

Web Service zum Überprüfen von digitalen Wasserzeichen

Bachelorarbeit
zur Erlangung des Grades eines Bachelor of Science
im Studiengang Informatik

vorgelegt von:
Vitali Keppel
vkeppel@uni-koblenz.de

Gutachter: Prof. Dr. Rüdiger Grimm,
Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich 4

Helge Hundacker,
Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich 4

Betreuer: Andreas Kasten,
Institut für Wirtschafts- und Verwaltungsinformatik, Fachbereich 4

Koblenz, im März 2010

Erklärung

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

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Koblenz, den

Abstract

Im Rahmen dieser Bachelorarbeit wird ein Webservice (SOA) entworfen und implementiert, mit dem eine Audiodatei auf Inhalt eines Wasserzeichens ferngesteuert überprüft werden kann.

Die Arbeit ist ein Teil des URM (Usage Rights Management) - Projekts, das von der Arbeitsgruppe Grimm an der Universität Koblenz-Landau entwickelt wird. Dabei handelt es sich um ein Konzept der Lizenzierung, bei dem die Nutzer über ihre Rechte an erworbenen digitalen Gütern informiert werden.

Diese Bachelorarbeit stellt am Beispiel von WAV-Dateien einen rudimentären Weg dar, wie im Rahmen von URM die Informationen, die bei der Lizenzerstellung verwendet werden, aus einem digitalen Medium extrahiert werden können.

In this bachelor thesis a Web Service (SOA) is designed and implemented. It can remotely detect a watermark in an audio file.

This thesis is a part of the URM (Usage Rights Management) – Project, which is developed by the WG Grimm at the University of Koblenz-Landau. It is a concept of a licensing process, in which users can inform themselves about their rights on acquired digital goods.

This bachelor thesis represents a rudimentary way of extracting information out of a digital medium e.g. a WAV-file.

Danksagung

An dieser Stelle möchte ich der gesamten Arbeitsgruppe Grimm danken, dass ich eine interessante Bachelorarbeit auf einem für mich neuen Gebiet schreiben durfte. Insbesondere möchte ich meinen Betreuern Prof. Dr. Grimm, Helge Hundacker und Andreas Kasten dafür danken, dass sie mich stets unterstützt und meine Fragen schnell und ausführlich beantwortet haben.

Des Weiteren möchte ich allen meinen Freunden danken, die mich im Laufe der Bachelorarbeit unterstützt haben. Insbesondere möchte ich mich bei Daniel Janke bedanken. Er brachte mir den Umgang mit Lyx bei, half bei einigen programmiertechnischen Problemen und hat meine Bachelorarbeit auf Fehler untersucht.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Übersicht	3
1.3	Aufbau der Arbeit	4
2	URM	5
2.1	Definition und kurzer Einblick	5
2.2	Lizenzerstellung	6
3	Grundlagen	9
3.1	Digitale Wasserzeichen	9
3.1.1	Eigenschaften und Anwendungsgebiete	9
3.1.2	Wasserzeichenverfahren	11
3.1.2.1	Least-Significant-Bit-Algorithmus (LSB)	12
3.1.2.2	Echo-Wasserzeichen	12
3.1.2.3	MPEG2-Scale-Factor-Algorithmus	13
3.2	Digitale Audioformate	13
3.2.1	Pulse-Code-Modulation	14
3.2.2	WAV-Dateiformat	15
3.2.2.1	Aufbau	16
3.2.2.2	WAV-Datei als Trägermedium	18
3.2.3	MPEG Layer 3 (MP3)	19
3.2.4	Advanced Audio Coding (AAC)	19
3.3	Web-Services	20
3.3.1	Service-orientierte Architektur (SOA)	20
3.3.2	Web-Services-Architektur	22
3.3.2.1	SOAP	23
3.3.2.2	Web-Services Description Language (WSDL)	24
3.3.2.3	Web-Services Inspection Language (UDDI)	25
3.3.3	Vor- und Nachteile von Web-Services	26

4	Watermark Embedding Tool	27
4.1	Entwurf	27
4.1.1	Anforderungen	28
4.1.2	Architektur	28
4.2	Implementation	30
4.2.1	Watermarker	30
4.2.1.1	WAV-Datei auf Gültigkeit überprüfen und auslesen	30
4.2.1.2	Wasserzeichen einbetten	31
4.2.1.3	WAV-Datei speichern	32
4.2.2	GUI	32
4.3	Bedienung	33
4.3.1	Start des Programms	33
4.3.2	Auswahl der Datei	34
4.3.3	Eingabe des Wasserzeicheninhalts	35
4.3.4	Einbetten und Speichern	35
4.3.5	Verbesserungsvorschläge	35
5	Verification-Web-Service	37
5.1	Entwurf	37
5.1.1	Anforderungen	37
5.1.2	Architektur	39
5.1.2.1	Web-Service	39
5.1.2.2	Client	39
5.1.2.3	Ablauf der Kommunikation	40
5.2	Implementierung	40
5.2.1	Web-Service	40
5.2.1.1	Verificator-Klasse	41
5.2.1.2	Web-Service-Klasse	43
5.2.1.3	Server-Klasse	43
5.2.2	Client	43
5.3	Beschreibung	44
5.3.1	Web-Service	45
5.3.2	Client	45
6	Zusammenfassung und Ausblick	47
A	Inhalt der CD	49
	Abbildungsverzeichnis	50
	Listingverzeichnis	51

Kapitel 1

Einleitung

1.1 Motivation

Der Vertrieb den digitalen Waren über das Internet hat in letzten Jahren stark zugenommen. Das virtuelle Geschäft bietet den Nutzern einen einfachen und kostengünstigen Weg, an die gewünschten Waren zu kommen. So konnte zum Beispiel Apple über sein in 2003 gestartetes Online-Musikportal „iTunes Music Store“ bis heute zehn Milliarden Songs verkaufen¹. Davon wurden sieben Milliarden nur in den letzten drei Jahren abgesetzt. Dabei steigt die Absatzmenge von Jahr zu Jahr stetig.

Doch der Handel mit den virtuellen Gütern hat auch seine Schattenseiten. Die illegalen Downloads sind eine neue Form der Kriminalität in der Internetwelt, unter der besonders die Musikindustrie leidet. Laut dem Digital Music Report [IFP09] von IFPI² kommen auf einen in 2009 legal erworbenen Song zwanzig illegale Kopien. Entsprechend hart sind die technischen und rechtlichen Maßnahmen der Musikanbieter und des Staates zum Schutz des Urheberrechts gegen Internetpiraterie.

Eins dieser technischen Mittel ist DRM³ - die Verfahren, mit denen die Nutzung und Verbreitung digitaler Medien kontrolliert werden kann. Mit DRM können Musikanbieter ihre Rechte durchsetzen und illegales Verhalten auf Seiten der Kunden unterbinden. Die DRM-Systeme haben jedoch einen entscheidenden Nachteil. Die Medien, die DRM-Schutz enthalten, sind für die Käufer unattraktiv, da die Nutzungsfreiheit der Kunden stark eingeschränkt ist. So wird beispielsweise die Installation von zusätzlichen Programmen gefordert, was das Abspielen auf externen Geräten wie beispielsweise mp3-Playern praktisch unmöglich macht. Deswegen wird vor allem in der Musikindustrie nach Alternativen gesucht.

So bieten iTunes und der Online-Musikshop von Amazon seit kurzem DRM-freie Songs an⁴. Auch viele andere Anbieter setzen in der letzten Zeit auf passive Schutzmechanismen wie zum Beispiel digitale Wasserzeichen⁵. Das Wasserzeichen ist eine zusätzliche Information, die in eine Datei eingebettet werden kann. So lassen sich leicht die Urheberinformationen oder Informationen über den Käufer mit einem Musikstück verknüpfen. Dadurch ist es möglich nachzuweisen, ob die Daten legal erworben sind oder nicht. Der User kann dabei über die erworbene Datei ohne Einschränkungen verfügen. Der amerikanische Internetanbieter AT&T plant beispielsweise ein Filtersystem auf

¹siehe auch <http://www.apple.com/de/itunes/10-billion-song-countdown/>; Letzter Abruf: 25.02.2010

²International Federation of the Phonographic Industry; <http://www.ifpi.org/>; Letzter Abruf: 26.02.2010

³Digital Rights Management. Siehe "Digital Rights Management – Business and Technology" [RTM02]

⁴Siehe <http://www.heise.de/newsticker/meldung/Amazon-startet-Online-Musikshop-mit-DRM-freien-MP3-Dateien-Update-178981.html>; Letzter Abruf: 02.03.2010

⁵Siehe [SIT08]

Wasserzeichenbasis einzuführen⁶, welches den gesamten Internetverkehr und besonders Verkehr in so genannten peer-to-peer-Tauschbörsen analysiert. Bei der Filterung soll das Wasserzeichen aus einer Datei ausgelesen werden und somit diese Datei auf Legalität überprüft werden.

Das Problem bei der Verwendung von Wasserzeichen ist, dass in ihm auch sensible kundenspezifische Daten wie beispielsweise Kontonummern gespeichert werden können. Zum Wasserzeicheninhalt sind keine eindeutigen Richtlinien vorgegeben. Diese sensiblen Daten können zum Beispiel von unbefugten Dritten missbraucht werden. Andererseits haben Kunden nach dem deutschen Datenschutzrecht ein Recht zu wissen, welche Informationen über sie in der jeweiligen Datei gespeichert sind. Dies ist jedoch aufgrund der häufig verschlüsselt vorliegenden Wasserzeichen schwierig.

Problematisch ist auch die technische Realisierung der Wasserzeichenüberprüfung. Die oben erwähnte Filterung des Internetverkehrs von AT&T könnte die Netze belasten oder sogar blockieren. Außerdem gibt es kein einheitliches System zum Lesen und Verwalten von Wasserzeichen, nicht zuletzt weil es keine eindeutigen Richtlinien zum Wasserzeichenaufbau beziehungsweise für Wasserzeichenalgorithmen gibt, so dass die verschiedenen Musikanbieter nur jeweils ihre eigene Wasserzeichen lesen können. Es wird aber bereits an diesen Problemen gearbeitet. So bietet zum Beispiel RIAA (Recording Industry Association of America) eine eigene 108 Bit große Wasserzeichenspezifikation⁷ an, die ein neuer Standard werden soll.

Neben den genannten technischen Maßnahmen setzten die Anbieter digitaler Medien auf die Kooperation mit Internet Providern. Das Right Protection System von IFPI zum Urheberrechtsschutz durchsucht das Internet auf Seiten, welche digitale Medien ohne Erlaubnis der Rechteinhaber anbieten. Diese Seiten werden dann mit Hilfe der Internetanbieter blockiert. Allerdings können die Webseiten, deren Server sich außerhalb des gesetzlichen Einflusses von IFPI wie beispielsweise in Osteuropa befinden, nicht blockiert werden. Des Weiteren wurde in Frankreich das so genannte Hadopi-Gesetz⁸ verabschiedet, welches 2010 in Kraft getreten ist. Nach diesem Gesetz bekommt der Urheberrechtsverletzer zwei Verwarnungen. Wenn er danach weiter seine illegale Tätigkeit ausübt, droht ihm bis zu einem Jahr Internetsperre und eine hohe Geldstrafe. Zur Zeit wird im EU-Parlament diskutiert, ob ein ähnliches Gesetz nach dem sogenannten „Three strikes“-Modell in weiteren EU-Staaten eingeführt werden soll. Obwohl solche Gesetze auf scharfe Kritik wegen Verletzungen des Datenschutzes und der Kommunikationsfreiheit stoßen, könnten sie in der Praxis als Abschreckung fungieren. So würden laut den Umfragen von IFPI, die in dem Digital Music Report 2009 veröffentlicht wurden, 90 Prozent der Befragten illegales Herunterladen einstellen, wenn sie entsprechende Warnhinweise von dem Internetanbieter bekommen hätten.

Alle oben genannten Maßnahmen haben ihre Nachteile. Meistens haben die Kunden nur eingeschränkte Freiheit an den erworbenen Medien. Außerdem lässt es sich auch nicht eindeutig bestimmen, ob der Internetzugangsberechtigter tatsächlich derjenige ist, der illegale Tätigkeiten ausübt. Obwohl die Urheberschutzmaßnahmen unausweichlich sind, mangelt es einfach zurzeit an einem Informationssystem, welches die ehrlichen Kunden über ihre Rechte an erworbenen digitalen Medien informiert. Heutzutage gibt es hunderte Quellen, wo Musikstücke erworben werden können. Einige davon sind Online-Shops, digitale Radios, der Umtausch mit Freunden und das CD-Rippen. Als Medienkonsument kann man da schnell den Überblick über eigene Rechte verlieren. Eine weitere Quelle sind peer-to-peer-Tauschbörsen. In diesem konkreten Fall ist es schwer für die P2P-Nutzer eine illegale Datei von legalem Content zu unterscheiden. Es kann passieren, dass auch ehrliche Nutzer unabsichtlich illegale Songs herunterladen.

⁶<http://www.heise.de/newsticker/meldung/AT-T-erlaeutert-geplanten-Filteransatz-gegen-illegale-Downloads-176996.html>; Letzter Abruf: 02.03.2010

⁷Siehe "Standard Watermark Payload Configuration" [RIA09]

⁸<http://www.heise.de/newsticker/meldung/Frankreich-Internetsperre-fuer-Urheberrechtsverletzer-gebilligt-837138.html>; Letzter Abruf: 02.03.2010

Die Arbeitsgruppe Grimm von der Universität Koblenz-Landau hat sich Gedanken darüber gemacht, wie den ehrlichen Medienkonsumenten bei der Verwaltung ihrer digitalen Rechte geholfen werden kann. Das von der Gruppe konzipierte System URM (Usage Rights Management)⁹ soll die oben erwähnten Urheberschutzmaßnahmen nicht ersetzen, sondern diese sinnvoll ergänzen. Das System ist eher dazu da, vor allem die ehrlichen Nutzer vor dem unabsichtlichen illegalen Handeln zu warnen und vor Missbrauch im Internet zu schützen. Wie das genau funktioniert, wird in dem nächsten Kapitel beschrieben.

Diese Bachelorarbeit beschäftigt sich mit der technischen Umsetzung von einem Teil des URM-Konzepts. Das Ziel der Arbeit ist, einen Web-Service-Prototypen zu implementieren, mit welchem Musikdateien auf Inhalt eines Wasserzeichens überprüft werden können. Diese Funktionalität erlaubt dem URM-System Wasserzeicheninhalte aus einer wasserzeichenbehafteten Datei für die Lizenzerstellung, die in Kapitel 2 beschrieben wird, ferngesteuert zu extrahieren.

1.2 Übersicht

Ein grober Überblick über die Arbeit ist in der Abbildung 1.1 zu sehen. Die farblich hervorgehobenen Teile des Diagramms werden in dieser Bachelorarbeit realisiert. Im Grunde funktioniert das in dieser Arbeit implementierte Programm wie folgt: Der Benutzer wählt über die GUI¹⁰ des Clients eine auf seiner Festplatte gespeicherte Datei. Der Client schickt dann diese Datei über ein Netzwerk an den Web-Service. Er extrahiert das enthaltene Wasserzeichen und schickt es an den Client zurück. Schließlich wird das Wasserzeichen dem Benutzer durch die GUI angezeigt.

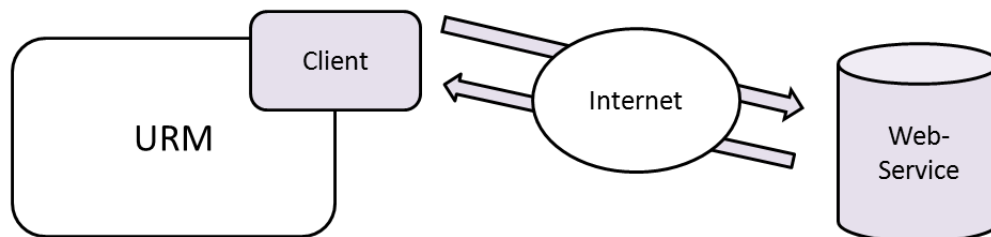


Abbildung 1.1: Systementwurf

In der fertigen Version des in dieser Bachelorarbeit implementierten Web-Service-Programms wird das URM-System mit einem Web-Service über Client-Komponente kommunizieren können. In dieser Arbeit werden jedoch nur prototypische Funktionen des Systems implementiert. Das bedeutet, dass die Client-Komponente vorerst kein Teil des URM-Systems, sondern ein eigenständiges Programm mit einer GUI ist. Das Integrieren des Client-Programms in das URM-System ist nicht die Aufgabe dieser Bachelorarbeit und kann in den nachfolgenden Arbeiten realisiert werden. Eine weitere Designentscheidung ist, die Umsetzung auf das WAV-Audioformat als Trägermedium und den LSB(Least Significant Bit)-Algorithmus als Wasserzeichenalgorithmus¹¹ zu beschränken. In einer Weiterentwicklung kann die Unterstützung anderer Medienformate und Wasserzeichenalgorithmen in das System ergänzt werden.

⁹Siehe "URM - Usage Rights Management" [HPG09]

¹⁰Graphical User Interface

¹¹detaillierte Beschreibung dieser Komponenten in Kapitel 3

1.3 Aufbau der Arbeit

Zunächst wird in Kapitel 2 das URM-System beschrieben und der Zusammenhang dieser Bachelorarbeit mit URM erläutert. Danach werden in Kapitel 3 die Hintergrundkenntnisse aufgeführt, die bei Erstellung dieser Arbeit erworben werden mussten. Kapitel 4 beschäftigt sich mit dem Watermark Embedding Tool, welches für das Einbetten eines Wasserzeichens in eine Datei zuständig ist. Schließlich wird der Entwurf und die Realisierung des eigentlichen Web-Services und des dazugehörigen Clients in Kapitel 5 beschrieben. In Kapitel 6 werden dann die wichtigsten Aspekte der Arbeit zusammengefasst und abschließend wird ein Ausblick gegeben.

Kapitel 2

URM

Da diese Bachelorarbeit ein Teil des in der Einleitung vorgestellten URM-Projekts ist, wird das Projekt in dem folgenden Kapitel ausführlicher beschrieben. Dabei wird besonders auf die Lizenz-erstellung von URM-System eingegangen, damit der Zusammenhang mit dieser Bachelorarbeit deutlicher wird. Dieses Kapitel beruht auf der Ausarbeitung „URM – Usage Rights Management“ [HPG09].

2.1 Definition und kurzer Einblick

Mit dem URM-Projekt verfolgt die Arbeitsgruppe Grimm ein neues Konzept im Bereich der digitalen Rechteverwaltung.

Das URM (Usage Rights Management) ist ein DRM-Instrument, da es typische Merkmale aufweist, wie zum Beispiel Kontroll- und Verwaltungsfunktionen an digitalen Rechten. Jedoch anders als bei den geschlossenen DRM-Systemen wird die Nutzung und Verbreitung digitaler Medien nicht von den Anbietern sondern von den Benutzern selbst kontrolliert.

Das Ziel des Projekts ist, den Nutzern über eigene Rechte an digitalen Waren aufzuklären und bei der Verwaltung dieser Rechte zu helfen. Das Konzept richtet sich dabei an die Medienkonsumenten, die sich fair und legal verhalten wollen. Deswegen verzichtet man zu Gunsten der Benutzerfreundlichkeit auf klassische Kopierschutzmechanismen. Stattdessen werden die Nutzer mit Hilfe des so genannten Ampelfarben-Systems über ihre Rechte informiert und eventuell auf ihr illegales Verhalten aufmerksam gemacht.

In der Abbildung 2.1 ist ein Prototyp der URM-Mediathek dargestellt. In so einer Mediathek kann der User seine Rechte an den auf seiner Festplatte gespeicherten Musikstücken verwalten. Die Rechte befinden sich in so genannten Lizenzen und werden als Ampelfarben in der Spalte „Rechte“ visualisiert. So bedeutet zum Beispiel „grün“, dass das entsprechende Musikstück abgespielt werden kann. „Rot“ bedeutet dagegen, dass der Song illegal erworben wurde und dass der Benutzer kein Recht hat ihn abzuspielen.

