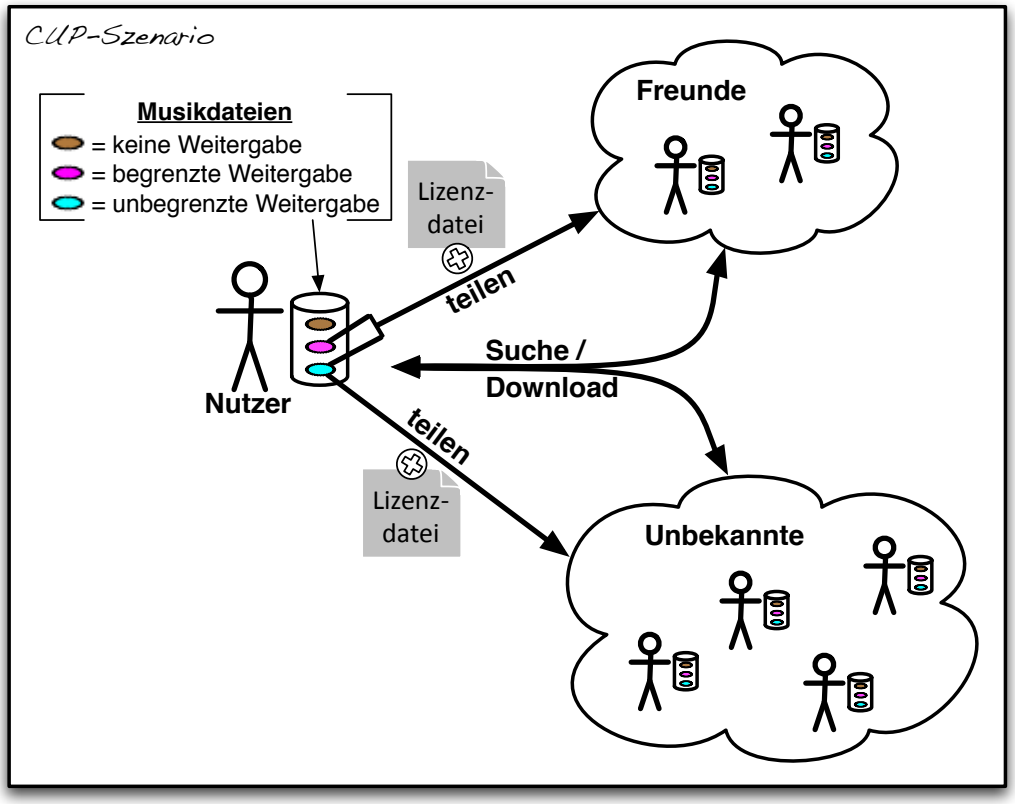


Abbildung 6.2: Szenario des CUPs

Die Anwendungsweise des CUPs ist in Abbildung 6.2 anhand eines Szenarios skizziert. In diesem Szenario steht bewusst der Nutzer im Vordergrund, um ausgehend von ihm die verschiedenen ausführbaren Aktionen

(teilen, suchen und downloaden) übersichtlich darstellen zu können.
Um zu verhindern, dass Dateien mit begrenzter Weitergabe nicht an Personen übermittelt werden, die dem Anbieter unbekannt sind und dadurch für Freunde keine Weitergabe mehr möglich wird, gibt es eine Freundesliste. In dieser Liste, welche in Abbildung 6.2 durch eine Wolke visualisiert ist, sind die Personen aufgeführt, die der Benutzer als Freund gekennzeichnet hat. Das bedeutet, dass Dateien die in der Weitergabe beschränkt sind nur mit den Benutzern der Freundesliste geteilt werden, während alle anderen Dateien, die unbegrenzt weitergegeben werden dürfen, allen Benutzern zur Verfügung stehen. Wird eine Datei vom Nutzer an einen anderen Benutzer geuploaded, wird zusätzlich immer noch die speziell für diese Datei erzeugte Exportlizenz übertragen.

Zusammenfassend wurden folgende Anwendungen durch den CUP hinzugefügt:

- P2P-Download mit Erhalt von Lizenzen. *(Selbstschutz)*
- Upload von nur legalen Inhalten. *(Selbstschutz)*
- Download von nur legalen Inhalten. *(Selbstschutz)*
- Automatischer Austausch von Inhalten, die der „Privatkopie“ unterliegen. *(funktionaler Mehrwert)*

Sobald sichergestellt und überprüft werden kann (z.B. mittels digitaler Signaturen), dass die über den CUP verteilten Lizenzen rechtmäßig sind, können mit der P2P-Technologie hinter dem CUP sicher legale Medien ausgetauscht werden. Somit kann der CUP in Zukunft eine Alternative zu bestehenden P2P-Tauschbörsen bieten.

# 7 Fazit und Ausblick

Um untersuchen zu können, ob die vorgegebenen Ziele erreicht werden konnten und die vorliegenden Ergebnisse den anfänglichen Erwartungen entsprechen, wird zunächst eine Zusammenfassung des vorhandenen CUPs gegeben.
Die Abbildung 7.1 gibt einen Gesamtüberblick über die Funktionsweise des entwickelten CUP-Systems. Der Benutzer kann über die Kommandozeile den CUP als Client oder Server starten und verschiedene Angaben wie Benutzername oder Suchtext hinzufügen. Zusätzlich steht noch die Freund-Funktion zur Verfügung bei der durch Angabe einer E-Mail Adresse diese einer Freundesliste hinzugefügt oder daraus entfernt wird. Je nach Parameterangabe startet der CUP den Client oder den Server. Beide Seiten führen die notwendigen Funktionen wie Instanziieren der NetPeerGroup und der CupPeerGroup aus, um das JXTA-Netzwerk zu initialisieren. Die Serverseite veröffentlicht zusätzlich den SearchService und den DownloadService und fängt an diese abzuhören. Die Clientseite sucht in der CupPeerGroup nach dem Search- und DownloadService und registriert diese. Der Benutzer kann nun auf der Kommandozeile durch Auflistung der *Requester* (Clients) bzw. der *Provider* (Server) erkennen, ob andere Peers gefunden wurden.
Mit der Verwendung einer Kommandozeilen-Eingabe, die über das nach Drücken der `ENTER`-Taste angezeigte Menü abgefragt werden kann, kann der Benutzer nun die Suche mit dem Befehl `s` + `ENTER` starten. Die Suchanfrage wird an die in der CupPeerGroup lauschenden Server gesendet. Bei vorhandenen Suchtreffern, welche über den TURM gesucht werden, wird eine Ergebnisliste an den Client gesendet. Der Client wählt in dieser