Künstler	Album	Nr.	Titel	Jahr	Rechte	UID (SHA1-Hash)
Bloodspot	Taste The Promo	03/09	In Honesty		●○○●	66bb139a84aaa24a525b7df839dca44928e4bb87
Demons & Wizards	Demons & Wizards	04/13	Fiddler on the ...	2000	●○○○	c52d3d886e3bc3d675a22adeb4fa832b054ad8ab
Dog Eat Dog	Play Games	04/11	Rocky	1996	●○○●	9014e6a21dab86462a2b4ee94426b28d2910af74
Tool	/Enima	05/15	Forty Six & 2	1996	○●○○	71edf2707ec8b29cee6250a78c8971d6c2398863
Grimfist	Ghouls Of Grandeur	05/10	No Compromise	2003	○○●○	7d97a40c9fa71e18e1a5f04a746d20c93166b6e6
Bloodspot	Taste The Promo	05/09	Unborn	2007	○●○○	285fd27d5f0f8a4db4a51b3f408189c3ccf59d00
Bloodspot	Taste The Promo	06/09	Instinct	2007	●○○●	9b18db17ae4cf7c3dba371f832107c48fedc1b39
Forsth	Winterfrost	06/09	Winterfrost		●○○●	02294c80cb6d73e14935b2e0d146f29d3c68def9
Blind Guardian	Imaginations From The ...	07/11	Bright Eyes	1995	○●○○	ffed7e2e68d98cb3d8176cd3ea9c1ea0a452e0a8
Graveworm	Scourge Of Malice	07/10	Fear Of The D...	2001	○●○○	94c1e9d7e0584de6dffcc0bd52e7b4c5602da153
Bloodspot	Taste The Promo	07/09	Taste The Can...	2007	○●○○	95c1d25a68356e02978238f5d9b0bcafc9a75525
Blind Guardian	A Night At The Opera	07/10	The Soulforged	2002	●○○○	7917bf5ffa3fb00056326b6bed7dc8df0047d5df

Abbildung 2.1: URM - Mediathek. Quelle: [JAH10]

Neben der vorgestellten Mediathek ist das URM-System als Plug-In für die Mediaplayer wie z.B. Winamp oder für Peer-to-Peer (P2P)-Programme realisierbar. Besonders bei P2P-Netzen ist ein Aufklärungsinstrument wie URM sinnvoll, denn häufig ist hier der Ursprung der Daten illegal. Den ehrlichen Usern bietet das System an, anstatt die Datei gleich herunterzuladen und sich eventuell strafbar zu machen, zuerst die Lizenz von der entsprechenden Datei herunterzuladen. Anhand dieser Lizenz entscheidet das System, ob das jeweilige Musikstück selbst heruntergeladen wird oder nicht. Dabei ist URM besonders für die P2P-Nutzer interessant, die ihre Dateien in Tauschbörsen anbieten. Das System sorgt von vorneherein dafür, dass die illegalen Daten in P2P-Netzen gar nicht erst angeboten werden können. Das Programm „Digital File Check“¹ von IFPI hat eine ähnliche Funktion. Dieses untersucht den Rechner auf P2P-Clients und blockiert sie. Der Nachteil dabei ist, dass das Tausch-Programm gar nicht mehr benutzt werden kann. Im Vergleich dazu bietet URM eine elegantere Lösung.

Das URM-Projekt befindet sich zurzeit in einer frühen Entwicklungsphase. Deswegen sind noch viele Entwicklungswege offen. So wäre zum Beispiel das Ergänzen des URM-Systems um zusätzliche DRM-Maßnahmen möglich. Des Weiteren ist auch geplant, die Userrechte nicht nur zu visualisieren, sondern auch bei den illegal erworbenen Medien zu den entsprechenden Shops weiterzuleiten, damit diese legal erworben werden können. Momentan arbeiten die Entwickler an der Realisierung der Prototypen für mp3-Dateien. In der Zukunft wird aber auch die Arbeit mit anderen digitalen Medien wie zum Beispiel digitale Bilder, Videos, Software und E-Books möglich sein. Zu dem heutigen Zeitpunkt steht jedoch schon fest, dass URM benutzerfreundlich und plattformunabhängig bleiben soll.

Nachdem man einen groben Überblick über URM bekommen konnte, wird in dem nächsten Unterabschnitt nun die Lizenzerstellung beschrieben.

2.2 Lizenzerstellung

Die Rechte der Nutzer werden in den oben erwähnten Lizenzen gespeichert. Die Lizenzen selbst werden mit dem XML-basierten Open-Source-Standard ODRL (Open Digital Rights Language)² beschrieben und bestehen aus folgenden Teilen:

asset

¹siehe http://www.musikindustrie.de/digital_file_check/; Letzter Abruf: 26.02.2010

²detaillierte Informationen in "Open Digital Rights Language (ODRL)" [REN02]

In diesem Abschnitt der Lizenz wird beschrieben, um welches Objekt (Datei) es sich handelt. Hier werden Informationen wie der Name der Datei, der Speicherort auf der Festplatte, Identifikationsnummer als SHA³-Hashwert und eventuell der physikalische Ort der Original-CD gespeichert.

permission

Der Abschnitt beschreibt die eigentlichen Rechte an dem Objekt, d.h. ob die Datei abgespielt sowie ob und wie oft sie kopiert werden darf. Die Verweise auf die entsprechenden Rechtsgrundlagen können optional enthalten sein.

party

In dem party-Teil befinden sich die Informationen über den Rechteinhaber wie zum Beispiel Name, Email-Adresse oder Identifikationsnummer.

Optional kann eine Lizenz das Datum des Rippens enthalten, wenn das Musikstück von einer CD gerippt wurde. Im Allgemeinen ist es also möglich, aus so einer Lizenz Informationen zu dem Rechteinhaber, zu der Datei selbst und zu den dazugehörigen Rechten zu entnehmen.

Wie in dem vorherigen Kapitel erwähnt wurde, kann der Nutzer im URM-System seine Rechte an digitalen Medien verwalten. Anhand dieser Rechte, die den tatsächlichen Rechten und Kopierschutzbestimmungen der Rechteinhaber entsprechen müssen, kann er die dazugehörigen Lizenzen selbst erstellt. Beim CD-Rippen zum Beispiel weiß der User, ob er die Songs abspielen oder weitergeben darf, weil er beim Kauf einer CD die gesetzliche Grundlage kennt. Problematisch wird es aber, wenn man mit Dateien zu tun hat, deren Herkunft nicht bekannt ist. Das können alte Daten sein oder die Daten, die aus P2P-Netzen oder von Radioaufnahmen stammen. Dabei kann der Nutzer seine Rechte nicht mehr eindeutig bestimmen. In diesem Fall soll URM helfen.

Einige Anbieter binden in die ID3-Tags⁴ der gekauften mp3-Dateien zusätzliche Informationen über den Käufer oder über den Rechteinhaber ein. Das URM-System kann diese Informationen lokal extrahieren und sie an den Benutzer weiterreichen oder anhand von Ihnen die Lizenzen eigenständig generieren. Jedoch sind die ID3-Tags leicht manipulierbar und daher nur begrenzt vertrauenswürdig.

Eine weitere Möglichkeit, den Ursprung eines Mediafiles zu bestimmen, sind Web-Services⁵. Der Hashwert einer Datei kann zu so einem Web-Service hochgeladen werden. Seinerseits benutzt der Service eine Liste mit Hashwerten bekannter Dateien. Wenn die hochgeladene Datei ihm bekannt ist, sendet er entsprechende Dateiinformationen zurück.

Letztendlich ist die Überprüfung des Wasserzeichens eine weitere Strategie zu Bestimmung der Herkunft einer Datei. Ein Mediafile mit Wasserzeichen kann Informationen über den Rechteinhaber, den Käufer und Benutzerrechte enthalten. Um an diese Informationen zu kommen, müssen sie aus der Datei extrahiert werden.

Das Extrahieren kann lokal oder Web-Service-orientiert durchgeführt werden. Dabei stellt sich die lokale Variante als in der Praxis kaum realisierbar dar. Häufig verschlüsseln die Musikanbieter ihre Wasserzeichen, um Manipulationen daran zu verhindern. Die Schlüssel zum Entschlüsseln werden dabei aus Sicherheitsgründen nicht an die Musikkonsumenten weitergegeben. Das lokale Extrahieren ist in diesem Falle unmöglich, obwohl es bei unverschlüsselten Wasserzeichen noch realisierbar wäre. Des Weiteren verfolgt das URM-Konzept einen plattform- und anwendungsnabhangigen Ansatz. Um eine lokale Variante zu realisieren, müsste man für jede Plattform und

³Secure Hash Algorithm. Siehe "Secure Hash Standard" [FIP95]

⁴siehe auch: "The Audience is informed"; <http://www.id3.org/>; Letzter Abruf: 25.02.2010

⁵siehe Kapitel 3

jedes Laufzeitsystem eine eigene Implementation erzeugt werden. Dies würde zu einem hohen Programmieraufwand führen.

Als Alternative zu der lokalen Überprüfung des Wasserzeichens können Web-Services betrachtet werden. In diesem Zusammenhang ist die Umsetzung eines Web-Services auch das Ziel dieser Arbeit. Das Extrahieren findet dabei nicht lokal statt, sondern auf einem entfernten Rechner in einem Netzwerk. Da Web-Services auf offene Standards setzen, kann so eine problemlose Kommunikation zwischen unterschiedlichen Plattformen und Systemen bei geringem Aufwand gewährleistet werden. Der Ansatz der Web-Services, der im Unterabschnitt 3.3 des Kapitels 3 detaillierter beschrieben wird, erlaubt auch eine Kommunikation zwischen Web-Services selbst. In unserem konkreten Fall bedeutet das, dass der Web-Service zum Überprüfen des Wasserzeichens die Verbindung zu dem Web-Service des Musikanbieters aufnehmen kann, der die Schlüssel zu den verschlüsselten Wasserzeichen verwaltet. So kann auch die verschlüsselte Information aus dem Wasserzeichen extrahiert werden, ohne dass der Musikanbieter den Schlüssel preisgeben muss. Der Einsatz solcher Web-Services erweist sich als vorteilhaft gegenüber der lokalen Umsetzung und ist Kernpunkt dieser Bachelorarbeit.

Kapitel 3

Grundlagen

In dem nachfolgenden Kapitel wird das Grundwissen zusammengefasst, welches für die Umsetzung von den in dieser Bachelorarbeit implementierten Programmen Watermark Embedding Tool und Verification Web-Service benötigt wurde.

Als erstes wird auf den Begriff, die Eigenschaften und Anwendungsgebiete von digitalen Wasserzeichen im Abschnitt 3.1 eingegangen. Der in dieser Arbeit verwendete Least-Significant-Bit-Wasserzeichenalgorithmus wird in dem Unterabschnitt 3.1.1 erläutert. Danach werden kurz das Echo-Wasserzeichen und das MPEG2-Scale-Factor-Wasserzeichen beschrieben.

Des Weiteren wird im Abschnitt 3.2 ein Überblick über die heute relevanten Audioformate gegeben. Dabei wird das WAV-Format im Gegensatz zu den Alternativen detaillierter beschrieben, da es in dieser Arbeit als Wasserzeichenträgermedium fungiert. Als Grundlage für den Aufbau des WAV-Formates wird die Umwandlung analoger Schallwellen in digitale Form erklärt.

Schließlich werden die theoretischen Grundlagen von Web-Services im Abschnitt 3.3 als Vorbereitung zur Implementierung des Verification Web-Service-Programms kurz beschrieben.

3.1 Digitale Wasserzeichen

Digitale Wasserzeichen sind passive DRM-Verfahren, mit welchen zusätzliche Informationen in ein digitales Medium eingebettet werden können. Obwohl bei den Wasserzeichenverfahren digitale Medien wie zum Beispiel Video, Audio, Bilder oder sogar Textdokumente als Trägermedium eingesetzt werden können, werden in diesem Abschnitt hauptsächlich digitale Audiowasserzeichen beschrieben¹.

3.1.1 Eigenschaften und Anwendungsgebiete

Prinzipiell unterscheidet man je nach Einsatzgebiet zwischen *wahrnehmbaren* und *nicht wahrnehmbaren Wasserzeichen*. Die *wahrnehmbaren Wasserzeichen* markieren ein Medium für den Anwender sichtbar oder hörbar. So können zum Beispiel Firmenlogos in Bild- und Videomaterial oder hörbare Muster in Audiodaten als wahrnehmbare Wasserzeichen eingebettet werden. Interessanter sind aber die *nicht wahrnehmbaren digitalen Wasserzeichen*, die in einem digitalen Medium mit Hilfe eines Einbettungsalgorithmus versteckt werden. Der Vorteil der nicht wahrnehmbaren Wasserzeichen ist, dass die Nutzer im Gegensatz zu aktiven DRM-Kopierschutzmechanismen das wasserzeichenbehaftete Medium ohne Einschränkungen abspielen oder kopieren können. Im Allgemeinen

¹Das gesamte Kapitel ist angelehnt an [Ach03, DS05, SIT08, Pus03, Hen06, HUN, Dit00]

können die nicht wahrnehmbaren Wasserzeichen folgende Eigenschaften besitzen, die gleichzeitig deren Anforderungen sind:

Als **Robustheit und Fragilität** wird die Widerstandfähigkeit eines Wasserzeichens gegenüber Veränderungen des Datenmaterials bezeichnet. Ein robustes Wasserzeichen soll auch nach gewissen auf das Trägermedium angewandten Operationen wie Kompressionen oder Analog-Digital-Umwandlung noch vorhanden sein. Im Gegensatz dazu wird ein fragiles Wasserzeichen durch Manipulationen zerstört. Häufig wird diese Eigenschaft in Zusammenhang mit der Veränderungsmethode gesetzt, da ein Wasserzeichen wie zum Beispiel das *MPEG2-Scale-Factor-Wasserzeichen*, das im nächsten Unterabschnitt vorgestellt wird, robust gegen Komprimierung und gleichzeitig fragil gegen Audioformatänderung sein kann.

Kapazität besagt, wie viel Wasserzeicheninhalt in ein Medium mit einem Algorithmus theoretisch eingebettet werden kann. Häufig ist die Kapazität eines Wasserzeichens aber nicht nur von dem Wasserzeichenalgorithmus abhängig, sondern auch von dem Trägermedium selbst.

Sicherheit charakterisiert im Gegensatz zur *Robustheit* die Resistenzfähigkeit eines Wasserzeichens gegenüber gezielten Angriffen. Die Wasserzeicheninhalte werden normalerweise verschlüsselt, um sie vor Angreifern zu schützen. Ein sicheres Wasserzeichen soll ohne Schlüssel nicht entschlüsselbar sein.

Mit **Transparenz** wird angegeben, ob ein eingebettetes Wasserzeichen vom menschlichen Wahrnehmungssystem optisch oder akustisch wahrnehmbar ist. Ein *nicht wahrnehmbares Wasserzeichen* muss transparent sein.

Abhängig von ihren Eigenschaften werden Wasserzeichen in unterschiedlichen Anwendungsgebieten eingesetzt. Die wichtigsten Anwendungsgebiete sind *Urheberschutz* und *Kundenidentifikation* sowie *Integritätsnachweis*.

Urheberschutz und Kundenidentifikation

In diesem Fall werden die Informationen über den Autor, Rechteinhaber oder sonstige Copyright-Bestimmungen als Wasserzeicheninhalt in ein Medium eingebettet. Bei individuellen Wasserzeichen werden kundenbezogene Informationen wie zum Beispiel Kundennummern oder Transaktionsnummern verwendet. Das Ziel ist dabei, die Rechteinhaber zu schützen. Diese Wasserzeicheninformationen erlauben es nachzuweisen, welcher Herkunft die markierte Datei ist, ob sie unberechtigt kopiert wurde und eventuell von wem diese unbefugte Kopie gemacht wurde. Prinzipiell werden dabei die *robusten Wasserzeichen* verwendet. Die eingebetteten urheber- und kundenspezifischen Daten sind auch für das URM-System interessant. Das System kann anhand dieser Informationen die Herkunft der Datei ermitteln und eine entsprechende Lizenz erstellen, wie es im Kapitel 2 beschrieben wurde.

Integritätsnachweis

Die *Integritätswasserzeichen* werden eingebettet, um die Manipulationen an einem Medium festzustellen. Hierbei werden häufig die *fragilen Wasserzeichen* eingesetzt. Durch das Verändern oder Komprimieren des wasserzeichenbehafteten Mediums wird ein *fragiles Wasserzeichen* zerstört. Somit kann nachgewiesen werden, ob eine Datei manipuliert wurde oder nicht.

Es gibt heutzutage kein universelles Wasserzeichenverfahren, welches alle oben genannten Anforderungen hundertprozentig erfüllt und bekannte Angriffe überstehen kann. Die Eigenschaften eines Wasserzeichens sind voneinander abhängig. So steigt zum Beispiel mit der sinkenden Robustheit und Transparenz die Kapazität [Ach03]. Die meisten Wasserzeichenalgorithmen sind anwendungsspezifisch und von der Art des Mediums abhängig. In nächstem Unterabschnitt werden drei Audiowasserzeichen vorgestellt. Der *Least-Significant-Bit-Algorithmus* wird in dieser Arbeit in den Programmen *Watermark Embedding Tool* und *Verification Web-Service* eingesetzt.

3.1.2 Wasserzeichenverfahren

Jeder Wasserzeichenalgorithmus besteht aus einem *Einbettungs-* und einem *Ausleseprozess*. Bei dem *Einbettungsprozess* werden die Wasserzeicheninhalte in ein Medium eingebettet. Die Art und Weise der Einbettung ist von dem jeweiligen Verfahren abhängig. Im Groben unterscheidet man zwischen zwei Arten von Verfahren. Bei einigen Algorithmen werden die weniger relevanten Teile der unkomprimierten Nutzdaten eines Mediums mit dem Wasserzeicheninhalt ersetzt. So wird zum Beispiel auch bei dem *Least-Significant-Bit-Algorithmus* vorgegangen. Andere Algorithmen basieren auf psychoakustischen Erkenntnissen, die im Unterabschnitt 3.2.3 genauer beschrieben sind. Dabei werden anhand dieser Erkenntnisse die Frequenzen einer Audiodatei, die vom menschlichen Gehör nicht wahrgenommen werden, extrahiert und verändert. Diese Vorgehensweise wird unter anderem bei dem *Echo-Wasserzeichenalgorithmus*² verwendet.

Ein Wasserzeichen wird je nach Eigenschaft des Trägermediums spezifisch eingebettet. Grundsätzlich gibt es zwei Arten, ein Wasserzeichen in eine Datei einzubetten: an einer bestimmten Position oder über die gesamte Datei verteilt. So wird beispielsweise ein *fragiles Wasserzeichen* über die gesamte Datei zum *Integritätschutz* eingesetzt. Dagegen können die *Urheber-* oder *Kundeninformationen* an bestimmten Markierungsstellen eingebettet werden. Die Markierungsstellen können zufällig gewählt und dabei an einen geheimen Schlüssel gebunden werden oder fest definiert sein. Dabei bietet die zufällige Markierung eine höhere Sicherheit, denn ein Angreifer muss das Wasserzeichen erst finden, bevor er den eigentlichen Angriff durchführen kann. Die Verschleierung der Markierungsstellen reicht aber im Normalfall nicht aus, um ein Wasserzeichen zu schützen. Deswegen wird häufig der Wasserzeicheninhalt selbst verschlüsselt. Um den Wasserzeicheninhalt auszulesen, werden das Markierungsmuster und der geheime Schlüssel benötigt. Häufig wird das Markierungsmuster anhand des geheimen Schlüssels pseudozufällig generiert. Wenn das der Fall ist, wird beim Ausleseprozess zunächst die Position der Wasserzeichendaten anhand des Schlüssels ermittelt. Anderenfalls müssen die Markierungspositionen bei dem Auslesen bekannt sein. Des Weiteren werden je nach Verfahren und Verschlüsselung die Wasserzeicheninhalte extrahiert und entschlüsselt.

Bei dem *Ausleseprozess* wird im Allgemeinen zwischen *blinden* und *nicht blinden Verfahren* unterschieden. Die *nicht blinden Verfahren* benötigen beim Auslesen eine Originaldatei ohne Wasserzeichen. Die *blinden Verfahren* sind komplexer und können die Daten ohne Original nur anhand des Wasserzeichenalgorithmus extrahieren [Ach03, DS05, Dit00]. Für diese Arbeit und für das URM-System sind besonders die *blinden Verfahren* interessant, da das URM-System normalerweise keinen Zugang zu Originaldateien ohne Wasserzeichen hat.