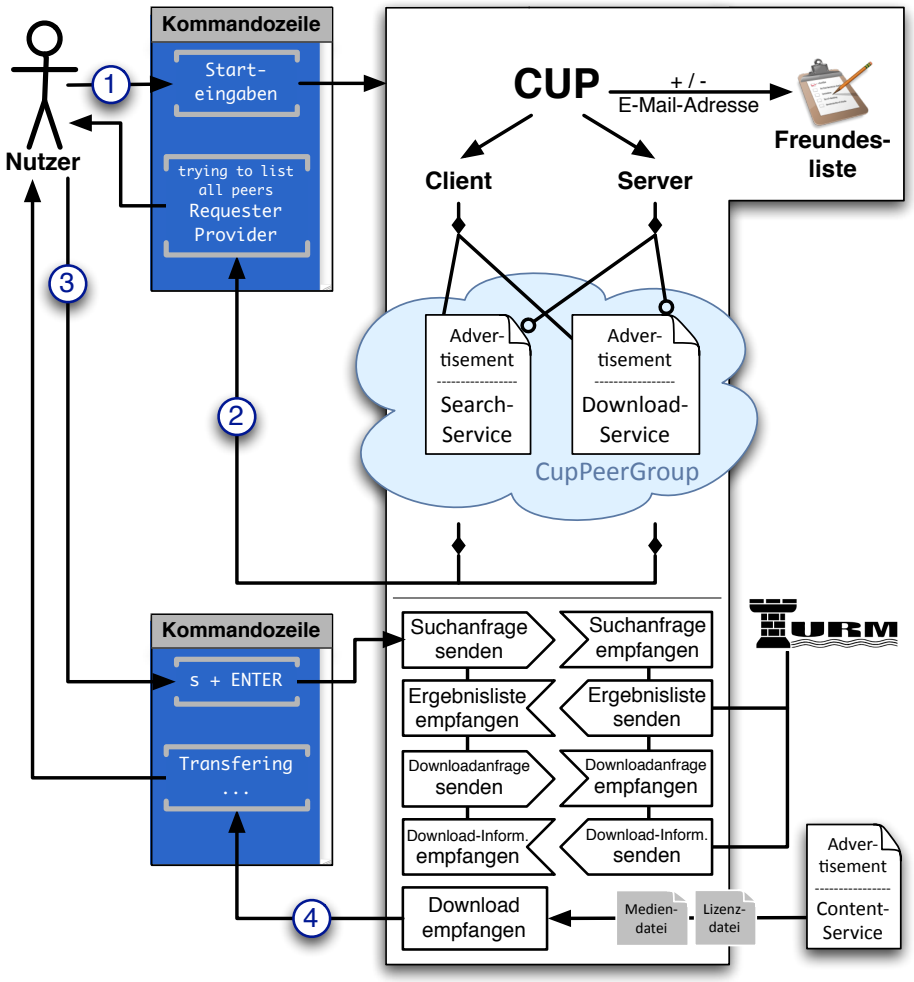


Abbildung 7.1: Gesamtüberblick des CUPs

prototypischen Realisierung immer die erste Datei auf der empfangenen Ergebnisliste aus und fragt diese automatisch an. Nachdem der Server dem Client die notwendigen Informationen zum Download, die mit Hilfe des TURMs erzeugt werden, übertragen hat, kann der Client die Mediendatei und die Lizenzdatei über den ContentService herunterladen. Das Downloaden der Datei wird dem Benutzer des Clients auf der Kommandozeile angezeigt.

Die zentrale Forderung dieser Arbeit, einen lauffähigen Prototypen zum Austausch von URM-basierenden Dateien zu erstellen (siehe Kapitel 1.2),

ist mit der zum Zeitpunkt der Arbeitsabgabe existierenden CUP-Version vollständig erfüllt. Das System kann sowohl Daten austauschen als auch mit Medien- und Lizenzdateien umgehen.
Der Austausch von Dateien über konventionelle Client/Server-Modelle ist zu statisch und nicht flexibel genug. Bei sich ständig verändernden Netzwerkteilnehmern, die die Lizenzvergabe selbständig durchführen, weist das Client/Server-Modell zu große Schwächen auf. Diese Schwächen bestehen unter anderem darin, dass die Frage, wer dazu berechtigt ist eine Datei zu empfangen individuell von jedem Netzteilnehmer, der Dateien bereitstellt, entschieden werden muss. Daher ist die Verwendung einer P2P-Topologie Bedingung für die vollständige Erfüllung der formulierten Anforderungen in Kapitel 4.1.
Die Annahme aus Kapitel 1.3, dass die implementierte P2P-Infrastruktur des CUPs einen Schlüsselfaktor zur erfolgreichen Umsetzung der Anforderungen (siehe Kapitel 4.1) repräsentiert, kann vom Autor bestätigt werden. Das JXTA-Framework stellt eine umfangreiche Grundfunktionalität für P2P-Anwendungen zur Verfügung und weist damit eine hohe Komplexität auf. Die zum Teil mangelhafte Dokumentation und das Fehlen von nachhaltigen Anwendungsbeispielen erschweren die Verwendung der JXTA-Konzepte.
Im vorgegebenen Zeitrahmen wurde der Austausch von Dateien innerhalb des CUP-Systems realisiert. Der Datenaustausch über verschiedene Netzwerke hinweg kann unter Verwendung der Rendezvous-Funktion bewerkstelligt werden. Im Prototypen wurde das Auffinden von Peers, die sich außerhalb des eigenen Netzwerks befinden umgesetzt. Dazu muss ein „Super-Peer“ mit der implementierten Funktion Rendezvous (siehe Kapitel 2.2.3) gestartet werden. Über diesen Knoten können bis zum Zeitpunkt der Abgabe noch keine Serviceanfragen bzw. -nachrichten ausgetauscht werden. Diese Funktion kann in auf dieser Arbeit aufbauenden Implementierungen weiterführend umgesetzt werden.

Durch den CUP wächst das TURM-System um eine weitere Komponente an. Benutzer können sich nicht nur wie bisher Informationen über ihre

Mediendateien anzeigen lassen und ihre Lizenzen verwalten. Die Möglichkeit des Austauschs von legalen Mediendateien mit anderen Personen wird durch das in der vorliegenden Diplomarbeit entwickelte CUP-System gegeben. Der Austausch kann mit Unbekannten oder Freunden stattfinden, während die Freunde zusätzlich zu den Dateien mit unbegrenztem Weitergabestatus auch Dateien mit begrenztem Weitergabestatus angeboten bekommen.

Im Prozess der prototypischen Realisierung des CUPs wurden notwendige und mögliche Weiterentwicklungen und Verbesserungen identifiziert, die in künftigen Versionen das Basissystem erweitern sollten bzw. können. Dazu gehört beispielsweise das Einsetzen von spezialisierten Peer-Knoten („Super-Peers“ siehe Kapitel 2.2.3) für Zusatzdienste. Solche Dienste können z.B. das Verwalten von IP-Adressen anderer Peers in der CupPeerGroup zur Stabilisierung des Netzwerks oder das Cachen von Suchanfragen für Effizienzsteigerungen sein. Mit diesen Super-Peers könnte sich jeder CUP beim Start über die bekannte IP-Adresse verbinden und kennt automatisch die Peers, dessen IP-Adressen im Super-Peer gespeichert sind bzw. bekommt vorhandene Suchanfragen direkt von diesem Peer. Ein Einsatzszenario für das Cachen von Suchanfragen ist beispielsweise, wenn ein spezialisierter „Cache“-Peer die Antwortlisten der Server zentralisiert und eine netzwerkweite Dateiliste aufbaut. Dies bietet des Weiteren die Möglichkeit, eine Datei verteilt über mehrere Server-Peers gleichzeitig zu laden. Hierbei müssen noch zwei Herausforderungen bewältigt werden. Erstens kann der Client sich nicht sicher sein, ob eine Datei in der globalen Dateiliste wirklich für ihn downloadbar ist, da die Datei eventuell einen begrenzten Weitergabestatus hat. Zweitens muss beim gleichzeitigen Laden von mehreren Quellen sichergestellt sein, dass beim Downloadvorgang nur eine Lizenz erzeugt werden darf. Dies setzt beim Beginn des Downloads einen festgeschriebenen Kommunikationsablauf zwischen den anbietenden Servern voraus.
Eine zusätzliche Weiterentwicklung des CUPs stellt die Spezialisierung der Suche bzw. des SearchServices dar. Unter Spezialisierung wird in

diesem Zusammenhang das Suchen nach Metadaten in den Mediendateien verstanden. Bislang wird die Suche nur auf Dateiattributen (=Dateiname) durchgeführt. Spezielle Instanzen des SearchServices könnten beispielsweise beim Austausch von MP3-Dateien sowohl Titel, Interpret als auch Jahr oder Album indizieren und damit die Suche einschränken.
Zwischen Client- und Serverfunktion sollte in naher Zukunft nicht mehr gewählt werden müssen. Beide Funktionen können mit Hilfe von `Threads`[1] parallel ausführbar gemacht werden und von einer durchgängigen Benutzersteuerungsschicht unterstützt werden.
Das umfangreiche Testen des CUPs durch eine angemessene Anzahl von Testpersonen erscheint sinnvoll, nachdem beispielsweise die geplante grafische Benutzeroberfläche (GUI) für den gesamten TURM umgesetzt wurde. Ein weiterer Aspekt, der die Akzeptanz des CUPs zusammen mit dem TURM-System erhöhen kann, ist die Etablierung von URM durch digitale Signaturen und die rechtliche Anerkennung.

Die vorliegende Diplomarbeit löst die Herausforderungen die in Kapitel 1.3 dargelegt wurden. Dabei werden wissenschaftliche Methoden unter anderem aus der Softwareentwicklung verwendet, um bisher unbeantwortete Fragestellungen zu lösen. Die eigene Arbeit bei der Konzipierung und Umsetzung des CUP-Prototypen und die Bestätigung der Hypothesen stellen einen Erkenntnisgewinn dar.

[1] Java-Klasse `Thread`; URL: http://java.sun.com/j2se/1.3/docs/api/java/lang/Thread.html; Stand: 10.05.2010

# A Anhang: Anforderungen

| Nr. | Anforderung | Typ |
|---|---|---|
| 01 | Der CUP soll als Komponente des TURM-Systems implementiert werden und an sinnvollen Stellen Funktionalitäten anderer TURM-Komponenten nutzen. | A |
| 02 | Der Benutzer muss das CUP-System über die Kommandozeile starten können. | A |
| 03 | Beim Starten des Systems muss der Benutzer die für den CUP notwendigen Informationen „Benutzername“, „E-Mail-Adresse“, „Client, Server oder Freund?“, bei Wahl des Clients „Suchtext“ und bei Wahl des Freundes „E-Mail-Adresse“ angeben können. | A |
| 04 | Der CUP wird nur dann gestartet, wenn die Kommandozeilen-Parameter korrekt sind. | A |
| 05 | Das System muss dem Benutzer anzeigen, wenn die Kommandozeilen-Parameter nicht korrekt sind. | A |
| 06 | Die Kommandozeilen-Parameter müssen im vorhandenen ConfigurationManager des TURM-Systems hinterlegt werden. | A |
| 07 | Die Netzwerkfunktionen des Systems sollen mit Hilfe des Java-Frameworks JXTA implementiert werden. | A |
| 08 | Ein laufender CUP soll zunächst entweder nur Clientfunktionen oder nur Serverfunktionen ausführen. | A |
| 09 | Der Benutzer muss in den Dateien anderer Peers suchen können. | C |

*wird fortgesetzt*

*Fortsetzung*

| Nr. | Anforderung | Typ |
|---|---|---|
| 10 | Der CUP in seiner Funktion als Client muss es dem Benutzer ermöglichen, gefundene Dateien von anderen Peers herunterladen zu können. | C |
| 11 | Der CUP muss in seiner Funktion als Server Dateien zur Verfügung stellen. | S |
| 12 | Die vom CUP zur Verfügung gestellten Dateien müssen weitergegeben werden dürfen (Datei-Weitergabe- und Personenstatus überprüfen). | S |
| 13 | Bevor eine Datei vom CUP geuploaded werden darf, muss eine Exportlizenz erstellt werden, die dann zusammen mit der Datei hochgeladen wird. | S |
| 14 | Der Benutzer soll andere Benutzer anhand dessen E-Mail-Adresse als Freund kennzeichnen können. | A |
| 15 | Der CUP muss zunächst nicht alle digitalen Inhalte, sondern nur MP3-Dateien unterstützen. | A |

A: Allgemein; C: Clientfunktion des CUPs; S: Serverfunktion des CUPs

Tabelle A.1: Anforderungsliste

# B Anhang: UML-Diagramme

Im Folgenden sind die während der Diplomarbeit herausgearbeiteten Aktionen der Komponente CUP (siehe Abbildung 4.4 auf Seite 75) als Aktivitäts- bzw. Sequenzdiagramme der UML dargestellt.

## B.1 *CUP starten*

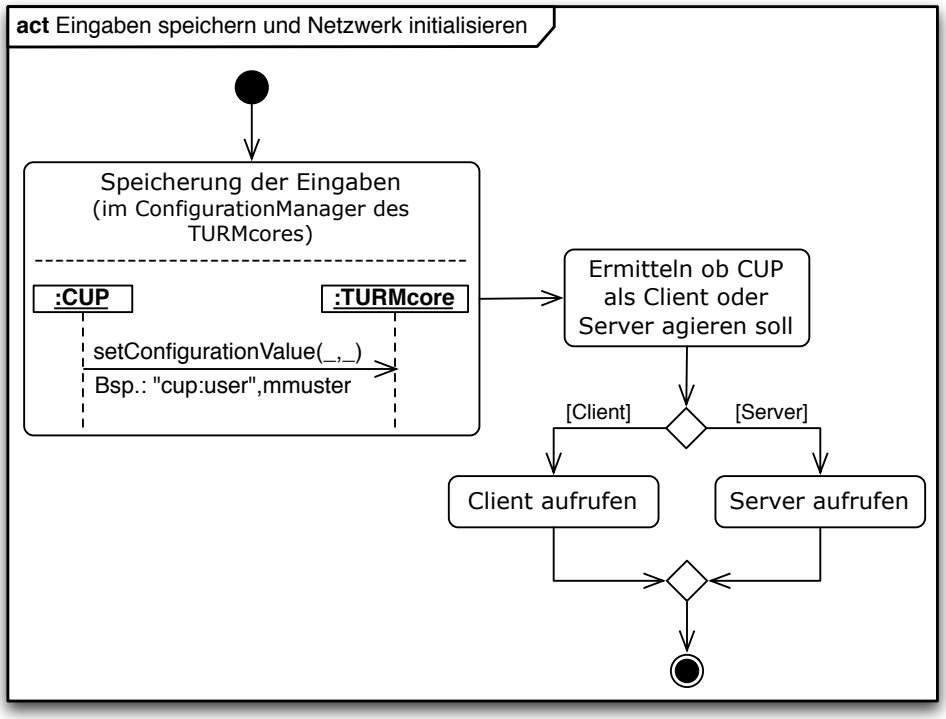


Abbildung B.1: Aktivitätsdiagramm *Eingaben speichern und Netzwerk initialisieren*