In den folgenden Unterunterabschnitten werden drei Wasserzeichen vorgestellt. Der *Least-Significant-Bit-Algorithmus* wurde bei der Erstellung dieser Bachelorarbeit implementiert. Die Echo- und MPEG2-Scale-Factor-Algorithmen können als Alternativen zum LSB-Algorithmus betrachtet werden.

²Siehe Unterunterabschnitt 3.1.2.2 und [Dit00]

3.1.2.1 Least-Significant-Bit-Algorithmus (LSB)

Bei dem *LSB-Wasserzeichenalgorithmus* wird der Wasserzeicheninhalt in die weniger wichtigen Mediumbereiche eingebettet. Das bedeutet, dass die einzelnen Bits eines Trägermediums ersetzt werden. Dabei soll das Ersetzen dieser Bits die Qualität des Mediums nicht wahrnehmbar beeinflussen. Im Prinzip werden die Bits mit niedrigerer Bitwertigkeit die sogenannten *Least Significant Bits* der Nutzdaten einer Datei manipuliert, da ihre Veränderungen für den Menschen nicht bedeutend wahrnehmbar sind. Das *LSB-Verfahren* ist deswegen stark von der Struktur des Trägermediums abhängig. So wird zum Beispiel das Wasserzeichen bei Tonmaterial in die unkomprimierten PCM-Rohdaten³ eingebettet. Die Einbettung ist hier von der Samplegröße abhängig⁴. Die Veränderung des niedrigstelligen Bits in den Samples kann vom menschlichen Gehör nicht wahrgenommen werden. Im Gegensatz dazu werden bei einem RGB-Bild die Bits der Blauwerte manipuliert⁵, da sie von menschlichem Auge am geringsten wahrgenommen werden.

Der *LSB-Algorithmus* ist leicht zu implementieren und bietet eine hohe Datenrate. Jedoch handelt es sich hier um ein verlustbehaftetes Wasserzeichenverfahren. Das heißt, dass die ersetzten Daten nicht mehr wieder hergestellt werden können. Das *Reversible Data Hiding*⁶ von der Universität Rochester macht aber die Wiederherstellung der Originaldaten möglich. Bei der Technik wird die Information über die Veränderungen am Original in einer komprimierten Form in den Wasserzeicheninhalt eingebunden. Dies reduziert zwar die Wasserzeichenkapazität, ermöglicht aber eine gewisse Originaltreue. Ein weiterer entscheidender Nachteil ist die Fragilität des *LSB-Wasserzeichens* gegenüber Kompressionen und im Allgemeinen gegenüber jeder Art von Manipulationen am Trägermedium. Ein *LSB-Wasserzeichen* kann gegenüber Manipulationen zwar robuster gemacht werden, indem es an verschiedenen Stellen in der Datei mehrfach eingebettet wird. Aber es wird dann immer noch bei einer Digital-Analog-Umwandlung zerstört. Die Fragilität des *LSB-Wasserzeichens* kann sich aber als vorteilhaft erweisen, wenn es zum Integritätschutz eingesetzt wird. [Pus03, Hen06]

Das *LSB-Wasserzeichen* ist wegen seiner Fragilität zu dem Urheberschutz und der Kundenidentifikation eher ungeeignet und daher für das URM-System irrelevant. Es erfüllt jedoch den Zweck der prototypischen Implementierung dieser Arbeit.

3.1.2.2 Echo-Wasserzeichen⁷

Der *Echo-Wasserzeichenalgorithmus* basiert auf psycho-akustischen Erkenntnissen. So kann zum Beispiel das menschliche Gehör die leisen Töne, die nach einem lauten Ton kommen, nicht wahrnehmen. Ein leiser Ton, der vor dem lauten Ton aufgenommen wird und einen gewissen Schwellenwert nicht übersteigt, kann ebenfalls von einem Menschen nicht wahrgenommen werden. Bei dem *Echo-Algorithmus* wird in der Datei nach solchen Stellen gesucht und, falls sie gefunden werden, werden diese mit Wasserzeichen markiert.

Das *Echo-Wasserzeichen* kann sowohl in die unkomprimierten als auch in die komprimierten Audiodaten eingebettet werden. Dieses Wasserzeichen ist robuster als *LSB-Wasserzeichen*, da es zum Beispiel bei Analog-Digital-Umwandlung erhalten bleibt. Es kann aber durch bestimmte Kompressionsverfahren, welche beispielsweise bei MP3-Codierung verwendet werden und ebenfalls auf Psychoakustik setzen, zerstört werden, da die nicht wahrnehmbaren Töne bei dieser Kompression entfernt werden.

³Diese Daten werden durch die Pulse-Code-Modulation gewonnen. Siehe Unterabschnitt 3.2.1

⁴Ein Sample ist ein Abtastwert. Siehe Unterabschnitt 3.2.2

⁵Siehe [Hen06]

⁶Siehe [Pus03]

⁷Die Beschreibung der zwei folgenden Verfahren ist an [Dit00] angelehnt.

3.1.2.3 MPEG2-Scale-Factor-Algorithmus

Der *Scale-Factor-Algorithmus* ist ein blindes Verfahren, welches spezifisch für das Einbetten in die MPEG-2-komprimierten Audiodaten entwickelt wurde. Somit kann dieses Wasserzeichenverfahren auch beim Advanced Audio Coding (ACC)-Format, das im nächsten Abschnitt kurz beschrieben wird, eingesetzt werden.

Bei dem Verfahren werden zuerst alle MPEG-2-Frames⁸ zu gleich großen Gruppen mit mehreren Frames zusammengefasst. In jede Gruppe wird ein Bit des Wasserzeichens nach einem bestimmten Muster eingebettet. Die einzelnen Audiowerte der Gruppe werden entsprechend dem Muster geändert. Zum Beispiel könnte das Muster für eine Eins der Vektor $(22, 10, -3)$ und für eine Null $(10, 10, 10)$. Die Muster können auch andere Vektoren sein. Für jede Gruppe wird dann derselbe Vektor verwendet. Wenn nun eine Eins in die Gruppe eingebettet werden soll, wird zunächst der erste Audiowert zum Beispiel 13 genommen. Danach folgt die Differenzenrechnung. Das heißt, dass nach der Einbettung die Werte 35, 33 und 10 in der Gruppe vorkommen müssen. Wenn diese Werte schon vorher da sind und keine Eins in die Gruppe codiert werden soll, so werden die Werte um $+ - 1$ verändert, damit nicht fälschlicherweise eine Eins erkannt wird. Falls das ausgewählte Muster eingesetzt werden kann, werden die am besten passenden Werte, die am wenigsten verändert werden müssten, durch die Werte 35, 33, 10 ersetzt. Um mögliche Fehler beim Auslesen zu vermeiden und das Verfahren robuster gegenüber Manipulationen zu machen, kann eine Eins oder eine Null in eine Gruppe mehrmals eingebettet werden, indem statt einem mehrere Anfangswerte verwendet werden. Bei dem Ausleseprozess wird zunächst nach einem Anfangswert oder gegebenenfalls nach mehreren Anfangswerten gesucht, um schließlich nach dem Muster die Differenzwerte auszurechnen. Wenn die Werte 35, 33 und 10 in der Gruppe vorkommen, wird daraus eine Eins interpretiert. Für eine Null sollen in der Gruppe die Werte 23, 23 und 23 zu finden sein. Sollten in einer Gruppe weder die Muster einer Eins noch einer Null zu finden sein, so wird nichts ausgelesen.

Das Verfahren ist wegen der redundanten Einbettung des Vektors innerhalb der Gruppen sehr robust und kann die meisten Manipulationsarten überstehen. Es ist aber in Hinsicht auf die Konvertierung in andere Audioformate fragil, weil dieser Algorithmus nur für den MPEG-2-Standard spezifiziert ist.

3.2 Digitale Audioformate

Ein Audioformat ist ein spezielles Dateiformat, in dem Toninformationen gespeichert werden. Im Prinzip wird grob zwischen zwei Audioformatarten unterschieden: zwischen denen die sogenannten *Samples* nutzen und denen die diese nicht nutzen. Bei Sample-bezogenen Formaten wird die Information durch Abtasten (*Sampling*) des analogen Signals gewonnen und in digitaler Form gespeichert⁹. Dieses Verfahren wird bei dem in dieser Arbeit relevanten Audioformat WAV verwendet. Deswegen wird es im nächsten Unterabschnitt genauer beschrieben. Im Gegensatz dazu verwendet man zum Beispiel bei dem MIDI-Format keine Samples. In einer MIDI¹⁰-Datei werden lediglich Steuerdaten wie Noten, Intensität des Klangs und Lautstärke für die elektronischen Tonerzeuger transportiert [SUN10]. Dieses Audioformat ist für diese Bachelorarbeit irrelevant.

⁸Ein Frame enthält eine feste Anzahl von Abtastwerten (Samples). Siehe auch Unterabschnitt 3.2.1

⁹Der gesamte Abschnitt basiert auf [Ahl02, DMB10, SIT09, ITW10, Kap, Don10, ELK10, SUN10, HR00, Wil03]

¹⁰Musical Instrument Digital Interface

3.2.1 Pulse-Code-Modulation

Die *Pulse Code Modulation* (PCM) ist eines der Pulsmodulationsverfahren¹¹, bei dem durch Abtasten der analogen Signale die entsprechenden digitalen Signale erzeugt werden. Dieses Verfahren wird unter anderem bei der Erstellung von WAV-Dateien verwendet. Es wird aber auch bei der digitalen Nachrichtenübertragung eingesetzt.

Die Klänge, die beispielsweise von Musikinstrumenten erzeugt werden, sind akustische Schall-druckwellen. Diese Schwingungen können zum Beispiel von einem Mikrofon aufgenommen und danach zu elektrischen Signalen (z.B. Spannungsänderungen) umgewandelt werden. Ein Pro-zessor kann solche analogen kontinuierlichen Audiosignale nicht verarbeiten. Deswegen werden sie mit Hilfe sogenannter Analog-Digital-Wandler digitalisiert. Bei der Umwandlung werden elektrische Signale in digitale Form gebracht. Im Prinzip geschieht das in drei Schritten:

1. Abtastung

Zunächst wird das analoge Eingangssignal mit einer konstanten Abtastfrequenz (*Sampling-Takt*) abgetastet. Wie in der Abbildung 3.1 zu sehen, werden die Werte aus der analogen Schwingungskurve in bestimmten Zeitabständen entnommen. Dabei sind in diesem konkreten Fall Spannungswerte auf der Y-Achse und Zeiteinheiten auf der X-Achse abzulesen. Je höher die *Abtastfrequenz* ist, desto genauer kann die analoge Kurve digital erfasst werden, was wiederum eine höhere Musikqualität in digitaler Form ermöglicht. Beispielsweise wird bei Musikstücken in CD-Qualität die *Abtastfrequenz* von 44,1 kHz (44100 Messungen pro Sekunde) verwendet. Die Abtastrate wird normalerweise in Hertz (Hz) oder in *Samples per Second* angegeben.

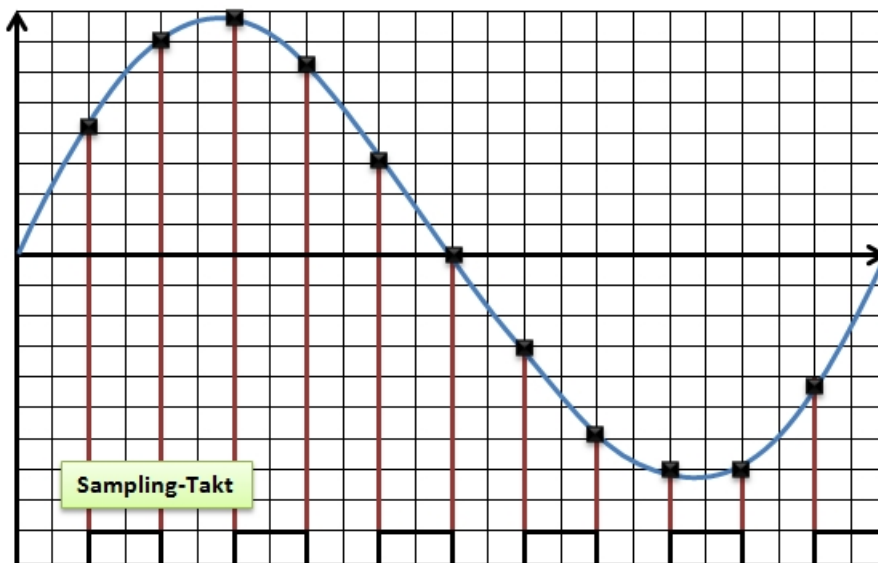


Abbildung 3.1: Abtasten (Angelehnt an [Ahl02])

2. Quantisierung

Bei der *Quantisierung* werden die beim Abtasten abgelesenen analogen Spannungswerte in die diskreten Werte umgewandelt. Unter diskreten Werten sind endliche binäre Werte zu verstehen. Es handelt sich dabei um eine Amplitudenquantifizierung. Der gesamte

¹¹Siehe [ELK10]

Amplitudenbereich (Spannungsstufen) wird in sogenannte Quantifizierungsstufen aufgeteilt. Die analogen Werte werden nun einer dieser Stufen zugeordnet und entsprechend zu diskreten Werten abgerundet, wie es in der Abbildung 3.2 zu sehen ist. Bei zu wenigen Quantisierungsstufen kann es passieren, dass mehrere benachbarte analoge Werte zu einer Quantisierungsstufe abgerundet werden. Das ist unerwünscht, da in diesem Falle der Originalton verzerrt wird. Deswegen kommt neben der *Abtastfrequenz* ein zweiter Parameter die *Sampletiefe* in Betracht. Mit der *Sampletiefe*, auch Quantisierungsaufösung genannt, gibt man die Anzahl an Quantisierungsstufen an. Je größer die *Sampletiefe* ist, desto naturgetreuer kann das analoge Signal erfasst werden. Da ein diskreter Wert später binär codiert und verarbeitet wird, werden die Quantisierungsstufen als Zweierpotenzen angegeben. Zum Beispiel hat die *Sampletiefe* in der Abbildung 3.2 den Wert 3. In diesem Fall wird es $2^3 = 8$ Quantisierungsstufen geben. Bei den Aufnahmen in CD-Qualität wird normalerweise die *Sampletiefe* der Größe 16 mit $2^{16} = 65535$ möglichen Abstufungen genommen.

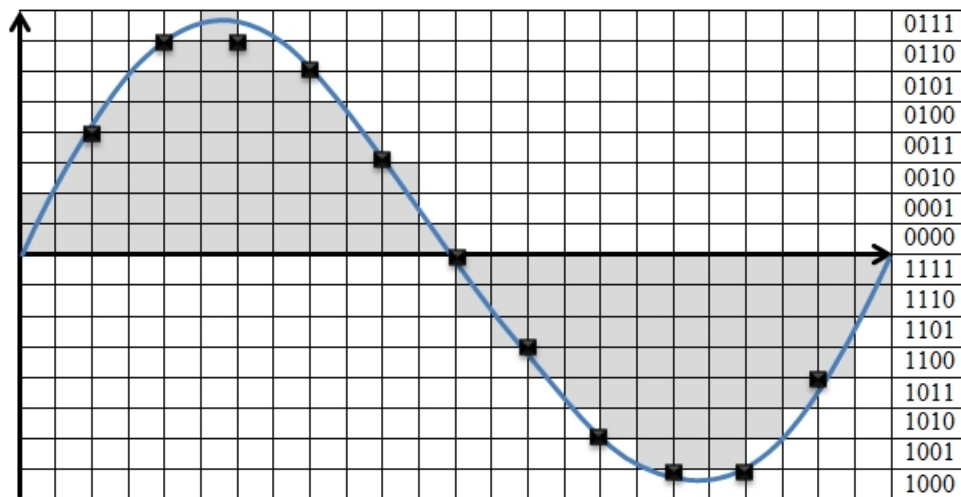


Abbildung 3.2: Quantisieren (Angelehnt an [Ahl02])

3. Codierung

Die quantisierten Werte werden in der Reihenfolge, in der sie aufgenommen wurden, binär codiert. Die Folge der binären Werte beschreibt den Verlauf der analogen Spannungskurve. Diese binären Abtastwerte können nun zum Beispiel beim WAV-Dateiformat in Abhängigkeit von der Sampletiefe in einem, zwei oder vier Bytes pro Abtastwert (*Sample*) untergebracht werden.

Die digitalen Daten können je nach Dateiformat komprimiert oder unkomprimiert gespeichert werden. Die Art der Speicherung dieser Daten ist für den *Least-Significant-Bit*-Wasserzeichenalgorithmus, welcher in dieser Bachelorarbeit verwendet wird, von Bedeutung. Deswegen wird der Aufbau vom WAV-Audioformat im nächsten Unterabschnitt genauer betrachtet.

3.2.2 WAV-Dateiformat

Das WAV-Dateiformat ist eine Untergruppe von Microsofts *Resource Interchange File Format* (RIFF), das für das Betriebssystem Windows entwickelt wurde. Während bei RIFF in einer Datei mehrere

Multimediatypen wie zum Beispiel Video, Audio und Texte untergebracht werden können, werden in einer WAV-Datei nur Audiodaten digital gespeichert. Im Prinzip unterscheidet man zwischen *PCM-WAV-Format* und *Non-PCM-WAV-Format*, die die gleiche Dateierweiterung .wav haben, jedoch teilweise unterschiedlich aufgebaut sind. Die PCM-WAV-Dateien enthalten die durch Pulse-Code-Modulation gewonnenen, unkomprimierten Audiodaten [Ahl02]. Bei dem Non-PCM-WAV-Format werden Daten verwendet, die durch andere Modulationsverfahren wie zum Beispiel Adaptive Differential Pulse Code Modulation (ADPCM) erzeugt wurden. Das Non-PCM-Format ist wenig verbreitet und wird daher von den in dieser Arbeit implementierten Programmen nicht unterstützt.

3.2.2.1 Aufbau

Nach der RIFF-Spezifikation ist eine RIFF-Datei in sogenannte *Chunks* (Abschnitte) unterteilt. Alle RIFF-Dateien beginnen mit einem 12 Bit großen RIFF-Chunk. Dieser enthält folgende Informationen:

chunkID

Hier wird das Wort „RIFF“ als eine ASCII-Zeichenfolge gespeichert. Sie beschreibt, dass es sich um das RIFF-Format handelt.

chunkSize

In ChunkSize wird die Größe der gesamten Datei in Bytes als eine vorzeichenlose Integer-Zahl angegeben.

riffType

gibt den Typ der RIFF-Datei an. Im Falle einer WAV-Datei enthält dieses Element die ASCII-Zeichenfolge „WAVE“.

Der weitere Aufbau der Datei hängt von dem WAV-Typ ab. Im Grunde beinhalten alle WAV-Dateien zwei weitere Chunks nämlich das *fmt-Chunk* und das *data-Chunk*. Diese beiden Abschnitte sind in der Abbildung 3.3 ausführlicher beschrieben. Im Prinzip kann eine WAV-Datei auch weitere Chunks enthalten, in denen zum Beispiel Autor- und Kopierschutzinformationen untergebracht werden können. Wenn eine WAV-Datei jedoch neben dem einleitenden RIFF-Chunk nur die zwei oben erwähnten Chunks enthält, spricht man von einer kanonischen WAV-Datei. Eine kanonische WAV-Datei hat einen 44 Bytes großen *Header*, der den RIFF-Chunk, den kompletten *fmt-Chunk* und den Beschreibungsteil des *data-Chunks* enthält. Unmittelbar nach dem Header folgen dann die eigentlichen PCM-Audiodaten. Dieser Teil der Datei wird auch *Payload* genannt. Bei den nicht-kanonischen WAV-Dateien können weitere Chunks sowohl vor als auch nach dem „data“-Chunk platziert werden [Ahl02, HR00, Wil03]. Die Programme *Watermark Embedding Tool* und *Verification Web-Service*, die in Rahmen dieser Bachelorarbeit implementiert wurden, unterstützen zurzeit nur den kanonischen WAV-Typ. Die Unterstützung für die anderen Typen kann jedoch leicht nachimplementiert werden.