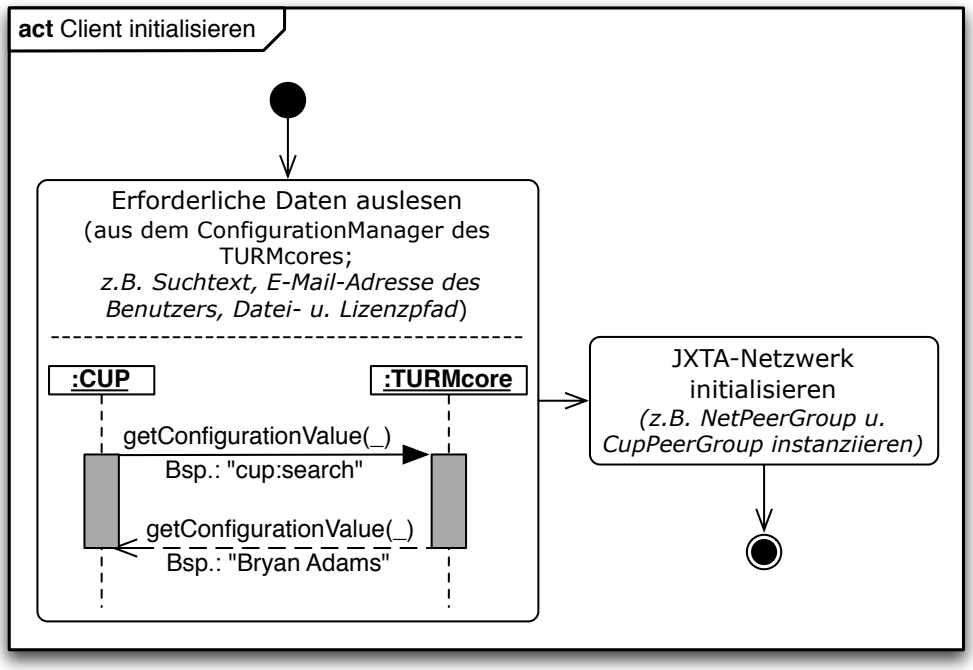


Abbildung B.2: Aktivitätsdiagramm *Client initialisieren*

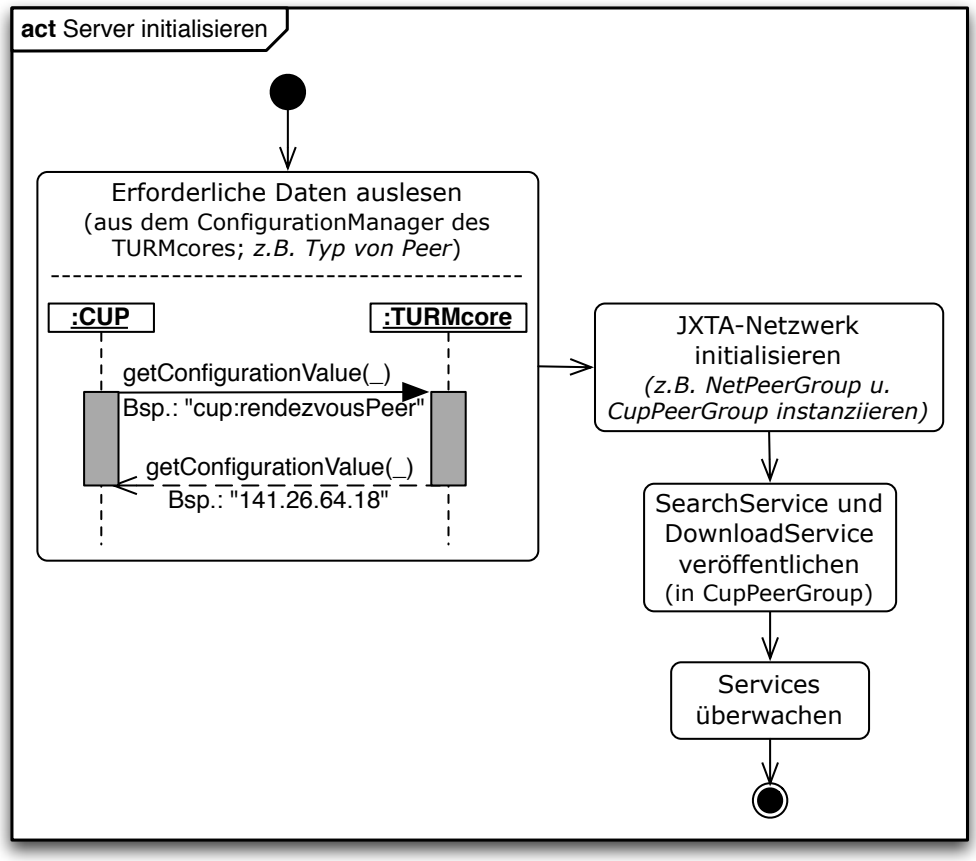


Abbildung B.3: Aktivitätsdiagramm *Server initialisieren*

## B.2 *Dateien suchen* und *Dateien herunterladen*

Die folgenden UML-Diagramme stellen sowohl den Anwendungsfall *Dateien suchen* als auch *Dateien herunterladen* aus dem in Abbildung 4.4 auf Seite 75 gezeigten CUP-Programmierablauf dar. Dabei sind die Anwendungsfälle *Dateien zur Verfügung stellen*, *Datei-Weitergabestatus prüfen*, *Personenstatus prüfen* und *Netzwerk verwenden* durch das Suchen und Herunterladen der Dateien automatisch inbegriffen und nicht explizit genannt.

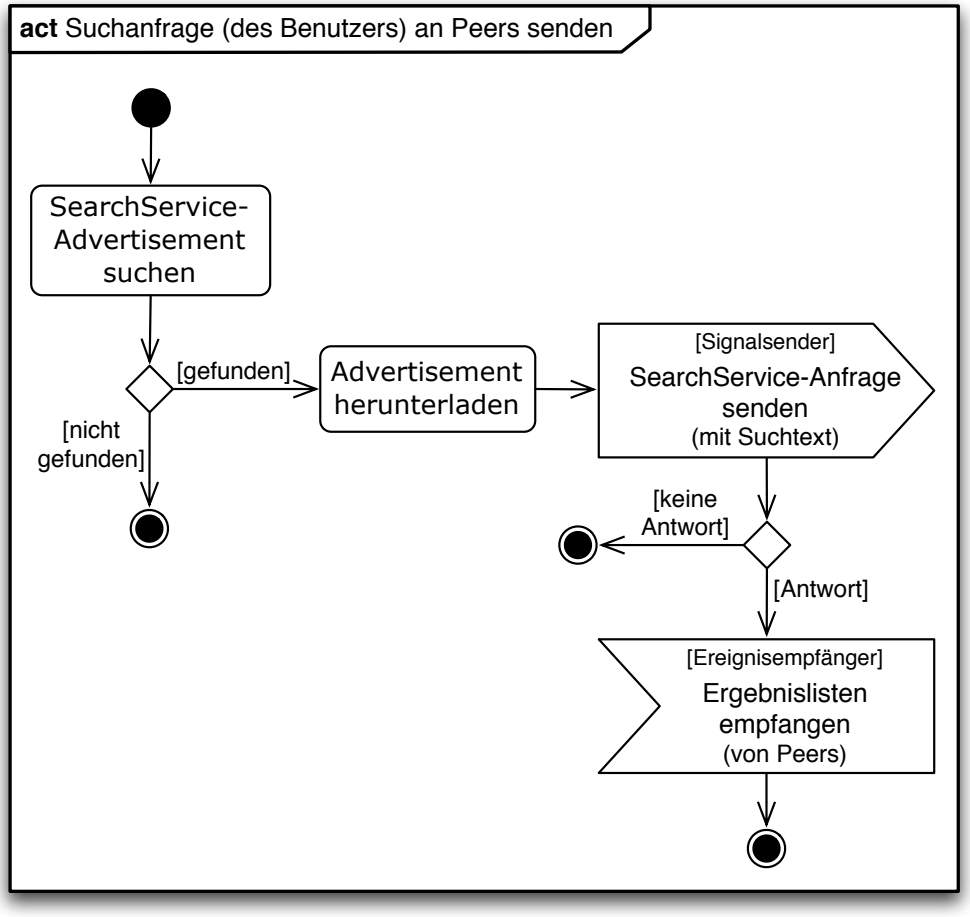


Abbildung B.4: Aktivitätsdiagramm *Suchanfrage (des Benutzers) an Peers senden* (CUP in Clientfunktion)