	Chunk	Feldname	Größe in Bytes	Inhalt	
HEADER	RIFF	chunkID	4	ASCII-Zeichen „RIFF“	
		chunkSize	4	Dateilänge	
		riffType	4	ASCII-Zeichen „WAVE“	
	fmt		Subchunk1ID	4	ASCII-Zeichen „fmt“
			Subchunk1Size	4	Größe des Chunks
			AudioFormat	2	Kodierung der Audiowerte; PCM = 1
			NumChannels	2	Anzahl der Kanäle; 1 = Mono, 2 = Stereo
			SampleRate	4	Abtastrate in Hz; z.B. 44100
			ByteRate	4	Durchschnittliche Datenrate in Byte/Sekunde
			BlockAlign	2	NumChannels * BitsPerSample/8
			BitsPerSample	2	Abtastrate in Bits; z.B. 8 oder 16 Bit
	data		Subchunk2ID	4	ASCII-Zeichen „data“
Subchunk2Size			4	Größe des „data“-Chunks	
PAYLOAD		Data	n	Hier sind die eigentlichen Audiodaten gespeichert	

Abbildung 3.3: Aufbau einer kanonischen WAV-Datei (Angelehnt an [Wil03, HR00])

Der *Header*- und *Payload*-Teil sind keine Datenstrukturen von einer WAV-Datei. Diese zwei Begriffe werden hier genannt, um die Nutzdaten, die für Wasserzeichen nützlich sind, von den restlichen Daten abstrakt abzutrennen.

Header In dem Header werden die Informationen über eine WAV-Datei gespeichert, wie es schon teilweise oben beschrieben wurde und in der Abbildung 3.3 zu sehen ist. Für die Programme *Watermark Embedding Tool* und *Verification Web-Service* sind nicht alle Daten des Headers von Bedeutung. So wird zum Beispiel das *riffType*-Feld auf den Inhalt des Wortes „WAVE“ überprüft, um festzustellen, ob es sich hier wirklich um eine WAV-Datei handelt. Aus dem Feld *AudioFormat* wird die Modulationsmethode ausgelesen. In unserem Fall muss der Inhalt des Feldes eine Eins sein, die für die *Pulse-Code-Modulation* steht. Die Felder *NumChannels* und *BitsPerSample* (Quantisierungsaufösung) beschreiben den Aufbau des Payloads [Wil03]. Die anderen Felder sind für die in dieser Arbeit realisierten Programmen nicht relevant und spielen zum Beispiel für digitale Audioplayer eine Rolle.

Payload¹² enthält die durch *Pulse-Code-Modulation* erzeugten, unkomprimierten Audiodaten. Diese befinden sich in dem *data*-Chunk, wie aus der Abbildung 3.3 zu entnehmen. Die bei PCM ermittelten digitalen Abtastwerte (Samples) werden nun als 8, 16 oder 32 Bit große Dualzahlen nacheinander gespeichert. In einer WAV-Datei ist die Anzahl der Samples immer gerade. Bei einer ungeraden Anzahl der Samples wird ein sogenanntes *Pad-Byte* mitgespeichert, welches nur aus Null-Bits besteht.

¹²Die Payload-Beschreibung ist an [Wil03, HR00] angelehnt

Bei dem *Least-Significant-Bit-Algorithmus*, welcher im vorherigen Abschnitt detaillierter beschrieben wurde, wird die Wasserzeicheninformation in den niedrigwertigsten Bits der Samples eingebettet. Bei der Darstellung einer Binärzahl bezeichnet man das Bit mit der niedrigsten Bitwertigkeit $1 (2^0=1)$ als *Least-Significant Bit*¹³. Im Prinzip besitzt jedes Sample ein LSB. Um die Wasserzeicheninformation auslesen und einbetten zu können, muss der Ort des niedrigwertigen Bits in einem Sample ermittelt werden. Er ist von der Samplegröße abhängig.

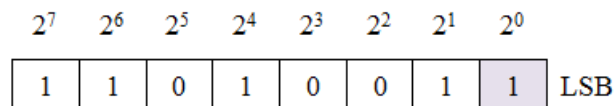


Abbildung 3.4: 8-Bit Sample (Angelehnt an [Wil03])

In dem in der Abbildung 3.4 vorgestellten Beispiel ist der Aufbau eines 8-Bit Samples zu sehen. Dabei werden die einzelnen Bits im sogenannten *Big-Endian-Format* gespeichert. Dies bedeutet, dass die Bits mit höherwertigen Stellen vor den Bits mit niedrigwertigen Stellen angeordnet sind. Das *Least-Significant-Bit* befindet sich deswegen an der letzten Stelle des Bytes. In der Abbildung 3.4 ist das LSB farblich hervorgehoben.

Die 16-Bit Samples werden im Gegensatz dazu anders gespeichert. Da ein Sample nun aus zwei Bytes besteht, muss die Byte-Ordnung berücksichtigt werden. Während die einzelnen Bits eines 16-Bit-Samples genauso wie bei einem 8-Bit Sample geordnet sind, werden seine Bytes im *Little-Endian-Format* gespeichert. Bei der Little-Endian-Reihenfolge werden die niedrigwertigen Bytes zuerst gespeichert. Ein 16-Bit Sample ist in der Abbildung 3.5 dargestellt. Somit befindet sich in diesem Fall das LSB in der Mitte eines Samples und ist farblich hervorgehoben.

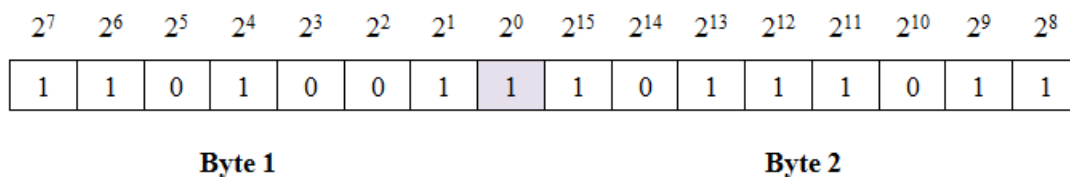


Abbildung 3.5: 16-Bit Sample (Angelehnt an [Wil03])

Nachdem in diesem Unterunterabschnitt ein grober Überblick über den Aufbau einer WAV-Datei gegeben wurde, soll nun die Einsetzbarkeit von WAV-Dateien als Trägermedium eingeschätzt werden.

3.2.2.2 WAV-Datei als Trägermedium

Die Speicherung von den PCM-Rohdaten ermöglicht eine hohe Datenrate beim PCM-WAV-Format. Dies hat Vor- und Nachteile. Bezogen auf den Least-Significant-Bit-Wasserzeichenalgorithmus bietet eine WAV-Datei viel Platz für die einzubettende Wasserzeicheninhalte. So können zum Beispiel in einem 50 Mbyte großen Song bis zu 3 Mbyte an Wasserzeicheninformationen eingebunden werden. Dies wird dabei wie folgt ausgerechnet: Von der

¹³Siehe <http://www.itwissen.info/definition/lexikon/least-significant-bit-LSB-Geringstwertiges-Bit.html>; Letzter Abruf: 16.03.2010

Größe der Datei in Bits $50 \text{ Mbyte} = 50 \times 1024 \times 1024 \times 8 = 419430400$ Bits wird die Größe des Headers in Bits abgezogen $419430400 - 44 \times 8 = 419430048$ Bits. Somit bekommt man die Größe des Payloads. Angenommen bei einer WAV-Datei mit 16 Bits pro Sample wird jedes 16te Bit des Payloads für die Wasserzeicheninformationen verwendet: $419430048/16 = 26214378$ Bits. Jetzt werden diese Bits wieder in MBytes umgerechnet: $26214378 / (8 \times 1024 \times 1024) = 3,123$ Mbyte. Da das LSB-Verfahren nicht robust ist, können so die Informationsverluste durch das redundante Einbetten an mehreren Stellen in der Datei potenziell verringert werden.

Andererseits sind die WAV-Dateien für den Internetverkehr zu groß. So hat zum Beispiel die Übertragung einer 10 Mbyte großen WAV-Datei mit der 800 kbit/s Übertragungsrates beim Testen des Verification-Web-Service-Programms mehr als 3 Minuten in Anspruch genommen. Solche Wartezeiten sind für das URM-System unzulässig. Im Allgemeinen werden WAV-Dateien bei der Internetübertragung wegen ihrer Größe nicht verwendet. Vielmehr kommen heutzutage die in dem nächsten Unterabschnitt vorgestellten Audioformate zum Einsatz, bei denen verschiedene Komprimierungstechniken verwendet werden. Aber für die prototypischen Funktionen der in dieser Arbeit implementierten Programme reicht das WAV-Format aus.

3.2.3 MPEG Layer 3 (MP3)

Das heutzutage am weitesten verbreitete Audioformat ist die MPEG Layer 3 (MP3)-Kodierung, die von der Motion Picture Experts Group (MPEG) entwickelt wurde. Dabei hat die Fraunhofer-Gesellschaft den entscheidenden Teil beigetragen. [SIT09, Kap, RTM02]

Die MP3-Audiokompression gehört zu den verlustbehafteten Datenkompressionsverfahren. Das bedeutet, dass die wenig relevanten Dateninhalte unwiderruflich entfernt werden. Bei dem MP3-Verfahren kommt die sogenannte Psychoakustik zum Einsatz. Im Gegensatz zu PCM-Verfahren werden hier nur die Audiosignale bearbeitet, die von einem Menschen wahrgenommen werden können. Die Töne, die sich zum Beispiel überlagern oder die zu leise sind, können von dem menschlichen Gehör gar nicht oder nur schlecht wahrgenommen werden. Entsprechend diesen Erkenntnissen wird das digitale Audiosignal in Frequenzbänder zerlegt und die irrelevanten Teilsignale werden ausgefiltert. Des Weiteren werden weitere mathematischen Schritte und Kompressionsverfahren angewendet, um die Datenmenge zu reduzieren. So kann bei geringem Speicherbedarf eine hohe Soundqualität erreicht werden. [SIT09, Kap, DMB10]

Das MP3-Format ist in der Musikindustrie schon längst zum Standard geworden. Eine MP3-Datei ist bei der gut wahrnehmbaren Qualität nur ein Zehntel der vergleichbaren WAV-Datei groß. Dies macht die MP3-Kodierung besonders für die Datenübertragung in Netzwerken wie zum Beispiel Internet interessant. Bei all diesen Vorteilen können jedoch nicht alle Wasserzeichenverfahren angewendet werden. So basiert beispielsweise das Echo-Wasserzeichen, welches im vorherigen Abschnitt beschrieben wurde, auf gleichen psychoakustischen Erkenntnissen wie das MP3-Verfahren. So können bei der MP3-Kodierung die Frequenzbereiche ausgefiltert werden, die das Wasserzeichen beinhalten. Außerdem wird heutzutage nach besseren Komprimierungsverfahren geforscht. Als potenzieller Nachfolger von MP3 kann das Advanced Audio Coding (AAC)-Format angesehen werden.

3.2.4 Advanced Audio Coding (AAC)

AAC ist eine Weiterentwicklung vom MP3-Format, welche die psychoakustischen Effekte noch stärker ausnutzt und somit eine höhere Kompression bei teilweise besserer Audioqualität bietet. AAC wurde ebenfalls von MPEG entwickelt und ist Bestandteil vom MPEG-2 und MPEG-4 Formaten.[ITW10]

Obwohl das AAC-Format technisch ausgereifter als MP3 ist, sind zurzeit die Lizenzkosten zu hoch, was die Ausbreitung des Formats verhindert. AAC wird zum Beispiel bei Apples iTunes teilweise mit der DRM-Kopierschutzmaßnahme Fairplay eingesetzt. [ITW10, Kap]

3.3 Web-Services¹⁴

Der Begriff Web-Service wird in der Literatur unterschiedlich aufgefasst. Eine präzise Definition kann mit einem Zitat vom World Wide Web Consortium (W3C) ausgedrückt werden:

“A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

Quelle: <http://www.w3.org/>

Die *Web-Services* sind im Allgemeinen eine Technologie zum Realisieren von verteilten Anwendungen. Im Mittelpunkt stehen lose Bindung und Nutzung von sogenannten Diensten über ein Netzwerk. Die Dienste (Services) sind als Knoten eines verteilten Systems zu verstehen, die XML-basierte Schnittstellen haben, klar gekapselt sind und genau definierte Aufgaben ohne menschlichen Einfluss bearbeiten können. Ein Mensch nutzt dabei *Web-Services* nur mittelbar. Die Kommunikation zwischen Komponenten des Systems erfolgt maschinenorientiert. Prinzipiell kann ein Web-Service aus mehreren Diensten (Services) bestehen, die unterschiedliche Funktionalitäten haben.

Die Verwendung von *Web-Services* basiert auf Ansätzen der Service-orientierten Architektur (SOA). Dabei wird das abstrakte Konzept der SOA durch Einsatz von bestimmten Techniken konkretisiert. SOA soll deswegen als theoretische Grundlage vor der Beschreibung der *Web-Services*-Architektur erläutert werden.

3.3.1 Service-orientierte Architektur (SOA)

Während es sich bei *Web-Services* um eine konkrete Technologie handelt, ist die Service-orientierte Architektur ein abstraktes Konzept der Software-Architektur, welches keinen genau definierten Implementierungsweg vorschreibt und in dessen Mittelpunkt das Anbieten, Suchen und Nutzen der Dienste in einem Netzwerk steht. Die unterschiedlichen, eventuell inkompatiblen Anwendungen oder Funktionalitäten, werden dabei als wiederverwendbare, plattform- und programmiersprachenunabhängige Dienste beziehungsweise Services repräsentiert. Die typischen Merkmale einer SOA sind lose Kopplung, dynamisches Einbinden, Verzeichnisdienste und die Verwendung von Standards.

Loose Kopplung und **dynamisches Binden** bedeuten, dass die Dienste zur Laufzeit gesucht und benutzt werden. Diese Eigenschaften sind insofern interessant, weil die Dienste in diesem Fall flexibel und effizient genutzt werden können. Die Dienste werden erst aufgerufen, wenn sie benötigt werden. Dabei soll ein passender Dienst dynamisch gefunden werden.

¹⁴Der gesamte Abschnitt basiert auf [Kli05, DJMZ05]

Ein **Verzeichnisdienst** ermöglicht eine dynamische Suche und ist damit Voraussetzung für das dynamische Binden. In den Verzeichnisdienst werden die zur Verfügung stehenden Dienste eingetragen.

Die **Verwendung von Standards** ist eine der wichtigsten Eigenschaften von SOA. Nachdem ein Dienst von einer Anwendung gefunden wurde, muss der Dienstanutzer fähig sein, mit dem Dienst kommunizieren zu können. Das setzt eine einheitliche Schnittstelle voraus. Es ist also wichtig, dass die Kommunikationsteilnehmer dabei gleiche Standards nutzen.

Im Groben kann die Struktur einer SOA mit dem sogenannten magischen Dreieck beschrieben werden, das in der Abbildung 3.6 abgebildet ist.

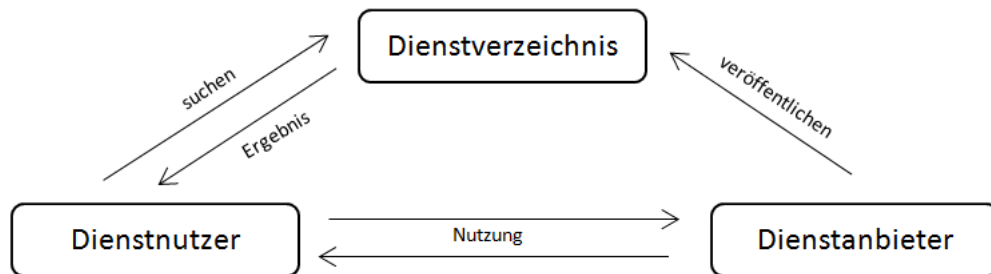


Abbildung 3.6: Struktur einer SOA (Quelle: [DJMZ05])

Das Zusammenspiel der Komponenten von SOA ist wie folgt zu verstehen:

Ein **Dienstanbieter** stellt mindestens einen Dienst zur Verfügung, auf den über ein Netzwerk zugegriffen werden kann. Ein Dienst ist in diesem Kontext eine Anwendung oder eine Software-Komponente. Der Zugriff auf Dienste erfolgt über eine vom *Dienstanbieter* öffentlich beschriebene Schnittstelle. Eine weitere Aufgabe des *Dienstanbieters* ist für die Sicherheit der von ihm angebotenen Dienste zu sorgen. Ein *Dienstanbieter* muss bei einem Zugriff auf seine Dienste prüfen, ob der Dienstaufrufer die Rechte hat, den entsprechenden Dienst aufzurufen und welche Funktionalitäten des Dienstes er dabei in Anspruch nehmen darf.

Bevor die Dienste genutzt werden können, müssen sie zunächst von dem *Dienstanbieter* veröffentlicht werden. Dabei wird die *Dienstbeschreibung* in ein *Dienstverzeichnis* (oder auch Verzeichnisdienst) eingetragen. Die *Dienstbeschreibung* enthält eine vollständige Beschreibung der Schnittstellen des jeweiligen Dienstes in einer für die Dienstanutzer lesbaren, plattform- und programmiersprachunabhängigen Form. Des Weiteren werden die Funktionalitäten des Dienstes und weitere Eigenschaften wie Verfügbarkeit und Kosten für die Nutzung beschrieben.

Das **Dienstverzeichnis** ist eine Art Eintragsbuch, dessen primäres Ziel ist, die eingetragenen Dienste zu verwalten und für den potenziellen Aufrufer auffindbar zu machen. Im Allgemeinen werden die Dienste von *Dienstanbietern* in solchen Verzeichnissen registriert. Ein *Dienstverzeichnis* kann aber auch selbst als ein Dienst fungieren und ähnlich einer Suchmaschine aktiv nach Diensten suchen.

Dem **Dienstanutzer** ist es im Prinzip egal, welcher Anbieter den von ihm gesuchten Dienst anbietet. Wichtig für ihn ist, dass die Standards eingehalten werden, die Schnittstellen übereinstimmen und der Kosten-Leistungsaspekt attraktiv ist. Wenn ein Nutzer die Funktionalität eines Dienstes in Anspruch nehmen möchte, muss er zunächst anhand der Beschreibung der gewünschten Funktionalität eine Suchanfrage an das Dienstverzeichnis stellen. Anhand der Dienstbeschreibung wird dann im Verzeichnis automatisch nach dem gewünschten Dienst gesucht und schließlich auf einen passenden Dienst verwiesen.

Nachdem ein Dienst gefunden worden ist, werden zunächst von dem Dienstanutzer die Schnittstellen dieses Dienstes abgefragt. Danach werden die Richtlinien ausgetauscht. Dabei wird von Seiten des Dienstes zum Beispiel überprüft, ob der Dienstanutzer die Rechte hat, den entsprechenden Dienst zu nutzen. Falls nach diesem Schritt eine Einigung erreicht werden konnte, kommt es zu der eigentlichen Nutzung des Dienstes.

Die service-orientierte Architektur wurde im Allgemeinen für die Unternehmen entwickelt, die ihre Geschäftsprozesse selber modellieren können und die Umsetzung von Teilprozessen den externen Partner übergeben. Es werden dabei folgende Ziele verfolgt:

- Kostensenkung durch die wiederverwendbaren Dienste
- Flexibilitätssteigerung durch lose Bindung der Komponenten von Geschäftsprozessen und
- gleichzeitig Reduktion der Komplexität durch Verteilung der Aufwände

Diese Idee ist nicht neu und kommt aus dem Bereich der komponentenbasierten Architekturen. Allerdings bietet SOA durch den komplett technologie-unabhängigen Ansatz eine große Freiheit bei der Umsetzung von service-orientierten Konzepten.

Die *Web-Services* sind eine mögliche Umsetzung des abstrakten Modells der SOA. In dem nächsten Unterabschnitt wird diese Technik genauer betrachtet. Angesichts der Tatsache, dass das *Web-Services*-Konzept auf den SOA-Grundprinzipien beruht, ist es sinnvoll die Vor- und Nachteile der SOA an dem konkreten Beispiel der *Web-Services* zu untersuchen.

3.3.2 Web-Services-Architektur

Die *Web-Services*-Implementierung ist flexibel, was die Umsetzungstechniken angeht. Der plattform- und programmiersprachenunabhängige Ansatz von SOA wird durch Verwendung von den weit verbreiteten und von Entwicklern akzeptierten Standards realisiert. Meistens wird dabei auf XML¹⁵-basierte Lösungen gesetzt, welche diesen Ansatz ermöglichen.

Die Standards werden von sogenannten Gremien¹⁶ beschlossen und sind im Gegensatz zu einer Norm für die Entwickler nicht bindend. Unter anderem beschäftigen sich *World Wide Web Consortium (W3C)* und *Organization for the Advancement of Structured Information Standards (OASIS)*¹⁷ mit Fragen der Standardisierung einer *Web-Services*-Architektur.

Bei der *Web-Services*-Architektur werden grundsätzlich folgende Standards, welche in den nächsten Unterunterabschnitten detaillierter beschrieben werden, verwendet:

SOAP

Bei der Kommunikation der Komponente wird das XML-basierte Nachrichtenformat SOAP eingesetzt. Die SOAP-Nachrichten werden dabei in ein Transportprotokoll eingebettet.

WSDL

Die Dienstbeschreibung der *Web-Services* erfolgt durch WSDL, eine XML-basierte Beschreibungssprache.

UDDI

UDDI beschreibt ein Verzeichnisdienst für *Web-Services*. Die Aufgabe von UDDI ist die Verwaltung von Web-Service-Metadaten. Diese sind zum Beispiel Web-Service-Informationen und allgemeine Anforderungen zum Aufrufen von *Web-Services*.

¹⁵Detaillierte Beschreibung zu XML im Allgemeinen ist in [Jä10] zu finden

¹⁶In diesem Kontext Zusammenarbeit von mehreren Unternehmen

¹⁷Siehe <http://w3c.de/> und <http://www.oasis-open.org/>

Die genannten Standards sind keinesfalls für die Implementierung der service-orientierten Architektur verpflichtend. So könnte zum Beispiel an Stelle von SOAP ein anderes Nachrichtenformat genommen werden, welches die SOA-Anforderungen wie Plattform- und Programmiersprachenunabhängigkeit erfüllt. Allerdings haben sich heutzutage diese Standards etabliert, nicht zuletzt wegen der XML-basierten Arbeitsweise.

Prinzipiell besitzt die *Web-Services*-Architektur die gleichen Basiskomponenten wie eine SOA. Die Beziehungen zwischen Dienstanbieter, Dienstanutzer und Dienstverzeichnis werden anhand der oben erwähnten Standards konkretisiert (Abbildung 3.7):

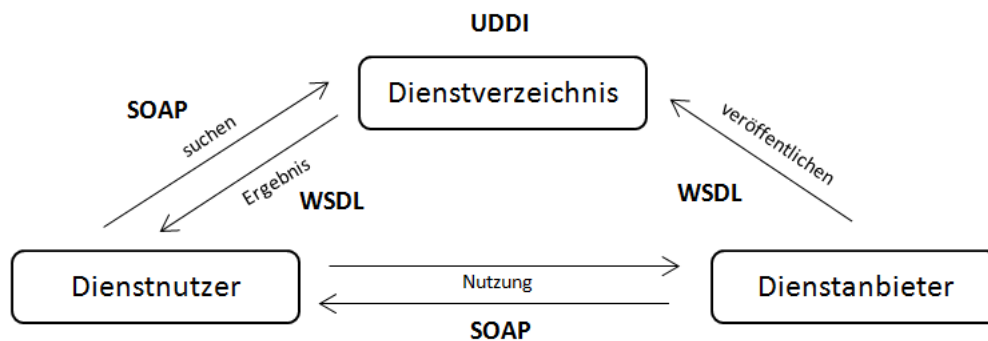


Abbildung 3.7: Web-Services-Architektur (Quelle: [DJMZ05])

Ein Dienstanbieter, der einen Dienst in Form eines *Web-Services* anbieten möchte, definiert zuerst die Schnittstellen des entsprechenden Dienstes in Form eines WSDL-Dokuments. Nun wird dieser Dienst veröffentlicht, indem der Anbieter die WSDL-Dienstbeschreibung zu einem UDDI-Dienstverzeichnis schickt. Ein potenzieller Dienstbenutzer stellt mit Hilfe vom SOAP-Nachrichtenprotokoll eine Suchanfrage an das Dienstverzeichnis mit der Beschreibung des gesuchten Dienstes. Falls ein passender Dienst existiert, liefert das Dienstverzeichnis eine Referenz auf das WSDL-Dokument des gesuchten Dienstes. Danach erzeugt der Dienstanutzer anhand der Schnittstellenbeschreibung die Komponente, die zu der Kommunikation mit dem Anbieter notwendig sind. Schließlich kommt es zum Nachrichtenaustausch zwischen dem Dienstanutzer und dem Dienstanbieter mittels SOAP .

3.3.2.1 SOAP

SOAP ist ein Kommunikationsprotokoll, welches den Aufbau von XML-basierten Nachrichten beschreibt. Durch Einsatz von dem wohlbekanntem XML-Standard ermöglicht es die plattform- und programmiersprachenunabhängige Übertragung von Daten. Damit erfüllt SOAP die SOA-Anforderungen und bildet eine Kommunikationsgrundlage für die *Web-Services*-Architektur.

Da SOAP nur das Nachrichtendesign beschreibt, wird es bei der Nachrichtenübertragung zusammen mit einem Transportprotokoll verwendet. Meistens wird dabei HTTP oder SMTP eingesetzt.

Aufbau

Eine SOAP-Nachricht ist ein wohlgeformtes XML-Dokument, welches aus drei wesentlichen Teilen besteht (Abbildung 3.8):

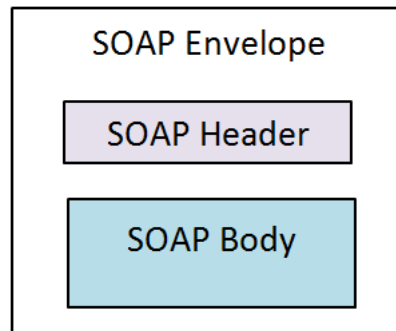


Abbildung 3.8: Aufbau einer SOAP-Nachricht (Quelle: [DJMZ05])

Der **SOAP Envelope** ist das Wurzelement der XML-Nachricht. Dieses Element muss in einer SOAP-Nachricht genau einmal vorhanden sein.

Der **SOAP Header** ist ein optionales Element, darf aber in der Nachricht maximal einmal vorkommen. Es gibt keine genaue Spezifikation darüber, was ein Header enthalten kann. Dieses Element wird hauptsächlich zum Übertragen von Sicherheitsinformationen wie dem Passwort und den Rechten des Dienstnutzers verwendet.

Der **SOAP Body** enthält die eigentlichen Nutzdaten der Nachricht. Diese Nutzdaten sind anwendungsspezifisch. Im Body-Teil kann ein optionales Fault-Element enthalten sein, welches die Informationen über die möglichen bei der Übertragung entstandenen Fehler trägt.

Im Groben können mit SOAP zwei Übertragungsarten von Body-Inhalten unterschieden werden:

document-style

In diesem Fall kann ein Body beliebige Informationen enthalten, die sich als ein XML-Dokument darstellen lassen. Zum Beispiel wären dies HTML-Seiten oder PDF-Dokumente.

RPC-style

Bei dieser Art enthält ein Body einen entfernten Prozeduraufruf (Remote Procedure Call). Dadurch wird eine entfernte Methode mit bestimmten Parametern aufgerufen und das Ergebnis an den Aufrufer zurückgeliefert.

Normalerweise bekommt der Anwendungsentwickler eines *Web-Services* eine SOAP-Nachricht nicht zu sehen. Moderne Entwicklertools erzeugen und übertragen diese automatisch. Es ist jedoch wichtig den Aufbau von SOAP-Nachrichten zu kennen, um Potenziale und Möglichkeiten von SOAP richtig einschätzen zu können.

3.3.2.2 Web-Services Description Language (WSDL)

Die Schnittstellen und Funktionalität eines *Web-Services* werden mit dem XML-basierten W3C-Standard Web-Services Description Language (WSDL) beschrieben. Diese Beschreibung erfolgt im Allgemeinen auf zwei Ebenen: abstrakt und konkret.

Die abstrakte Ebene enthält allgemeine Informationen über die Funktionalität eines *Web-Services*. Diese wären zum Beispiel die Beschreibung von Methoden, dazugehöriger Parameter und abstrakten Datentypen. Die abstrakte Ebene hat folgende Elemente¹⁸:

¹⁸Siehe auch [Jä10]

types	beschreibt Definitionen und Datentypen, die beim Nachrichtenaustausch eingesetzt werden.
message	ist eine abstrakte Beschreibung von Nachrichten, die ausgetauscht werden. Die Nachrichten dürfen nur die bei <i>types</i> definierten Typen verwenden. Eine Nachricht kann dabei aus mehreren logischen Teilen (<i>parts</i>) bestehen, die jeweils an einen Datentypen gebunden sind.
operation	beschreibt einen Funktionsaufruf mit den dazugehörigen input, output und fault-Nachrichten.
portType	fasst mehrere Funktionen mit Angabe ihrer Nachrichtentypen zusammen.

Die konkrete Ebene beschreibt die technische Seite eines *Web-Services*. Diese Ebene enthält solche technischen Details wie Informationen über verwendete Netzwerkprotokolle, genaue Adresse des *Web-Services* und wie ein Dienst angeboten wird. Sie besteht aus folgenden Elementen:

binding	bindet die vorher definierten, abstrakten Nachrichten und Operationen an konkrete Netzwerkprotokolle und Datenformate.
port	spezifiziert die Netzwerkadresse (auch <i>Endpoint</i> genannt), unter der ein Web-Service in einem Netzwerk zu erreichen ist.
service	fasst alle unter port definierten Adressen zusammen.

Das Zusammenspiel dieser Elemente wird an dem konkreten Beispiel der Programmbeschreibung vom Verification Web-Service genauer betrachtet.

Eine vollständige und programmiersprachenunabhängige Schnittstellenbeschreibung ist ein wichtiger Bestandteil eines flexiblen und verteilten Systems. Durch den Einsatz von WSDL wird die technische Umsetzung von SOA-Prinzipien ermöglicht. WSDL ist daher ein wichtiger Bestandteil von einer *Web-Services-Architektur*.

3.3.2.3 Web-Services Inspection Language (UDDI)

UDDI ist ein XML-basiertes Konzept zum Auffinden von *Web-Services*. Wie bereits erwähnt wurde, beschreibt UDDI, wie ein Verzeichnisdienst aufgebaut sein soll. Das Ziel von so einem Verzeichnisdienst ist, die Schnittstellen- und Funktionalitätsbeschreibung von *Web-Services* in Form der XML-basierten WSDL-Dokumente zu verwalten und dem Dienstanutzer diese zugänglich zu machen.

Das UDDI-Konzept setzt auf ein zentralisiertes Verzeichnissystem mit wenigen Verzeichnissen, in denen mehrere Anbieter ihre Dienste veröffentlichen können. Als Alternative zu UDDI wird das sogenannte Web-Services-Inspection-Language-Konzept (oder einfach WS-Inspektion) betrachtet. Im Gegensatz zu UDDI setzt WS-Inspection auf viele kleinere Dienstverzeichnisse, die die Dienstbeschreibungen von wenigen oder sogar einem Anbieter verwalten müssen.

Im Prinzip wird ein Dienstverzeichnis nicht mehr benötigt, wenn ein Dienstanbieter einem Dienstanutzer schon bekannt ist. In diesem Fall kennt der Nutzer die Netzwerkadresse des Anbieters und kann mit Hilfe vom WSDL-Dokument des Anbieters die zur Kommunikation benötigten Komponenten erzeugen. Bei der Implementierung vom Verification-Web-Service-Programm wird aus

diesem Grund ein Verzeichnisdienst nicht benötigt. Da ein Verzeichnisdienst für diese Arbeit nicht relevant ist, wird UDDI an dieser Stelle nicht weiter beschrieben.

3.3.3 Vor- und Nachteile von Web-Services

Die Eigenschaften von *Web-Services* sind im Allgemeinen durch Merkmale einer SOA geprägt. In diesem Zusammenhang sind folgende Vor- und Nachteile zu erwähnen:

Vorteile

Durch den Einsatz von den offenen, weit verbreiteten XML-basierten Standards sind *Web-Services* plattform-, betriebssystem- und programmiersprachenunabhängig. Dadurch entsteht eine flexible Architektur, bei der die einzelnen Komponenten leicht ausgetauscht oder zusammengesetzt werden können. Dies führt zu der von SOA angestrebten Wiederverwendbarkeit von Komponenten. Die wiederverwendbaren, beliebig kombinierbaren Komponenten ermöglichen ihrerseits die Kostensenkung und Aufwandsreduktion von Geschäftsprozessen.

Nachteile

Der Einsatz von XML erweist sich in Bezug zur Performance als nachteilig. Durch den textorientierten Ansatz von XML müssen Web-Service-Komponenten die Daten mit geringerer Nutzungsdichte als zum Beispiel Bitströme verarbeiten und übertragen. Die *Web-Services* sind dadurch eventuell langsamer als andere Techniken in diesem Bereich. Im Prinzip bringt aber der Einsatz der XML-Technik keine großen Leistungseinbrüche mit sich. Bei komplexen Geschäftsprozessen kann zu starke Verteilung von Teilaufgaben ebenfalls zur Performanceminderung und unnötigen Aufwänden führen.

Kapitel 4

Watermark Embedding Tool

In diesem Kapitel wird das Programm Watermark Embedding Tool beschrieben, mit dem ein Wasserzeichen mit Hilfe des Least-Significant Bit-Algorithmus in eine WAV-Datei integriert werden kann. Dieses Programm sollte implementiert werden, um die Funktionalität des Verification-Web-Services testen zu können. Mit dem Watermark Embedding Tool wird ein Wasserzeichen in eine Datei eingebettet, welches wiederum vom Verification-Web-Service ausgelesen werden kann.

In den folgenden Abschnitten wird zunächst der Entwurf des Programms vorgestellt, welcher einen kurzen Einblick in die Programmarchitektur und die entsprechenden Programmteile gibt. Danach wird die technische Umsetzung des Programms beschrieben. Schließlich wird die Funktionalität des Programms für einen Benutzer deutlich gemacht.

4.1 Entwurf

Der Entwurf von dem Watermark Embedding Tool besteht aus zwei Teilen. In dem Unterabschnitt 4.1.1 Anforderungen soll ein grober Überblick gegeben werden, was alles bei der Implementierung des Programms umgesetzt werden soll. Der zweite Unterabschnitt Architektur enthält eine konkrete Programmarchitektur mit den dazugehörigen Klassen und dem Zusammenspiel der einzelnen Komponenten.

Der Entwurf und die Implementation des Watermark Embedding Tools sind hauptsächlich durch Anforderungen an den Verification-Web-Service geprägt. So soll das Programm mit WAV-Dateien umgehen können. Bei der Implementation des Wasserzeichenalgorithmus soll wie bei dem Verification-Web-Service ebenfalls der Least-Significant-Bit-Algorithmus verwendet werden. Die weiteren Anforderungen an das Programm konnten frei gewählt werden. So habe ich mich für die Programmierumgebung Java entschieden, da sie alle für das Programm notwendigen Funktionalitäten wie:

- Arbeit mit WAV-Dateien
- Speichern und Auslesen der Dateien im Allgemeinen und
- Arbeit mit binären Datenstrukturen bietet.

Außerdem konnte ich im Laufe meines Studiums einige Erfahrungen mit Java ansammeln. Die Implementation der GUI ist eine optionale Anforderung und erleichtert lediglich den Umgang mit dem Programm. Alternativ könnte das Programm durch Konsolenbefehle genutzt werden, was aber nicht besonders benutzerfreundlich wäre.

4.1.1 Anforderungen

In diesem Abschnitt werden die Anforderungen zusammengefasst, die für das Watermark Embedding Tool relevant sind:

- Die Interaktion des Programms mit dem Benutzer soll mit Hilfe eines Graphical User Interfaces (GUI) erfolgen.
- Das Least-Significant-Bit-Wasserzeichen soll in eine WAV-Datei eingebettet werden können.
- Der Benutzer muss eine WAV-Datei (Input-Datei), in die er ein Wasserzeichen einbettet möchte, auswählen können.
- Die Auswahl der Input-Datei soll durch direktes Eintippen von dem Namen der Datei ermöglicht werden.
- Die Auswahl der Input-Datei kann per Filebrowser erfolgen.
- Die Ausgewählte Datei soll auf Gültigkeit überprüft werden.
- Bei einer nicht erfolgreichen Gültigkeitsüberprüfung soll eine entsprechende Meldung dem Benutzer angezeigt werden.
- Der Benutzer soll die Möglichkeit haben, den Wasserzeicheninhalt in Form eines Textes eingeben zu können.
- Die Einbettung des Wasserzeicheninhalts soll mit dem Least-Significant-Bit-Algorithmus realisiert werden.
- Der Benutzer soll die Möglichkeit haben, die Input-Datei mit dem integrierten Wasserzeichen zu können.
- Der Name der zu speichernden Datei (Output-Datei) soll frei wählbar sein.
- Die Auswahl der Output-Datei soll durch direktes Eintippen von dem Namen der Datei ermöglicht werden.
- Die Auswahl der Output-Datei kann per Filebrowser erfolgen.
- Beim Speichern der Output-Datei soll dem Benutzer eine entsprechende Meldung angezeigt werden, wenn die Datei schon existiert.
- Bei den erfolgreichen Einbettungs- und Speichervorgängen soll dem Benutzer eine entsprechende Meldung angezeigt werden.
- Beim Fehlschlag dieser Vorgänge soll eine entsprechende Fehlermeldung erscheinen.

4.1.2 Architektur

Die Programmarchitektur besteht aus zwei logischen Teilen: *Watermarker* und *GUI*. Ich werde im Folgenden den Programmabschnitt, der für das Einbetten in eine WAV-Datei zuständig ist, *Watermarker* nennen und den Programmabschnitt, der für die Interaktion mit den Benutzern zuständig ist, *GUI* bezeichnen. Der Programmkern *Watermarker* wird von dem *GUI* logisch getrennt. Der Vorteil dieser Trennung ist, dass der *Watermarker* in Kombination mit unterschiedlichen GUIs eingesetzt werden kann.

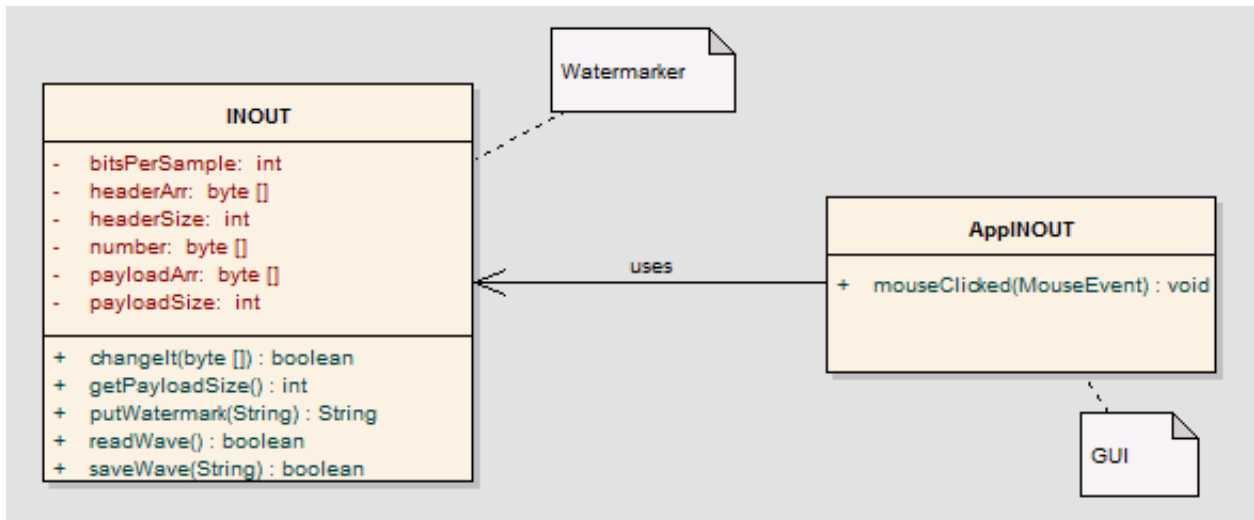


Abbildung 4.1: Programmarchitektur

Wie in Abbildung 4.1 zu sehen, haben die von mir vorgestellten logischen Abschnitte jeweils eine Klasse:

Watermarker

Der Watermarker-Abschnitt enthält die Klasse *INOUT.java*. Diese Klasse enthält die eigentliche Funktionalität des Least-Significant-Bit-Algorithmus. Dabei unterscheide ich zwischen mehreren Hauptfunktionen, die in fester Reihenfolge nacheinander ausgeführt werden sollen, und den Hilfsfunktionen.