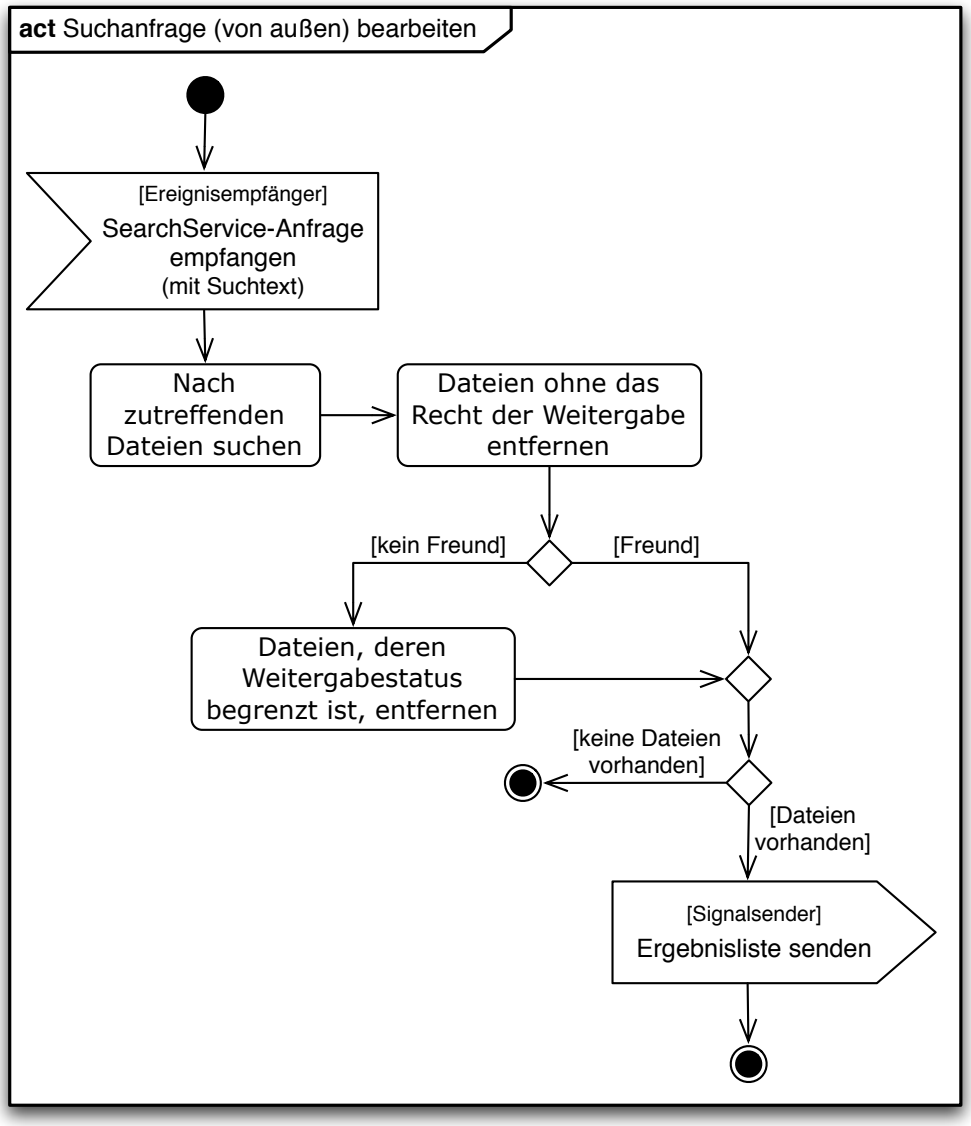


Abbildung B.5: Aktivitätsdiagramm *Suchanfrage (von außen) bearbeiten* (CUP in Serverfunktion)