- Hauptfunktionen:

readWave() In dieser Funktion wird eine WAV-Datei eingelesen. Dabei wird überprüft, ob diese Datei ein unterstütztes WAV-Format besitzt. Wenn das Lesen der Datei erfolgreich war, wird ein boolescher Wert true zurückgegeben. Anderenfalls wird false zurückgegeben. Nach dem erfolgreichen Auslesen werden die Header- von den Payload-Daten der Datei voneinander getrennt. Wie im Unterabschnitt 3.1.2.1 beschrieben wurde, sind für den LSB-Algorithmus nur die Nutzdaten von Bedeutung. Deswegen werden diese von dem Rest getrennt.

changeIt(byte[] wmArr)

Wenn die Payload-Daten von dem Rest separiert wurden, wird damit weitergearbeitet. Dieser Funktion wird ein Wasserzeichen, das eingebettet werden soll, in Form eines Byte-Arrays als ein Parameter übergeben. Nun wird das Wasserzeichen in die Payload-Daten integriert. Die genaue Beschreibung dieses Vorgangs wird im Abschnitt Implementation beschrieben. Die Funktion liefert den booleschen Wert true, wenn das Wasserzeichen eingebettet werden konnte, anderenfalls false.

saveWave(String outName)

Nachdem das Wasserzeichen integriert wurde, müssen die Änderungen an den Nutzdaten gespeichert werden. Dabei wird das Header-Array mit dem geänderten Payload-Array wieder vereinigt und zu einem Dateistrom transformiert. Wenn das Speichern ohne Fehler verlief, wird true zurückgeliefert, sonst false.

- Hilfsfunktionen:

getPayloadSize()

Mit der Funktion wird die Größe der Nutzdaten zurückgeliefert.

putWatermark(String wm)

Die Funktion bekommt das einzubettende Wasserzeichen in Form eines Strings übergeben. Nun wird an dieses String die Wasserzeichensignatur `|wm&UNIKoLa_2009|v01|` und die Größe des Wasserzeicheninhalts vorne drangehängt. Dies ist notwendig, damit beim Auslesen eines Wasserzeichens, was im Verification-Web-Service realisiert ist, das Wasserzeichen wieder erkannt wird.

GUI

Die GUI besitzt auch nur eine Klasse *AppINOUT.java*, mit deren Hilfe die Benutzeroberfläche erzeugt wird. Im Prinzip enthält diese Klasse nur eine Funktion, die für das Einbetten des Wasserzeichens wichtig ist:

mouseClicked(MouseEvent e)

In der Funktion wird ein Objekt der Klasse *INOUT.java* erzeugt und seine oben genannten Hauptfunktionen nacheinander ausgeführt. Damit wird die Benutzeroberfläche mit dem eigentlichen Wasserzeichen-Einbettungsalgorithmus verbunden. Was in dieser Funktionen genau passiert, wird bei der Implementierung erläutert.

4.2 Implementation

In dem folgenden Abschnitt wird beschrieben, wie das *Watermark Embedding Tool* implementiert wurde und welche Techniken dabei verwendet wurden. Wie schon in dem Entwurf geschrieben wurde, gibt es zwei logische Programmteile: *Watermarker* und *GUI*. Sie bestehen jeweils aus einer Klasse. Nun wird die Funktionalität der beiden Klassen in den zwei folgenden Unterabschnitten genauer erleuchten.

4.2.1 Watermarker

Wie es schon im Entwurf erwähnt wurde, enthält die Klasse *INOUT.java* die eigentliche Implementierung des *LSB-Wasserzeichenalgorithmus*. Außerdem muss die Klasse Methoden bereitstellen, mit deren Hilfe WAV-Dateien ausgelesen und die veränderten Dateien wieder gespeichert werden können. Wie dies im Einzelnen implementiert wurde, wird in den folgenden Unterabschnitten beschrieben.

4.2.1.1 WAV-Datei auf Gültigkeit überprüfen und auslesen

Für das Auslesen einer WAV-Datei muss zunächst ein Objekt der Klasse *INOUT.java* erzeugt werden. Dies geschieht in der GUI-Klasse *AppINOUT.java*. Das Objekt der Klasse wird mit dem Konstruktor der Klasse erzeugt, welcher den gewünschten Dateiname bzw. Dateipfad in Form eines Strings übergeben bekommt. Dieser String wird der globalen Variable `fileName` zugewiesen.

Nun wird die Funktion *readWave()* mit einem booleschen Wert als Rückgabeparameter aufgerufen. Als erstes wird die Datei, in die ein Wasserzeichen eingebettet werden, auf Gültigkeit untersucht. Diese Gültigkeitsüberprüfung basiert auf dem Aufbau einer WAV-Datei, der im Unterabschnitt 3.2.2.1 beschrieben wurde. Dabei wird versucht, die ASCII-Zeichenkette „WAVE“ aus dem

RIFF-Chunk auszulesen. Dies ist mit dem Paket *javax.sound.sampled* der Java Sound-API möglich. Dieses Paket ermöglicht den Zugriff auf die einzelnen Chunks einer WAV-Datei. Somit kann auch das „*riffType*“-Feld ausgelesen werden. Konnte aus ihm die Zeichenkette „*WAVE*“ extrahiert werden, wird mit dem Auslesen der gesamten Datei fortgefahren. Anderenfalls bricht die Funktion *readWave()* ab und liefert den booleschen Wert *false* zurück. Des Weiteren kann mit Hilfe der Java Sound-API die *Sample-Rate* ausgelesen werden, die beim Einbetten des Wasserzeichens benötigt wird. Die *Sample-Rate* wird der Variable `bitsPerSample` zugewiesen.

Nach der Gültigkeitsüberprüfung kann nun das eigentliche Auslesen der Datei beginnen. Dabei werden aus dem Dateistrom mit Hilfe der Byte-Stream-Klasse *FileInputStream* des *java.io*-Pakets die einzelnen Bytes der Datei ausgelesen und zu einem Byte-Array zusammengesetzt. Dadurch wird eine WAV-Datei nun als Byte-Array repräsentiert.

Danach werden die *Header*- von den *Payload*-Daten getrennt, weil beim *LSB-Algorithmus* nur die *Payload*-Daten von Bedeutung sind. Dies geschieht wie folgt: in dem Datei-Array wird nach der Zeichenkette „*data*“ gesucht, da es bekannt ist, dass nach dem „*data*“-Wort der *data-Chunk* mit den *Payload*-Daten kommt. Anhand der Position des „*data*“-Wortes wird die Größe des *Headers* und des *Payloads* ausgerechnet und entsprechend der Variablen `headerSize` und `payloadSize` zugewiesen. Falls das Wort „*data*“ nicht gefunden werden konnte, bedeutet dies, dass die Datei nicht im WAV-Format ist. Dementsprechend liefert die Funktion an dieser Stelle den booleschen Wert *false* zurück.

Wenn die Größe des *Headers* und *Payloads* bekannt ist, werden die entsprechenden Teile des Datei-Arrays in die zwei globalen Byte-Array-Variablen `headerArr` und `payloadArr` kopiert. Die Variable `payloadArr` enthält nun die wasserzeichenrelevanten *Payload*-Daten in Form eines Byte-Arrays. Das `headerArr`-Array wird später beim Speichern der Datei benötigt. Wenn die Funktion *readWave()* bis zum Ende abgearbeitet werden konnte, liefert sie einen Wert *true* zurück.

4.2.1.2 Wasserzeichen einbetten

Das Einbetten eines Wasserzeichens erfolgt in der Funktion *changeIt(byte[] wmArr)*, die ebenfalls einen booleschen Wert als Rückgabeparameter hat. Wie aus der Signatur der Funktion zu sehen ist, bekommt sie ein Wasserzeichen in Form eines Byte-Arrays übergeben. Bevor dieses Wasserzeichen eingebettet werden kann, wird überprüft, ob die Input-Datei die Gültigkeitsprüfung bestanden hat. Falls dies nicht der Fall ist, wird der Aufruf der Funktion abgebrochen und sie liefert einen *false* Wert zurück. Anderenfalls wird mit der Einbettung begonnen.

```
1  int i = 0;
2
3  for (int j = 0; j < wmArr.length; j++) {
4      for (int k = 0; k < number.length; k++) {
5
6          if (i > payloadArr.length) return false;
7
8          if ((wmArr[j] & (byte) (number[k])) == (byte) (number[k])) {
9              payloadArr[i] = (byte) ((byte) payloadArr[i] | Byte.parseByte("0000001", 2));
10
11          } else {
12              payloadArr[i] = (byte) ((byte) payloadArr[i] & Byte.parseByte("-0000010", 2));
13          }
14          i = i + bitsPerSample;
15      }
16  }
```

Listing 4.1: Einbetten eines Wasserzeichens

Die Implementation des *LSB-Wasserzeichenalgorithmus* ist in Listing 4.1 dargestellt. Dabei ist `wmArr` das Byte-Array mit dem Wasserzeicheninhalt, welches eingebettet werden soll. Die Variable

payloadArr ist das beim Auslesen erzeugte Byte-Array mit den Nutzdaten. Das Byte-Array number wurde bisher nicht erwähnt und ist in Listing 4.2 dargestellt. Im Prinzip soll mit dem number Array jedes niedrigwertigste Bit von jedem Byte im Wasserzeichen-Array auf Inhalt einer Eins oder einer Null überprüft werden.

```

1 private byte[] number =
2     {Byte.parseByte("-1000000", 2), Byte.parseByte("01000000", 2),
3     Byte.parseByte("00100000", 2), Byte.parseByte("00010000", 2),
4     Byte.parseByte("00001000", 2), Byte.parseByte("00000100", 2),
5     Byte.parseByte("00000010", 2), Byte.parseByte("00000001", 2) };

```

Listing 4.2: Array number

Nun wird der *LSB-Algorithmus* beschrieben. In jedem Schleifendurchlauf (Zeilen 3-16 in Listing 4.1) wird jeweils ein Byte des Wasserzeichen-Array genommen. In der zweiten Schleife (Zeilen 4-15) wird in Zeile 6 in jedem Schleifendurchlauf zunächst überprüft, ob die Größe des Wasserzeicheninhalts die Größe der Nutzdaten übersteigt. In den Zeilen 8 bis 13 wird mit den Bit-Operatoren von Java überprüft, an welcher Bit-Stelle des einzelnen Bytes des Wasserzeichen-Arrays eine Eins steht und an welcher Stelle eine Null steht. Abhängig von der *Sample-Rate* wird dann entsprechend am Ende jedes Bytes oder am Ende jedes zweiten Bytes des *Payload-Arrays* eine Eins oder eine Null gesetzt. Der Vorgang ist am Beispiel einer Datei mit *Sample-Rate* 8 Bits in der Abbildung 4.4 dargestellt:

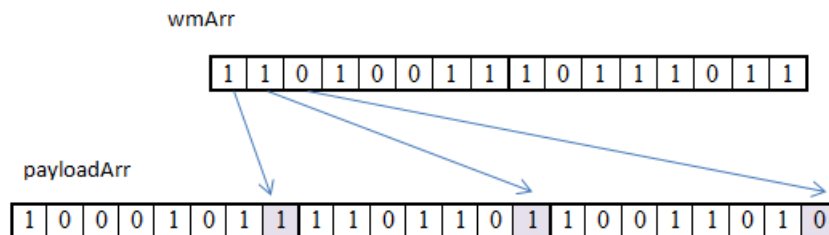


Abbildung 4.2: Beispiel

Im Allgemeinen wird jedes *Least Significant Bit* des Payload-Arrays (payloadArr) nacheinander mit jeweils einem Bit aus dem Wasserzeichen-Array (wmArr) ersetzt.

4.2.1.3 WAV-Datei speichern

Das Speichern der veränderten Datei wird in der Funktion *saveWave(String outName)* realisiert. Dabei wird zunächst überprüft, ob die Datei überhaupt gelesen wurde. Danach werden das *Header-Array* und das geänderte *Payload-Array* zu einem Byte-Array zusammengesetzt. Dieses Array wird dann mit Hilfe der Klasse *FileOutputStream* zurück in einen Dateistrom umgewandelt.

4.2.2 GUI

Im Prinzip besteht die GUI-Klasse *AppINOUT.java* aus den graphischen Komponenten der Java-Pakete *java.awt.** und *javax.swing.**. In der Funktion *mouseClicked(MouseEvent e)* wird ein Ereignis abgefangen, sobald die Maus geklickt wurde. Im Allgemeinen wird ja nach dem geklickten Button durch Aufruf der oben genannten Funktion eine WAV-Datei gelesen, geändert oder gespeichert.

4.3 Bedienung

In diesem Abschnitt wird die Arbeitsweise des Programms ähnlich einem Benutzerhandbuch beschrieben. Das Verwenden des Programms kann man in vier unten genannte Arbeitsschritte unterteilen.

4.3.1 Start des Programms

Das Starten des Programms erfolgt durch die ausführbare JAR-Datei WETool.jar. Diese kann durch Doppelklick oder Kommandozeilenbefehl `java -jar WETool.jar` ausgeführt werden. Im Allgemeinen setzt das Programm keine Administratorrechte voraus. Es wird jedoch empfohlen dieses mit Administratorrechten zu nutzen, um problemlosen Zugriff auf die WAV-Dateien zu ermöglichen¹. Nachdem die JAR-Datei ausgeführt wurde, bekommt der Benutzer das GUI-Fenster zu sehen, welches eine feste Größe besitzt (Abbildung 4.1):

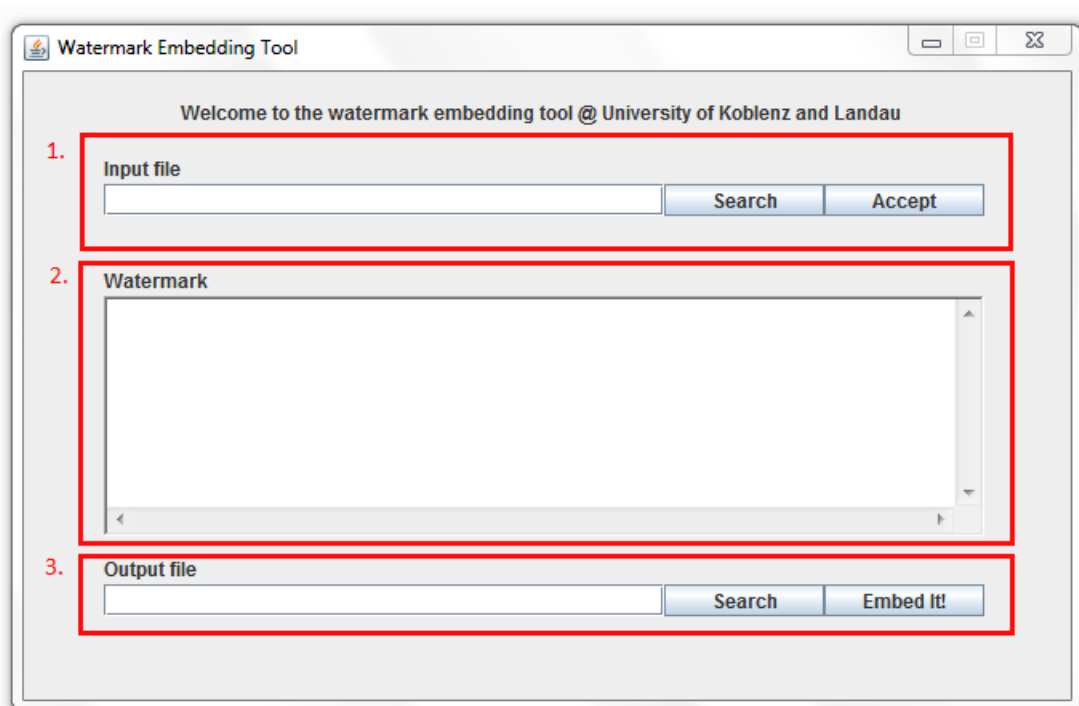


Abbildung 4.3: Watermark Embedding Tool

Die Bedienung des Programms kann mit drei Schritten beschrieben werden:

1. eine WAV-Datei wird ausgewählt und auf Gültigkeit überprüft
2. der Wasserzeicheninhalt wird in Form eines Textes eingegeben
3. das Wasserzeichen wird in die ausgewählte Datei eingebettet und gespeichert

¹Anderenfalls könnte der Zugriff auf eine Datei verweigert werden.

4.3.2 Auswahl der Datei

Nachdem nun das GUI-Fenster gestartet ist, kann eine WAV-Datei ausgewählt werden, in die ein Wasserzeichen eingebettet werden soll. Für die Auswahl der Datei gibt es zwei Möglichkeiten:

1. Der Dateiname kann mit dem dazugehörigen Dateipfad direkt in das Input-File-Feld eingetippt werden. Ohne Pfadangabe werden nur Dateien innerhalb des Programmverzeichnisses akzeptiert.
2. Eine Datei kann mit Hilfe eines File-Browsers ausgewählt werden. Mit einem Klick auf den -Button erscheint ein Dialogfenster:

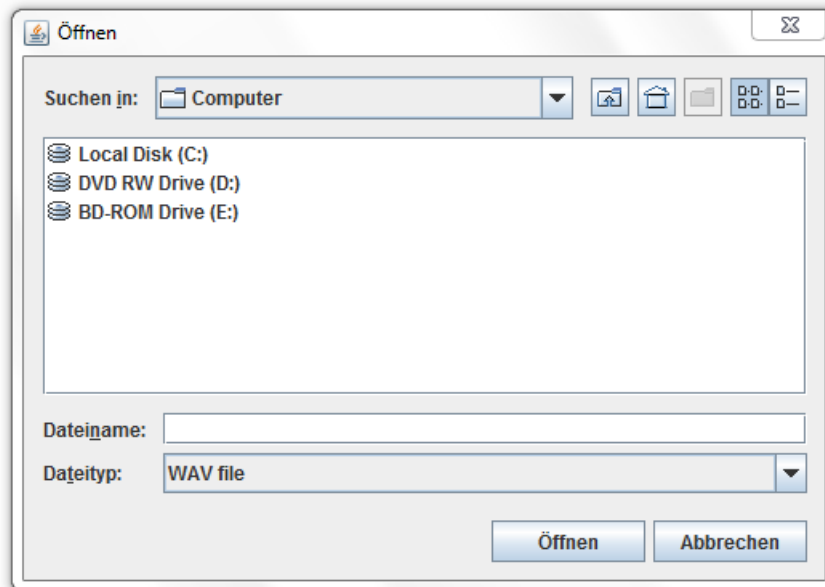
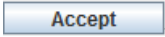
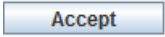


Abbildung 4.4: File-Browser

Der Dateibrowser ist standardmäßig so eingestellt, dass nur die WAV-Dateien angezeigt werden. Dies erleichtert die Suche nach einer WAV-Datei. Der Dateityp kann aber auf „Alle Dateien“ geändert werden.

Falls der Dateiname und eventuell der Dateipfad in das Input-File-Feld eingetragen wurden, muss die Datei vor der Eingabe vom Wasserzeicheninhalt auf Gültigkeit überprüft werden. Dies geschieht mit einem Klick auf den -Button. Wenn bei der Überprüfung Fehler aufgetreten sind, werden sie unterhalb vom Input-file-Feld angezeigt:

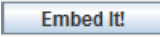
- `could not get audio input stream from input file` bedeutet, dass die eingegebene Datei entweder keine WAV-Datei oder eine WAV-Datei in einem nicht unterstützten Format ist
- `C:\notExist.wav (The system cannot find the file specified)` erscheint, wenn die eingegebene Datei nicht existiert oder der Pfad zu dieser Datei falsch ist.

Jedes Mal nach der Auswahl einer neuen WAV-Datei muss sie durch Betätigung des -Buttons auf Gültigkeit überprüft werden. Falls die Datei die Gültigkeitsprüfung nicht bestanden hat, wird der Wasserzeicheninhalt nicht eingebettet. Die Meldung `Accepted !!!` bedeutet, dass die Überprüfung einer WAV-Datei erfolgreich abgeschlossen ist und das Einbetten von einem Wasserzeichen nun beginnen kann.


4.3.3 Eingabe des Wasserzeicheninhalts

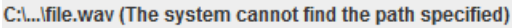
Der Wasserzeicheninhalt wird in dem Feld Watermark (siehe Abbildung 4.1, 2. Abschnitt) eingetippt. Im Prinzip kann ein beliebiger Text eingegeben werden. Die Wasserzeichensignatur |wm&UNIKoLa_2009|v01| darf im Text vorkommen, muss aber nicht. Im Allgemeinen ist die Textgröße bei der Eingabe nicht begrenzt. Wenn der Text jedoch mehr Platz in Anspruch nimmt, als in der Datei für ein Wasserzeichen reserviert ist, dann wird der Einbettvorgang nicht ausgeführt. Dies passiert jedoch ganz selten, da schon eine 3 Mbyte große WAV-Datei bis zu 24000 Zeichen transportieren kann.