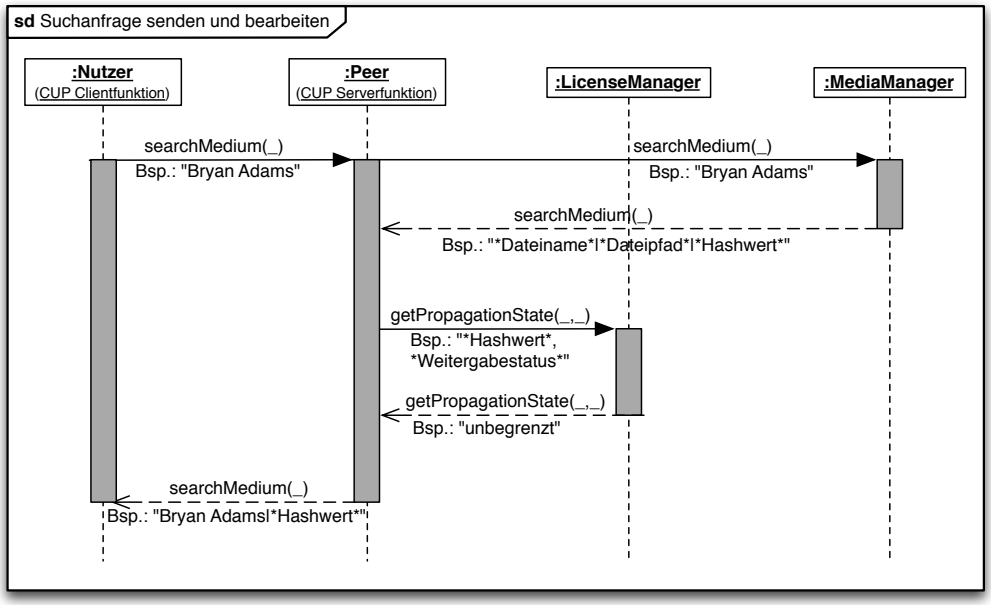


Abbildung B.6: Sequenzdiagramm *Suchanfrage senden und bearbeiten*

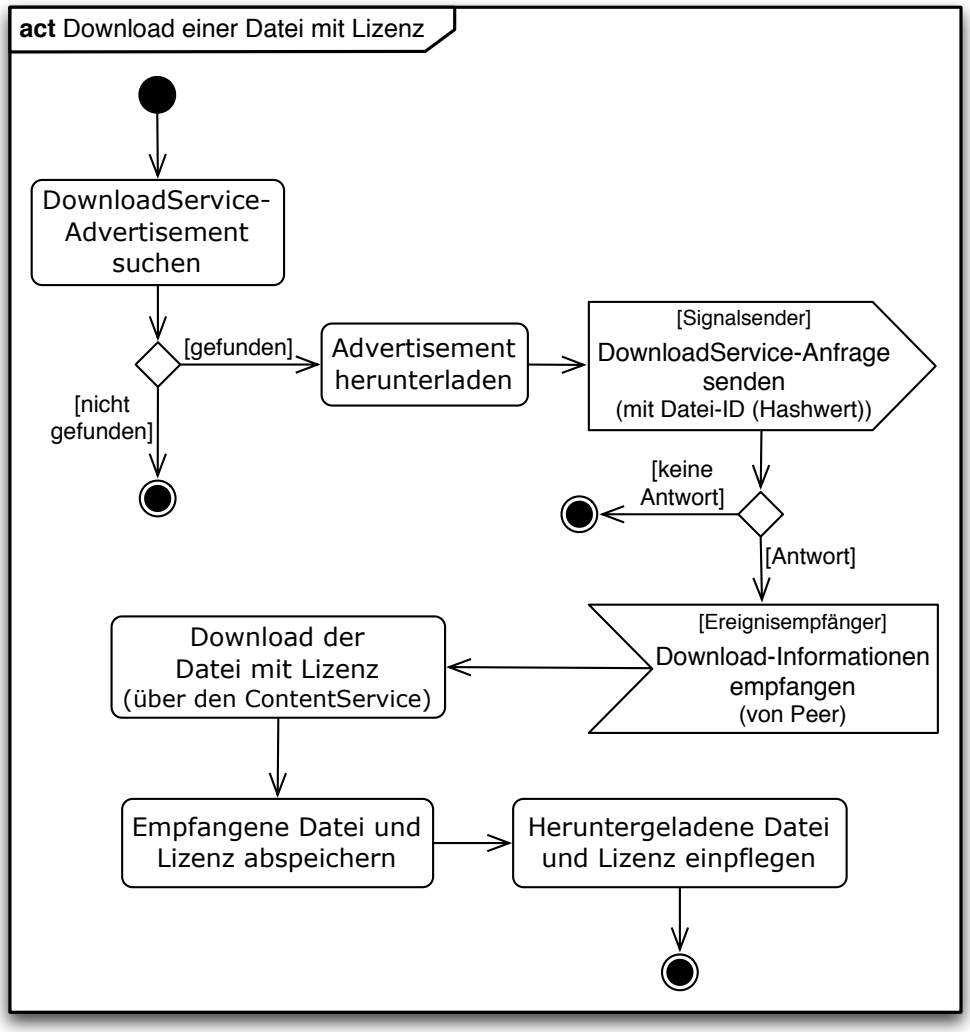


Abbildung B.7: Aktivitätsdiagramm *Download einer Datei mit Lizenz* (CUP in Clientfunktion)

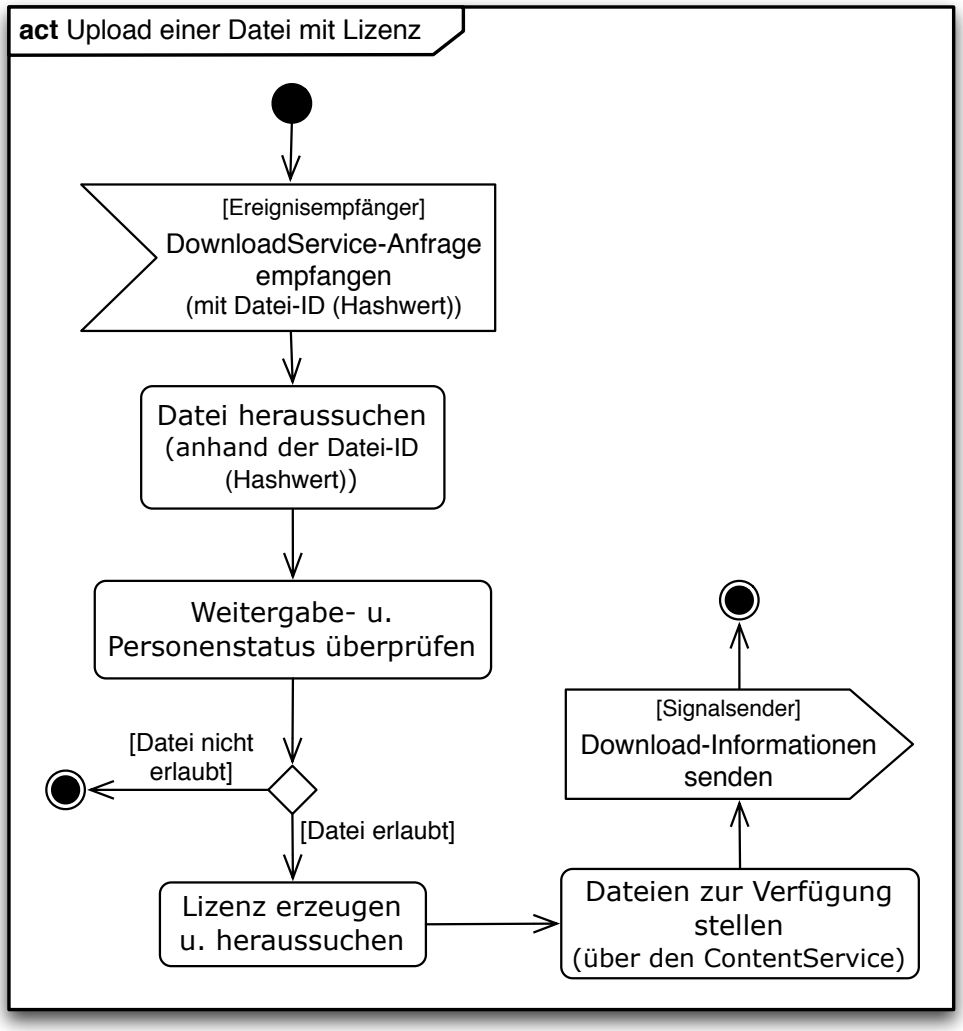


Abbildung B.8: Aktivitätsdiagramm *Upload einer Datei mit Lizenz* (CUP in Serverfunktion)

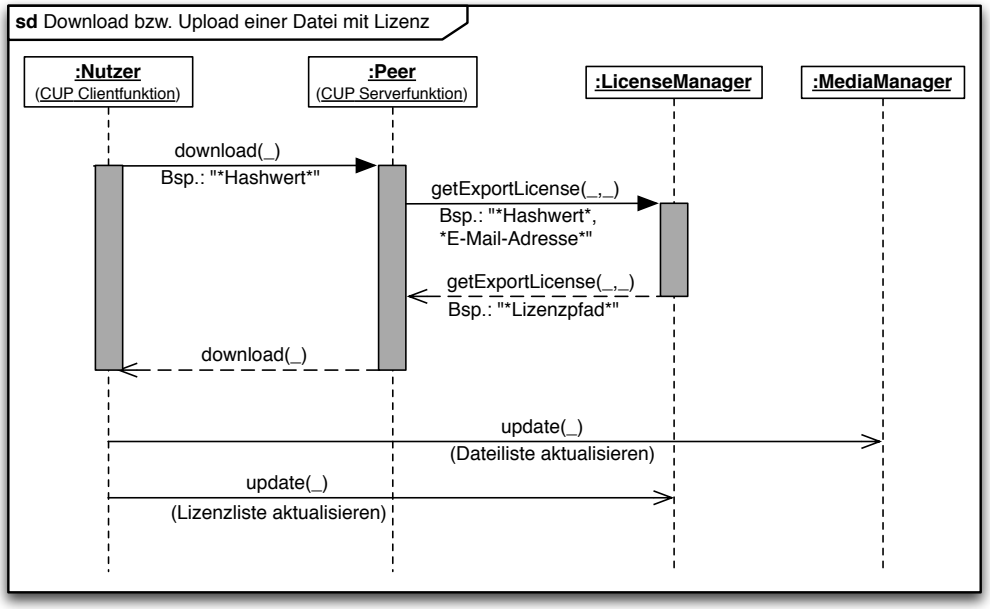


Abbildung B.9: Sequenzdiagramm *Download bzw. Upload einer Datei mit Lizenz*

# C Anhang: Details zur Implementierung

Im Folgenden sind wichtige Klassen und Interfaces der CUP-Implementierung mit ihren Methoden dargestellt.

## C.1 Allgemeine Klassen und Interfaces

| Klasse (Interface) | Methoden |
|---|---|
| Cup | getConfigurationManager() : ConfigurationManager<br>getTurmMediaManager() : TurmMediaManager<br>getTurmLicenseManager() : TurmLicenseManager<br>saveArguments(args) : void<br>updateFriendsList(args) : void |
| CupNetworkManager | startNetworkManager() : void |
| ConsoleInputProcessor | run() : void |
| ConsoleCommandListener | consoleCommandreceived(command) : void |
| P2P | P2P(name, tcpPort, configMode) |
| P2PClient | P2PClient(name, tcpPort, configMode, cup)<br>startCln() : void<br>consoleCommandreceived(command) : void<br>searchFor(search, email) : void<br>processAnswer(SearchServiceEvent) : void<br>askForDownload(dlHash, email) : void<br>processAnswer(DownloadServiceEvent) : void |
| P2PServer | P2PServer(name, tcpPort, configMode, cup) : void<br>startSrv() : void<br>processSearchRequest(SearchRequestEvent)<br>: SearchResponseMessage<br>processDownloadRequest(DownloadRequest-<br>Event) : DownloadFileLicensePair |

Tabelle C.1: Allgemeine Klassen und Interfaces des CUPs

## C.2 Service-Klassen und -Interfaces

| Klasse (Interface) | Methoden |
|---|---|
| ...ServiceListener (Interface) | processAnswer(SearchServiceEvent) : void / processAnswer(DownloadServiceEvent) : void processSearchRequest(SearchRequestEvent) : SearchResponseMsg / processDownloadRequest(Download-RequestEvent) : DownloadFileLicensePair |
| ...ServiceImpl | search(search, email) : void / download(dlHash, email) : void processQuery(ResolverQueryMsg) : int processResponse(ResolverResponseMsg) : void |
| ...Service (Interface) | search(search, email) : void / download(dlHash, email) : void |
| ...ServiceConfigAdv | getAdvertisementType() : String |
| ...QueryMsg | SearchQueryMsg(search, email) / DownloadQueryMsg(dlHash, email) SearchQueryMsg(InputStream) / DownloadQueryMsg(InputStream) toString() : String |
| ...RequestEvent | SearchRequestEvent(SearchQueryMsg) / DownloadRequestEvent(DownloadQueryMsg) |
| ...ResponseMsg | SearchResponseMsg(search, searchhitList) / DownloadResponseMsg(dlHash, fileID, licenseID) SearchResponseMsg(InputStream) / DownloadResponseMsg(InputStream) getDocument(MimeMediaType) : Document |
| ...ServiceEvent | SearchServiceEvent(SearchResponseMsg) / DownloadServiceEvent(DownloadResponseMsg) |
| DownloadFileLicensePair | getFile() : String, setFile(String) : void getLicense() : String, setLicense(String) : void |

Tabelle C.2: Service-Klassen und -Interfaces des CUPs

# D Anhang: Datenträger

Auf der beigefügten CD-ROM sind folgende Daten gespeichert:

- Die vorliegende Ausarbeitung im `PDF`-Format.
- Der CUP-Quellcode mit dem benötigten TURM-Code als `ZIP`-Archiv.
- Der CUP in einer ausführbaren `JAR`-Datei mit den benötigten Menüdateien und einer README.

# Literaturverzeichnis

[Alisch u. a. 2005] ALISCH, Katrin ; WINTER, Eggert ; ARENTZEN, Ute: *Gabler Wirtschaftslexikon*. Gabler Verlag, 2005

[Arlt 2006] ARLT, Christian: *Digital Rights Management Systeme. Der Einsatz technischer Maßnahmen zum Schutz digitaler Inhalte.* Beck Juristischer Verlag, 2006

[Asche u. a. 2008] ASCHE, Michael ; BAUHUS, Wilhelm ; MITSCHKE, Ernest ; SEEL, Bernd: *Open Source: Kommerzialisierungsmöglichkeiten und Chancen für die Zusammenarbeit von Hochschulen und Unternehmen*. Waxmann Verlag, 2008

[BDSG 2009] BDSG: *Bundesdatenschutzgesetz*. Bundesministerium der Justiz, 08 2009. – http://bundesrecht.juris.de/bdsg_1990/index.html, Stand: 02.02.2010

[Boles 2005] BOLES, Dietrich: *Objektorientierte Softwareentwicklung spielend gelernt mit dem Java-Hamster-Modell*. September 2005. – http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/band5/hamster5-html/hamster5.html, Stand: 15.09.2009

[Braun u. a. 2004] BRAUN, Christian ; HAFNER, Martin ; WORTMANN, Felix: *Methodenkonstruktion als wissenschaftlicher Erkenntnisansatz*. Februar 2004. – http://www.alexandria.unisg.ch/Publikationen/24346, Stand: 14.09.2009

[Chlebek 2006] CHLEBEK, Paul: *User Interface-orientierte Softwarearchitektur*. Vieweg, 2006

[Daniel u. a. 1999] DANIEL, Eric D. ; MEE, C. D. ; CLARK, Mark H.: *Magnetic Recording - The First 100 Years*. IEEE Press, 1999

[Dreier u. Nolte 2003] DREIER, Thomas ; NOLTE, Georg: Das deutsche Urheberrecht und die digitale Herausforderung. In: *Informatik Spektrum* 26 (2003), Nr. 4, S. 247–256. – http://dx.doi.org/10.1007/s00287-003-0318-0, Stand: 02.02.2010

[Fechner 2007] FECHNER, Frank: *Die Privatkopie: Juristische, ökonomische und technische Betrachtungen*. Universitätsverlag Ilmenau, 2007. – http://www.db-thueringen.de/servlets/DocumentServlet?id=7543, Stand: 01.02.2010

[Gamma u. a. 2009] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2009

[Gehrke 2004] GEHRKE, Nick: *Peer-to-Peer Applikationen für elektronische Märkte*. Deutscher Universitäts-Verlag, 2004. – ISBN 382442181X, 9783824421817

[Gensch u. a. 2009a] *Kapitel* Digitale Musikdistribution und die Krise der Tonträgerindustrie; Huber, Michael. In: GENSCH, Gerhard ; STÖCKLER, Eva M. ; TSCHMUCK, Peter: *Musikrezeption, Musikdistribution und Musikproduktion*. Gabler, 2009, S. 163–185

[Gensch u. a. 2009b] *Kapitel* Vom Tonträger zur Musikdienstleistung; Tschmuck, Peter. In: GENSCH, Gerhard ; STÖCKLER, Eva M. ; TSCHMUCK, Peter: *Musikrezeption, Musikdistribution und Musikproduktion*. Gabler, 2009, S. 141–162

[GG 2009] GG: *Grundgesetz für die Bundesrepublik Deutschland*. Bundesministerium der Justiz, 07 2009. – http://bundesrecht.juris.de/gg/index.html, Stand: 02.02.2010

[Gossmann 2007] GOSSMANN, Christian: *Digital Rights Management Systeme*. Academic Transfer, 2007

[Grechenig u. a. 2009] GRECHENIG, Thomas ; BERNHART, Mario ; BREITENEDER, Roland ; KAPPEL, Karin: *Softwaretechnik - Mit Fallbeispielen aus realen Entwicklungsprojekten*. Pearson Studium, 2009

[Grimm 2006] GRIMM, Rüdiger: E-Commerce - Technik, Entwicklung und Anwendungen. In: *Fraunhofer Technologieführer*. Springer Verlag, 2006

[Grimm u. Nützel 2002] GRIMM, Rüdiger ; NÜTZEL, Jürgen: Geschäftsmodelle für virtuelle Waren. In: *Datenschutz und Datensicherheit (DuD), Vieweg Verlag* 26 (2002), Nr. 5, S. pp. 261–266