4.3.4 Einbetten und Speichern

Bevor das Wasserzeichen in die Datei eingebettet werden kann, muss zunächst die Output-Datei bestimmt werden. Hierzu hat man die Möglichkeit den Namen der Ausgabedatei direkt in das Output-File-Feld einzugeben. Alternativ hierzu kann auch ein Filebrowser genutzt werden. Um das Wasserzeichen nun einzubetten, muss der -Button betätigt werden. Dabei wird das Wasserzeichen in der im Input-File-Feld ausgewählten Datei eingebettet und gespeichert. Beim Speichern sind folgende Fälle zu unterscheiden.

1. Die im Output-File-Feld eingegebene Datei existiert bereits:

In diesem Fall wird die Meldung  angezeigt.

- Die ausgewählte Datei ist eine WAV-Datei: sie wird mit der wasserzeichenbehafteten Input-Datei überschrieben.
 - Die ausgewählte Datei ist keine WAV-Datei: die Kopie von der Input-Datei mit dem eingebetteten Wasserzeichen wird unter dem Namen der Output-Datei als eine WAV-Datei gespeichert.
2. Die in Output-File-Feld eingegebene Datei existiert nicht: In diesem Fall wird eine Kopie der wasserzeichenbehafteten Input-Datei erstellt und mit dem angegebenen Namen gespeichert.
 3. Falls der Pfad zur Datei ungültig ist, wird die Meldung  angezeigt.

Wenn nun das Wasserzeichen erfolgreich eingebettet wurde und die wasserzeichenbehaftete Datei gespeichert werden konnte, wird die Meldung  angezeigt.

4.3.5 Verbesserungsvorschläge

An dieser Stelle wird erwähnt, was in der nächsten Version des Programms bezüglich GUI anders gemacht werden könnte.

Als erstes müsste die zwei Felder Input-File-Feld und Output-File-Feld durch ein einziges Dateieingabefeld ersetzt werden. Obwohl die aktuelle Version mehr Freiheit beim Speichern einer wasserzeichenbehafteten Datei bietet, kann dies auch verwirrend sein.

Als nächstes könnte beim Watermark-Feld ein Counter einbaut werden, welcher zeigt, wie viele Zeichen noch theoretisch eingebettet werden könnten. Des Weiteren könnte eine Meldung angezeigt werden, wenn ein Wasserzeichen zu groß für eine Datei ist. Bei der aktuellen Version wird das Wasserzeichen einfach nicht eingebettet, wenn es zu groß ist. Dies ist nicht besonders schlimm,

da die maximal mögliche Größe des Wasserzeichens selten erreicht wird. Es ist aber auch nicht besonders nutzerfreundlich.

Als letztes kann die Fenstergröße variabel gemacht werden. Dazu müssten die Layout-Manager von Java verwendet werden. Eine variable Fenstergröße könnte die Benutzerfreundlichkeit erhöhen.

Kapitel 5

Verification-Web-Service

Das Hauptziel dieser Bachelorarbeit war, einen *Web-Service* zu entwickeln, mit welchem die ferngesteuerte Überprüfung einer Datei auf ein Wasserzeichen ermöglicht werden konnte. Als Trägermedien sollten dabei die WAV-Dateien fungieren und es sollte der Least-Significant-Bit-Algorithmus zum Einsatz kommen. In diesem Kapitel wird nun die Umsetzung eines Web-Services, basierend auf dem Abschnitt 3.3, beschrieben. Dabei soll mit dem Abschnitt 5.1 zunächst ein grober Überblick über die Architektur des Web-Services gegeben werden. Danach wird im Abschnitt 5.2 die technische Umsetzung beschrieben. Schließlich wird im Abschnitt 5.3 die Bedienung des Programms für die Benutzer beleuchtet.

5.1 Entwurf

Um den *Web-Service* testen zu können, wurde ein *Client* entwickelt. Der *Web-Service* und der dazu gehöriger *Client* wurden mit der Programmiersprache Java entwickelt. Das Erstellen des *Web-Services* wurde mit der Java-API JAX-WS realisiert, welche in dem Java Development Kit (JDK) bereits seit Version 6 integriert ist. Alternativ könnte Apache Axis2 (Apache eXtensible Interaction System) verwendet werden, welches das Erstellen der SOAP-basierten *Web-Services* ebenfalls ermöglicht. Die Entscheidung JAX-WS zu nutzen, ist gefallen, weil damit ein *Web-Service* mit der geforderten Funktionalität auf eine einfache Weise ohne zusätzliche Programme erstellt werden kann. Im Mittelpunkt von JAX-WS stehen die entfernten Methodenaufrufe. Dabei beruht die Kommunikation zwischen dem *Web-Service* und dem *Client* auf dem sogenannten Proxy-Pattern.

Das Proxy-Pattern, oder auch Stellvertreter-Pattern genannt, funktioniert wie folgt: Aus der Schnittstellenbeschreibung eines *Web-Services* wird auf der Seite des *Clients* ein Proxy-Objekt erzeugt. Der *Client* ruft nun eine Funktion dieses Objektes auf. Das Proxy-Pattern erweckt den Anschein, dass diese Funktion lokal aufgerufen wird. In Wirklichkeit wird der Funktionsaufruf in Form einer Anfrage an den *Web-Service* geschickt. Der *Web-Service* nimmt diese Anfrage entgegen, ruft die lokale Service-Methode auf und sendet das Ergebnis des Aufrufes zurück an den *Client*. Die Verwaltung des Proxy-Objekts und die Kommunikation zwischen *Client* und *Web-Service* werden dabei von der JAX-WS-Laufzeitumgebung übernommen.

5.1.1 Anforderungen

In diesem Unterabschnitt werden die Anforderungen an den *Web-Service* und den dazu gehörigen *Client* zusammengefasst, welche bei den Gesprächen mit Andreas Kasten und Helge Hundacker, im Laufe der Bachelorarbeit und aus den Anforderungen an die *Web-Services* im Allgemeinen

entstanden sind. Dabei wird zwischen [Kann]-Anforderungen, die realisiert werden konnten, und [Pflicht]-Anforderungen, die realisiert werden sollten, unterschieden:

Web-Service

- Bei der Implementation von dem Web-Service sollen offene, programmiersprachen- und plattformunabhängige Standards verwendet werden. [Pflicht]
- Der Web-Service soll eine Funktionalität bieten, mit der ein Least-Significant-Wasserzeichen in eine WAV-Datei eingebettet werden kann. [Pflicht]
- Der Web-Service soll eine grafische Benutzerschnittstelle (GUI) haben. [Kann]
- Der Web-Service soll seine Schnittstellenbeschreibung in Form eines WSDL-Dokuments veröffentlichen können. [Pflicht]
- In den Web-Service soll ein HTTP-Server integriert werden. [Pflicht]
- Die Schnittstellenbeschreibung des Web-Services soll mit dem HTTP-Server veröffentlicht werden können. [Pflicht]
- Der HTTP-Server soll auf SOAP-Anfragen von dem Client reagieren können. [Pflicht]
- Der HTTP-Server soll SOAP-Antworten an den Client senden können. [Pflicht]
- Der Web-Service soll unter einer Netzwerkadresse (URI) erreichbar sein. [Pflicht]

Client

- Der Client soll eine grafische Benutzeroberfläche haben. [Kann]
- Der Benutzer muss die zu überprüfende WAV-Datei auswählen können. [Pflicht]
- Falls die Überprüfung der Datei fehlgeschlagen ist, soll dem Benutzer eine entsprechende Meldung angezeigt werden. [Pflicht]
- Eine entsprechende Meldung soll angezeigt werden, falls der Übertragungsvorgang länger dauert. [Kann]
- Falls die Überprüfung erfolgreich war, soll eine entsprechende Meldung erscheinen. [Pflicht]
- Falls die vom Benutzer ausgewählte Datei ein Wasserzeichen enthält, sollen entsprechende Wasserzeicheninformationen, wie die Wasserzeichensignatur und Wasserzeichengröße angezeigt werden. [Kann]
- Falls das Wasserzeichen vorhanden ist, soll der Wasserzeicheninhalt angezeigt werden. [Pflicht]

5.1.2 Architektur

Die gesamte Systemarchitektur besteht im Allgemeinen aus der *Web-Service*- und der *Client*-Systemkomponente. Das Zusammenspiel dieser Komponenten ist in der Abbildung 5.1 dargestellt. Dabei wurde auf das Dienstverzeichnis verzichtet, da das System nur einen Web-Server hat. Als nächstes werden die einzelnen Komponenten kurz beschrieben, um danach die Beschreibung der Beziehungen dieser Komponenten besser verstehen zu können.

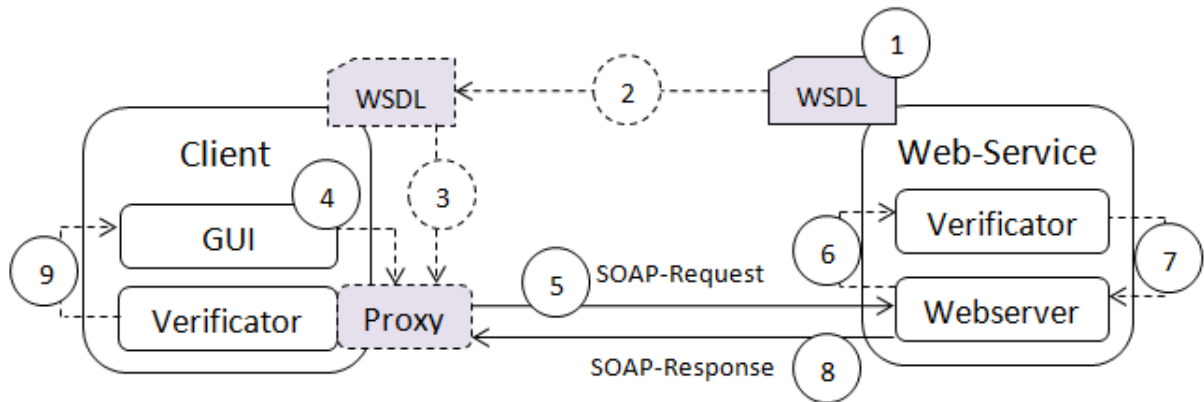


Abbildung 5.1: *Web-Service*-Architektur

5.1.2.1 Web-Service

Der *Web-Service* ist die Hauptkomponente des Systems. Auf ihm wird die Überprüfung einer WAV-Datei auf Inhalt eines Wasserzeichens realisiert. Im Prinzip besteht der *Web-Service* aus zwei Teilkomponenten: *Webserver* und *Verificator* (Abbildung 5.1).

Webserver

Der Webserver ist die Teilkomponente, mit der ein *Web-Service* veröffentlicht wird. Des Weiteren ermöglicht der Webserver die Kommunikation mittels SOAP-Nachrichten zwischen dem *Web-Service* und dem *Client* mit Hilfe des HTTP-Transportprotokolls. Im Java Development Kit 6 ist bereits ein kleiner Webserver integriert, der für die Implementation des Systems ausreicht. Als Alternative könnte Apache Tomcat verwendet werden.

Verificator

In der Verificator-Teilkomponente wird die eigentliche Überprüfung einer WAV-Datei durchgeführt. Eine Methode des Verificators bekommt eine WAV-Datei als Parameter übergeben. Als Antwort liefert sie dann das Wasserzeichen zurück, falls dieses in der Datei enthalten ist.

5.1.2.2 Client

Der *Client* wurde im Prinzip nur dazu entwickelt, um den *Web-Service* zu testen. Er besteht ebenfalls aus zwei Teilkomponenten: GUI und Verificator. Die Aufgaben des *Client*s sind, dem Benutzer die Möglichkeit zu geben, die gewünschte Datei auszuwählen, einen Aufruf der entfernten Überprüfungsmethode auszulösen und das Ergebnis dem Benutzer anzuzeigen. Die Interaktion mit dem Benutzer erfolgt mittels einer graphischen Benutzeroberfläche (GUI). Die entfernte Methode wird in der Verificator-Teilkomponente aufgerufen.

5.1.2.3 Ablauf der Kommunikation

In diesem Unterunterabschnitt wird das Zusammenspiel der Systemkomponenten beleuchtet, indem die Arbeitsschritte bei der Übertragung einer Datei anhand der Abbildung 5.1 beschrieben werden:

1. Der *Web-Service* stellt seine Schnittstellenbeschreibung in Form eines WSDL-Dokuments mittels eines integrierten Webservers zur Verfügung und wartet auf eine Anfrage vom *Client*.
2. Von der Seite des *Clients* wird auf diese WSDL-Datei zugegriffen.
3. Dabei werden aus der Schnittstellenbeschreibung die zu der Kommunikation mit dem *Web-Service* notwendigen Komponenten generiert. Unter anderem wird das Interface für das Proxy-Objekt erzeugt. Die Schritte 2 und 3 werden nur bei der ersten Verbindung zum *Web-Service* durchgeführt.
4. Wenn der Benutzer das Ereignis zur Überprüfung ausgelöst hat, wird ein Proxy-Objekt erzeugt und seine Überprüfungs-methode aufgerufen.
5. Der Aufruf der Proxy-Methode wird von der Java-Laufzeitumgebung in eine SOAP-Nachricht verpackt und zum *Web-Service* geschickt.
6. Diese Daten werden von dem Webserver empfangen, von der Java-Laufzeitumgebung entpackt und zum Verificator weitergeleitet, wo die eigentliche Überprüfung stattfindet.
7. Das Ergebnis der Überprüfung wird wieder in eine SOAP-Nachricht verpackt und dem Webserver übergeben.
8. Die SOAP-Nachricht wird zurück an den *Client* geschickt.
9. Schließlich wird die Nachricht von der Java-Laufzeitumgebung entpackt. Die Proxy-Methode liefert nun das Ergebnis der Überprüfung, welches dem Benutzer in der GUI angezeigt wird.

5.2 Implementierung

In diesem Abschnitt wird die programmtechnische Umsetzung der im Entwurf vorgestellten Komponenten beschrieben.

5.2.1 Web-Service

Der *Web-Service* besteht aus drei Klassen (Abbildung 5.2):

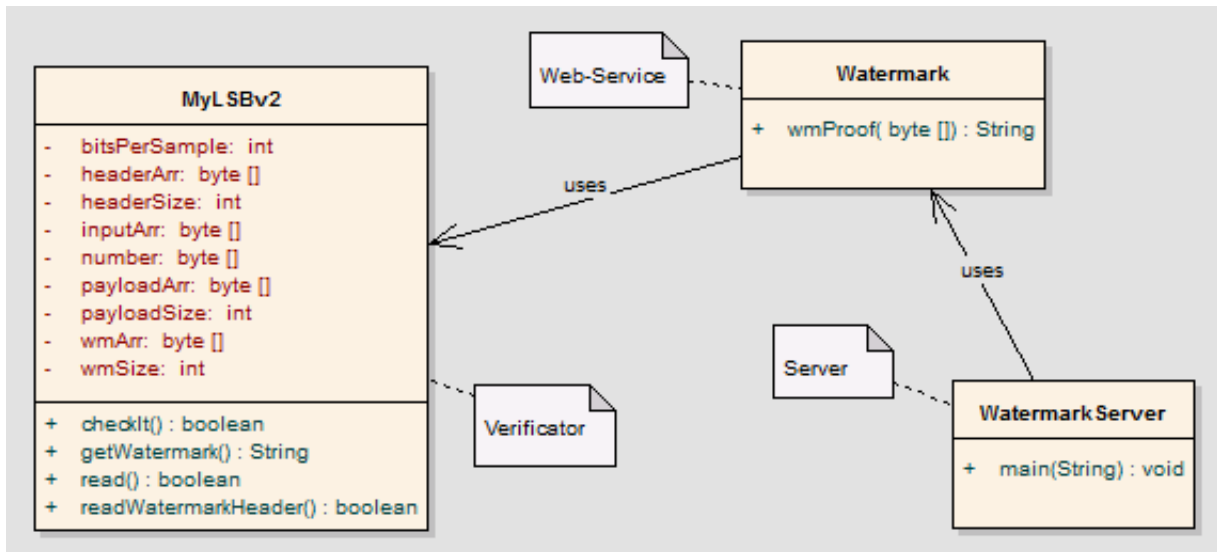


Abbildung 5.2: Web-Service-Klassendiagramm

Im Prinzip ist *MyLSBv2.java* eine Klasse, in der eine Datei auf ein Wasserzeichen überprüft wird. Die Klasse *Watermark.java* ist der eigentliche *Web-Service* und die Klasse *WatermarkServer.java* ist ein *Server*, mit dem der *Web-Service* veröffentlicht wird. Nun werden diese Klassen in den folgenden Unterunterabschnitten genauer beschrieben.

5.2.1.1 Verificator-Klasse

In der Klasse *MyLSBv2.java* wird aus einer WAV-Datei ein Wasserzeichen, welches mit dem *Watermark Embedding Tool* integriert wurde, ausgelesen. Beim Erzeugen des Objektes der Klasse bekommt der Konstruktor eine WAV-Datei in Form eines Byte-Arrays übergeben. Dieses Array wird einer Klassen-Variable `inputArr` zugewiesen. Des Weiteren folgen zwei Abarbeitungsschritte:

Input-Array lesen

Bevor das Wasserzeichen ausgelesen werden kann, müssen die *Header*- und *Payload*-Daten voneinander getrennt werden, da bei dem Auslesen des Wasserzeichens nur die *Payload*-Daten benötigt werden. Dies geschieht in der Methode `read()`. Dabei wird wie bei dem *Watermark Embedding Tool* nach der Zeichenkette `data` in dem Input-Array gesucht (Listing 5.1).

```

1  for (int i = 0; i < inputArr.length - 4; i++) {
2
3      bufferArr[0] = (byte) inputArr[i];
4      bufferArr[1] = (byte) inputArr[i + 1];
5      bufferArr[2] = (byte) inputArr[i + 2];
6      bufferArr[3] = (byte) inputArr[i + 3];
7      String s = new String(bufferArr);
8
9      if (s.contentEquals("data")) {
10         headerSize = i + 8;
11         break;
12     } else if (i == (inputArr.length - 5)) {
13         return false;
14     }
15 }

```

Listing 5.1: Suchen nach der Zeichenkette `data`

Bei jedem Schleifendurchlauf werden jeweils 4 Bytes des Input-Arrays in ein Buffer-Array kopiert. Jedes Mal wird das Buffer-Array in einen String umgewandelt. Wenn die Zeichenkette „data“ gefunden werden konnte, wird die Position des Wortes notiert und die Schleife abgebrochen. Andernfalls läuft die Schleife, bis das Ende des Input-Arrays erreicht wurde.

Anhand der Position von dem „data“-Wort werden die *Header*- und *Payload*-Größen ausgerechnet und in Variablen `headerSize` und `payloadSize` gespeichert. Nun werden die `headerArr`- und `payloadArr`-Arrays mit den entsprechenden Daten gefüllt. Wenn die Funktion bis zum Ende abgearbeitet werden konnte, wird der boolesche Wert `true` zurückgeliefert.

Wasserzeichen auslesen

Wenn die Nutzdaten aus dem Input-Array extrahiert werden konnten, wird mit dem Auslesen des Wasserzeichens fortgefahren. Dies geschieht in der Methode `checkIt()`. Bevor das Wasserzeichen ausgelesen wird, wird zunächst überprüft, ob sich ein Wasserzeichen in dem Input-Array befindet. Dazu wird eine klasseninterne Methode `readWatermarkHeader()` aufgerufen.

Diese Methode versucht aus dem Input-Array die Wasserzeichensignatur `|wm&UNIKoLa_2009|v01|` auszulesen. Falls die Signatur ausgelesen werden konnte, wird die Wasserzeichengröße ausgelesen, die im Input-Array nach der Signatur kommt. Andernfalls wird `false` zurückgeliefert.