[Grimm u. Pähler 2009] GRIMM, Rüdiger ; PÄHLER, Daniel: Sicherheitsanforderungen im Digital Rights Management. In: RULAND, Christoph (Hrsg.): *Wissenschaftliches Kommunikations- und Sicherheitskolloquium 2009* Bd. 25, Shaker Verlag, 2009, S. 33–44

[Haring 2002] HARING, Bruce: *MP3: Die digitale Revolution in der Musikindustrie*. Orange Press, 2002

[Heinrich 2007] HEINRICH, Gert: *Allgemeine Systemanalyse*. Oldenbourg, 2007

[Hong 2001] HONG, Theodore: Performance. In: ORAM, Andy (Hrsg.): *Peer-To-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly Media, 2001

[Hundacker 2008] HUNDACKER, Helge: Forensic DRM. In: GRIMM, Rüdiger (Hrsg.) ; GUTH, Susanne (Hrsg.): *Virtual goods 2008*. Poznan, Poland : Poznan, 2008. – http://virtualgoods.org/2008/161_VirtualGoods2008Book.pdf, Stand: 20.05.2010

[Hundacker u. a. 2009] HUNDACKER, Helge ; PÄHLER, Daniel ; GRIMM, Rüdiger: URM – Usage Rights Management. In: NÜTZEL, Jürgen (Hrsg.) ; ARNAP, Alapan (Hrsg.): *Virtual Goods – Proceedings of the 7th International Workshop for Technology, Economy and Legal Aspects of Virtual Goods, incorporating the 5th International ODRL Workshop*. Nancy, Frankreich : Nancy University Press, September 2009

[IEEE 1998] IEEE: *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Computer Society, 1998. – IEEE Std 830-1998

[Jahn 2010] JAHN, Nico: *ODRL-Mediathek zur Unterstützung von Usage Rights Management*. Studienarbeit an der Universität Koblenz, 2010

[Lahres u. Rayman 2009] LAHRES, Bernhard ; RAYMAN, Gregor: *Objektorientierte Programmierung*. Galileo Computing, 2009

[Lutz 2009] LUTZ, Peter: *Urheberrecht*. C. F. Müller, 2009

[Microsystems 2007] MICROSYSTEMS, Sun: *JXTA Java Standard Edition v2.5: Programmers Guide*. September 2007. – https://guest@jxta-guide.dev.java.net/svn/jxta-guide/trunk/src/guide_v2.5/JXSE_ProgGuide_v2.5.pdf, Stand: 21.04.2010

[Müller-Kalthoff 2002] *Kapitel* Technologien als Enabler für effizientes Cross-Media Publishing; Stamer, Sören. In: MÜLLER-KALTHOFF, Björn: *Cross-Media Management*. Springer-Verlag, 2002, S. 89–124

[Nützel 2006] NÜTZEL, Jürgen: *Die informatorischen Aspekte virtueller Güter und Waren*. Universitätsverlag Ilmenau, 2006

[Nützel u. Grimm 2005] NÜTZEL, Jürgen ; GRIMM, Rüdiger: Musikvertrieb mit Potato Web Services - Kaufanreize für Musik über die Web Services des PotatoSystems. In: *Datenschutz und Datensicherheit (DuD), Vieweg Verlag* 29 (2005), 3, S. pp. 125–129

[Oaks u. a. 2002] OAKS, Scott ; TRAVERSAT, Bernard ; GONG, Li: *JXTA in a nutshell*. First Edition. O'Reilly, 2002

[Papazoglou 2007] PAPAZOGLOU, Michael P.: *Web Services: Principles and Technology*. Pearson Prentice Hall, 2007

[Peterson u. Davie 2008] PETERSON, Larry L. ; DAVIE, Bruce S.: *Computer Networks. A Systems Approach.* Morgan Kaufmann, 2008

[Pichocki 2002] PICHOCKI, Ralf: *Objektorientierte Softwareentwicklung*. 2002. – http://www.pisoftware.de/download/free/Objektorientierung.pdf, Stand: 15.09.2009

[Rupp u. a. 2007] RUPP, Chris ; QUEINS, Stefan ; ZENGLER, Barbara: *UML 2 glasklar: Praxiswissen für die UML-Modellierung*. 3. Auflage. Hanser, 2007

[Schildt 2002] SCHILDT, Herbert: *Java 2 IT-Tutorial*. 1. Auflage. mitp, 2002

[Schmidt u. a. 2004] SCHMIDT, Andreas U. ; TAFRESCHI, Omid ; WOLF, Ruben: Interoperability Challenges for DRM Systems. In: *Presented at: International Workshop for Technology, Economy, Social and Legal Aspects of Virtual Goods*, 2004. – http://virtualgoods.tu-ilmenau.de/2004/Interoperability_Challenges_for_DRM_Systems.pdf, Stand: 01.02.2010

[Sierra u. Bates 2005] SIERRA, Kathy ; BATES, Bert ; LOUKIDES, Mike (Hrsg.): *Head First Java*. Second Edition. O'Reilly, 2005

[Sommerville 2007] SOMMERVILLE, Ian: *Software Engineering*. 8. Auflage. Pearson Studium, 2007

[Steinmetz u. Wehrle 2004] STEINMETZ, Ralf ; WEHRLE, Klaus: Peer-to-Peer-Networking & -Computing. In: *Informatik Spektrum* 27 (2004), Februar, Nr. 1, S. 51–54. – http://dx.doi.org/10.1007/s00287-003-0362-9

[Stryszowski u. Scorpecci 2009] STRYSZOWSKI, Piotr ; SCORPECCI, Danny: *Piracy of Digital Content*. OECD Publishing, 2009

[Takeda u. a. 1990] TAKEDA, H. ; TOMIYAMA, T. ; YOSHIKAWA, H. ; VEERKAMP, P.: Modeling Design Processes. In: *AI Magazine* (1990), S. 37–48

[Tanenbaum 2003] TANENBAUM, Andrew S.: *Computernetzwerke*. Pearson Studium, 2003

[Templeton 2002] TEMPLETON, Daniel: JxGrid Application: Project JXTA in the Sun Grid Engine Context. In: *SunNetwork 2002 Conference and Pavilion*, 2002

[Ullenboom 2009] ULLENBOOM, Christian: *Java ist auch eine Insel*. 8. Auflage. Galileo Computing, 2009

[UrhG 2008] URHG: *Gesetz über Urheberrecht und verwandte Schutzrechte (Urheberrechtsgesetz)*. Bundesministerium der Justiz, 12 2008. – http://bundesrecht.juris.de/urhg/index.html, Stand: 02.02.2010

[Vahidov 2006] VAHIDOV, R.: *Design Research's IS Artifact: a Representational Framework*. DESRIST 2006.Claremont, 24.-25. Februar 2006

[Vaishnavi u. Kuechler 2004] VAISHNAVI, V. ; KUECHLER, W.: *Design Research in Information Systems*. 20. Januar 2004. – http://ais.affiniscape.com/displaycommon.cfm?an=1&subarticlenbr=279, Stand: 14.12.2009

[Vonhoegen 2009] VONHOEGEN, Helmut: *Einstieg in XML*. Galileo Computing, 2009

[Wazeck 2003] WAZECK, Dr. J.: *Projektplanung und Projektoptimierung mit MS Project 2003*. Teia Lehrbuch Verlag, E-Book, 2003. – http://www.teialehrbuch.de/Kostenlose-Kurse/Projektplanung-mit-MS-Project-2003/31136-Wasserfallmodell.html, Stand: 14.12.2009

[Wilson 2002] WILSON, Brendon J.: *JXTA*. First Edition. New Riders Publishing, 2002

[Wolf 2007] WOLF, Jürgen: *Shell-Programmierung: Das umfassende Handbuch*. Galileo Press, 2007