Wenn die Signatur und Größe des Wasserzeichens erfolgreich extrahiert werden konnten, wird nun der gesamte Wasserzeicheninhalt ausgelesen (Listing 5.2).

```
1  int i = 0;
2  wmArr = new byte[wmSize];
3  for (int j = 0; j < wmArr.length; j++) {
4      for (int k = 0; k < number.length; k++) {
5          if ((payloadArr[i] & Byte.parseByte("00000001", 2)) == Byte
6              .parseByte("00000001", 2)) {
7
8              wmArr[j] = (byte) (wmArr[j] | (byte) number[k]);
9          }
10         i = i + bitsPerSample;
11     }
12 }
```

Listing 5.2: Wasserzeichen auslesen

Zunächst wird ein leeres Byte-Array initialisiert (Zeile 2 in Listing 5.2), in das das Wasserzeichen gespeichert werden soll. In der äußeren Schleife wird jedes Element dieses Arrays genommen. Danach wird in der inneren Schleifen in Abhängigkeit von der *Sample-Rate* überprüft, ob in dem *Payload*-Array an der Least-Significant-Bit-Stelle jedes oder jedes zweiten Bytes eine Null oder eine Eins steht (Zeile 5-6). Wenn eine Eins gelesen werden konnte, wird sie in das anfangs leere `wmArr`-Array geschrieben (Zeile 8). Wenn eine Null gelesen wurde, wird eine Bit-Stelle in dem `wmArr`-Array übersprungen.

In der Abbildung wird das Auslesen des Wasserzeichens am Beispiel einer Datei mit 8 Bit pro Sample dargestellt.

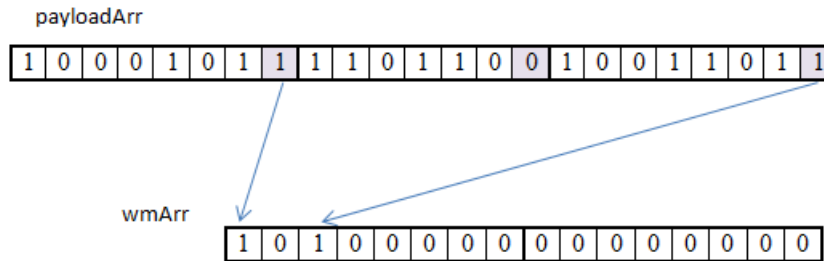


Abbildung 5.3: Beispiel

Im Allgemeinen wird jedes *Least Significant Bit* des *Payload*-Arrays (`payloadArr`) in ein anfangs leeres Array (`wmArr`) gespeichert.

Schließlich liefert die Methode `getWatermark()` das Wasserzeichen-Array `wmArr` als String zurück.

5.2.1.2 Web-Service-Klasse

Bei der Definition eines *Web-Services* mit JAX-WS werden Annotationen verwendet, die sich im Paket `javax.jws` befinden. Ein *Web-Service* ist dabei eine einfache Klasse, die eine Klassen-Annotation `@WebService` besitzen muss. Die Annotation `@WebMethod` macht eine Methode der Klasse zu einer *Web-Service-Methode*. Die Annotation `@SOAPBinding(style = Style.RPC)` sagt aus, dass der *Web-Service* mittels SOAP-Nachrichten kommuniziert und dass es sich dabei um der entfernten Funktionsaufrufe handelt.

Im Prinzip besitzt die *Web-Service-Klasse* `Watermark.java` nur eine *Web-Service-Methode* `wm-Proof(byte[] inputArr)`. Diese Methode bekommt eine WAV-Datei in Form eines Byt-Arrays übergeben. In der Methode wird ein Objekt der Klasse `MyLSBv2.java` erzeugt und Methoden der Klasse nacheinander aufgerufen. Schließlich liefert die *Web-Service-Methode* ein Wasserzeichen zurück. Im Allgemeinen bietet die *Web-Service-Methode* dem Client die Funktionalität des *Web-Services*.

5.2.1.3 Server-Klasse

In der Klasse `WatermarkServer.java` wird ein kleiner in Java integrierter Webserver erzeugt, welcher die Anfragen von einem Client entgegennimmt und sie weiterleitet. Die Server-Infrastruktur wird durch die JAX-WS-Umgebung verwaltet. Dabei wird der Server gestartet, indem die Funktion `publish` aufgerufen wird. Eine Instanz der *Web-Service-Klasse* `Watermark.java` wird erzeugt und an eine Adresse gebunden. In diesem Fall lautet die Adresse `http://ba-vkeppel.it-risk.iwvi.uni-koblenz.de/watermark`, da der *Verification-Web-Service* zurzeit an einem Rechner der Universität Koblenz läuft.

Der Webserver veröffentlicht den *Web-Service* und bietet eine WSDL-Datei mit der Schnittstellenbeschreibung unter der Adresse

`http://ba-vkeppel.it-risk.iwvi.uni-koblenz.de/watermark?wsdl an.`

5.2.2 Client

Bei dem Client müssen zunächst die für die Kommunikation mit dem *Web-Service* benötigten Komponenten aus der WSDL-Beschreibungsdatei generiert werden. Dies wird mit dem JDK 6 Tool `wsimport` gemacht.

```

- <definitions targetNamespace="http://wmServer/" name="WatermarkService">
  <types/>
  - <message name="wmProof">
    <part name="arg0" type="xsd:base64Binary"/>
  </message>
  - <message name="wmProofResponse">
    <part name="return" type="xsd:string"/>
  </message>
  - <portType name="Watermark">
    - <operation name="wmProof">
      <input message="tns:wmProof"/>
      <output message="tns:wmProofResponse"/>
    </operation>
  </portType>
  - <binding name="WatermarkPortBinding" type="tns:Watermark">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    - <operation name="wmProof">
      <soap:operation soapAction="">
        - <input>
          <soap:body use="literal" namespace="http://wmServer"/>
        </input>
        - <output>
          <soap:body use="literal" namespace="http://wmServer"/>
        </output>
      </operation>
    </binding>
  - <service name="WatermarkService">
    - <port name="WatermarkPort" binding="tns:WatermarkPortBinding">
      <soap:address location="http://ba-vkeppel.it-risk.iwvi.uni-koblenz.de/watermark"/>
    </port>
  </service>
</definitions>

```

Abbildung 5.4: WSDL-Dokument

Das WSDL-Dokument des *Verification-Web-Services* ist in der Abbildung 5.2 dargestellt. Dieses zeigt, dass der *Web-Service* mittels der SOAP-Nachrichten *wmProof* und *wmProofResponse* kommuniziert. Dabei empfängt der *Web-Service* die Nachrichten von dem Typ *wmProof* mit einem Byte-Array als Übergabeparameter und sendet die Nachrichten von dem Typ *wmProofResponse* mit einem String als Rückgabeparameter.

Nun werden unter anderem die Klasse *WatermarkService.java* und das Interface *Watermark.java* anhand der WSDL-Datei erzeugt. Im Prinzip enthält die Klasse *WatermarkService.java* die *Web-Service*-Informationen und ermöglicht somit den Zugriff auf die entfernte Methode *wmProof* des *Web-Services*.

Um den *Web-Service* nutzen zu können, wird nun in der Methode *run(String fileName)* der *Client*-Klasse *WatermarkClient.java* zunächst ein Proxy-Objekt *watermark* aus dem Interface *Watermark.java* erzeugt. Nun kann die Web-Methode *wmProof* aufgerufen werden, als wäre sie eine lokale Methode: *watermark.wmProof(inputArr)*. Dabei wird dieser Methode eine WAV-Datei in Form eines Byte-Arrays als Parameter übergeben.

Nun wird mit der Klasse *AppClient.java* die graphische Benutzeroberfläche erzeugt und die *run(String fileName)*-Methode aufgerufen. Dabei wird der Aufruf dieser Methode in einen neuen Thread verlegt, da sonst der *EventDispatchThread* bis zum Erhalt der Antwort des Servers blockiert wäre, was zur Konsequenz hätte, dass die GUI blockiert wäre.

5.3 Beschreibung

In dem folgenden Abschnitt wird die Bedienung des Programms einem Benutzer deutlich gemacht. Das System besteht aus zwei Programmteilen, die von einem Benutzer bedient werden

können.

5.3.1 Web-Service

Bevor die Arbeit mit einem *Client* beginnen kann, muss der Server gestartet sein, mit dem ein *Client* dann kommunizieren kann.

Der *Web-Service* hat keine Benutzeroberfläche und kann mit einem Kommandozeilenbefehl `java -jar server.jar` oder mit einem Doppelklick auf JAR-Datei gestartet werden. Die Datei `server.jar` kann sowohl auf einem lokalen als auch auf einem entfernten Rechner ausgeführt werden. Standardmäßig besitzt der *Web-Service* eine feste Adresse `http://ba-vkeppel.it-risk.iwvi.uni-koblenz.de/watermark`, da er sich auf einer entfernten Maschine der Universität Koblenz befindet. Unter dieser Adresse ist er für die *Clients* erreichbar. Wenn der *Web-Service* auf einem anderen Rechner gestartet werden soll, muss die Adresse entsprechend angepasst werden. Dies geschieht, indem das Adressen-Attribut einer internen Klasse geändert wird. Falls der *Web-Service* gestartet ist, kann nun ein *Client* auf ihn zugreifen. Zu den Testzwecken ist der *Web-Service* an einem Rechner der Universität bereits gestartet worden und ist unter der oben erwähnten Adresse zu erreichen. Wenn der *Web-Service* mit dem Kommandozeilenbefehl gestartet wurde, kann er mit Tastenkombination "Strg + C" beendet werden.

5.3.2 Client

Der *Client* wird durch einen Doppelklick auf die JAR-Datei `wmClient.jar` oder mit Kommandozeilenbefehl `>java -jar wmClient.jar` gestartet. Nach dem Ausführen der Datei, bekommt der Benutzer eine Benutzeroberfläche zusehen, welche eine feste Größe (Abbildung 5.1):

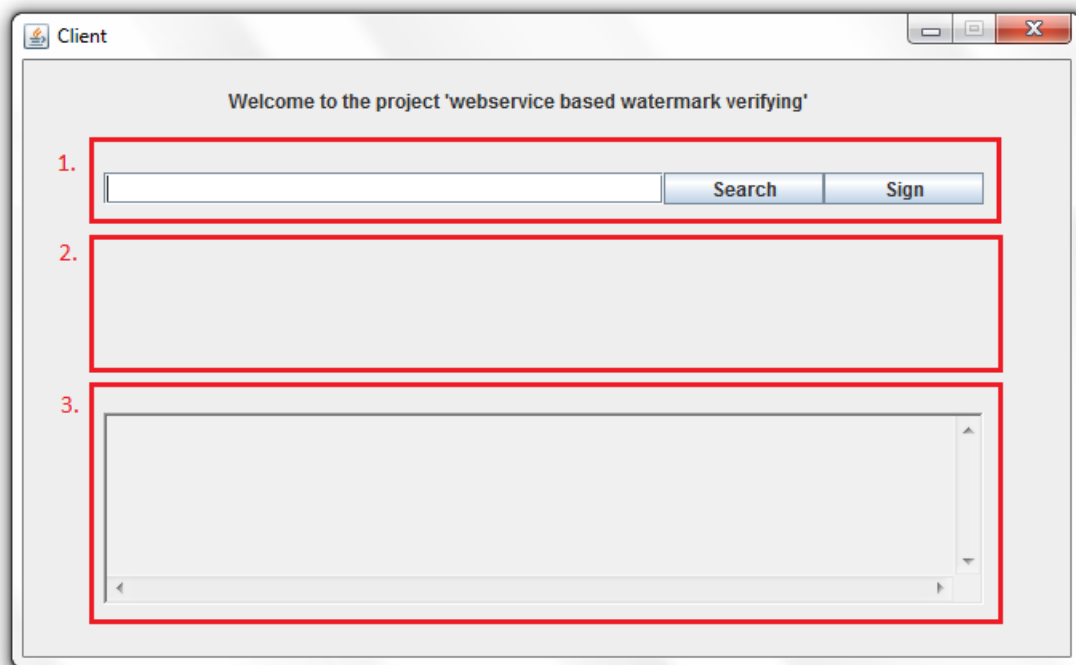




Abbildung 5.5: *Client*

Die Benutzeroberfläche kann in drei Abschnitte aufgeteilt sein:

1. In diesem Abschnitt wird eine WAV-Datei ausgewählt, welche auf Inhalt eines Wasserzeichens überprüft werden soll. Die Auswahl der Datei kann durch direktes Eintippen des Dateinamens und zugehörigen Pfades erfolgt werden. Wenn der Dateipfad nicht angegeben ist, werden nur die Dateien innerhalb des Programmverzeichnis erkannt. Des Weiteren kann die Datei mit Hilfe des Filebrowsers gesucht werden, welcher schon aus Kapitel 4 Watermark Embedding Tool bekannt ist. Mit dem Klick auf den  – Button beginnt die Überprüfung der Datei.
2. Nach dem Klick auf den  – ButtonIn wird dem zweiten Abschnitt zunächst der Übertragungsstatus angezeigt:

Connecting to Server . . .
(this can take several minutes)

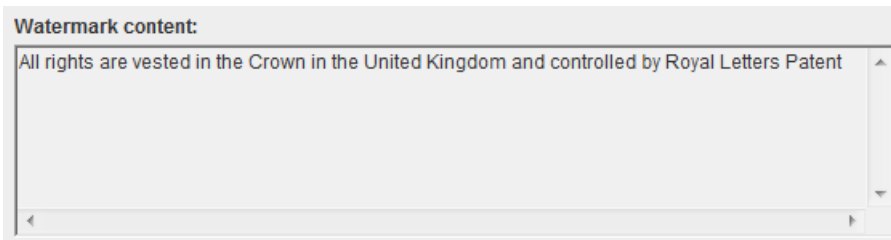
Im Falle einer erfolgreichen Wasserzeichenüberprüfung werden folgende Meldungen angezeigt:

- **No watermark here** , falls die ausgewählte Datei kein Wasserzeichen enthält.
- **File contains a watermark** , falls das Wasserzeichen gefunden werden konnte.
- **Service not reachable. Timeout** , erscheint, wenn die Datei nicht gefunden werden konnte, der Server offline ist oder die Dateiübertragung zu viel Zeit in Anspruch genommen hat.

Wenn das Wasserzeichen extrahiert werden konnte, werden solche Wasserzeicheninformationen wie Signatur, Version und die Größe des Wasserzeichens in Bytes angezeigt:

Signatur: wm&UNIKoLa_2009
Version: v01
Size: 126

3. Wenn ein Wasserzeichen in der vom Benutzer ausgewählten Datei gefunden werden konnte, wird in dem dritten Abschnitt der Wasserzeicheninhalt in Form eines Textes ausgegeben:



Danach kann eine weitere Datei ausgewählt werden, bei der eine Wasserzeichenüberprüfung durchgeführt werden soll.

Kapitel 6

Zusammenfassung und Ausblick

In diesem Kapitel werden zunächst die wichtigsten Aspekte der Bachelorarbeit zusammengefasst. Danach wird ein Ausblick gegeben, in dem beschrieben wird, welche Erweiterungsmöglichkeiten bestehen.

Zusammenfassung

Das Ziel dieser Bachelorarbeit war einen Web-Service zu entwickeln, mit dem eine WAV-Datei auf ein Least-Significant-Bit-Wasserzeichen überprüft wird. Die Idee der service-orientierten Überprüfung wurde von dem URM-Projekt initiiert, welches an der Universität Koblenz-Landau von der Arbeitsgruppe Grimm entwickelt wird.

Bei der Entwicklung des Web-Services mussten zunächst die folgenden Grundlagen erarbeitet werden: Aufbau einer WAV-Datei, der Least-Significant-Bit-Algorithmus und die Web-Service-Grundlagen. Bevor der Web-Service realisiert werden konnte, musste ein Programm entwickelt werden, welches ein Wasserzeichen in eine WAV-Datei integriert. Dabei ist das Watermark Embedding Tool entstanden. Schließlich wurde der Verification-Web-Service implementiert. Damit dieser getestet werden konnte, wurde ein dazu gehöriger Client entwickelt.

Mit Hilfe des Verification-Web-Services kann eine WAV-Datei auf Existenz eines Wasserzeichens ferngesteuert überprüft werden. Dabei ist der Web-Service programmiersprachen- und plattformunabhängig. Deshalb ist man bei der Realisierung des Clients an keine bestimmte Programmiersprache oder Betriebssystem angewiesen.

Ausblick

In dieser Arbeit sollte nur ein Prototyp eines Web-Services entwickelt werden. Daher bietet sich viel Raum für Erweiterungen. So könnten neben dem Least-Significant-Bit-Algorithmus weitere Algorithmen implementiert werden. Mögliche Kandidaten wären die im Unterabschnitt 3.1.2 erwähnten Echo-Wasserzeichen und MPEG2-Scale-Factor-Algorithmus. Eine weitere Ergänzung des Programms bestünde in der Unterstützung verschiedener Audioformate. Da die WAV-Dateien für die Netzwerkübertragung eher ungeeignet sind, wird dabei häufig das MP3-Format verwendet. Deswegen wäre die Unterstützung gerade dieses Formates sinnvoll. Eine Ausweitung des Web-Services auf andere Medien wie Bilder, Texte und Videos wäre auch denkbar.

Die Web-Services sind eine interessante Technologie, die in Zukunft an Bedeutung gewinnen könnte. Für das URM-System sind sie insofern interessant, dass sie einen plattform- und

programmiersprachenunabhängigen Ansatz bieten. Da ein Web-Service nicht alle gängigen Wasserzeichenalgorithmen unterstützen kann, ist es notwendig, dass auf andere Web-Service-Implementierungen zugegriffen werden kann. Außerdem müssen für verschlüsselte Wasserzeichen die entsprechenden Schlüssel angefordert werden können. Dies alles ist mit der Web-Service-Technologie möglich.

Anhang A

Inhalt der CD

Auf der beiliegenden CD befinden sich die folgenden Ordner:

- *code*: In diesem Verzeichnis befindet sich der komplette Quellcode.
- *jar*: Dieses Verzeichnis enthält die ausführbaren jar-Dateien.
- *sample*: Das Verzeichnis enthält Beispiel-WAV-Dateien.
- *thesis*: Dieser Ordner enthält die Lyx- und PDF-Dokumente dieser Bachelorarbeit.

Abbildungsverzeichnis

1.1	Systementwurf	3
2.1	URM - Mediathek. Quelle: [JAH10]	6
3.1	Abtasten (Angelehnt an [Ahl02])	14
3.2	Quantisieren (Angelehnt an [Ahl02])	15
3.3	Aufbau einer kanonischen WAV-Datei (Angelehnt an [Wil03, HR00])	17
3.4	8-Bit Sample (Angelehnt an [Wil03])	18
3.5	16-Bit Sample (Angelehnt an [Wil03])	18
3.6	Struktur einer SOA (Quelle: [DJMZ05])	21
3.7	Web-Services-Architektur (Quelle: [DJMZ05])	23
3.8	Aufbau einer SOAP-Nachricht (Quelle: [DJMZ05])	24
4.1	Programmarchitektur	29
4.2	Beispiel	32
4.3	Watermark Embedding Tool	33
4.4	File-Browser	34
5.1	<i>Web-Service-Architektur</i>	39
5.2	<i>Web-Service-Klassendiagramm</i>	41
5.3	Beispiel	43
5.4	WSDL-Dokument	44
5.5	<i>Client</i>	45

Listings

4.1	Einbetten eines Wasserzeichens	31
4.2	Array <code>number</code>	32
5.1	Suchen nach der Zeichenkette <code>data</code>	41
5.2	Wasserzeichen auslesen	42

Literaturverzeichnis

- [Ach03] ACHZIGER, R.: *Digitale Wasserzeichen*, Westfälische Wilhelms-Universität Münster, Seminararbeit, 2003
- [Ahl02] AHLEMANN, O.: *Methoden der digitalen Audiobearbeitung*. <http://www.fh-wedel.de/~si/seminare/ss02/Ausarbeitung/9.digitalaudio/audio1.htm>.
Version: 2002
- [Dit00] DITTMANN, J.: *Digitale Wasserzeichen : Grundlagen, Verfahren, Anwendungsgebiete*. 1. Auflage. Springer, 2000. – 88–102 S. – ISBN 3–540–66661–3
- [DJMZ05] DOSTAL, W. ; JECKLE, M. ; MELZER, I. ; ZENGLER, B.: *Service-orientierte Architekturen mit Web Services: Konzepte-Standards-Praxis*. Spektrum, 2005
- [DMB10] DEUTSCHES-MUSEUM-BONN: *Die MP3-Audiokompression*, Deutsches-Museum-Bonn, Paper, 2010
- [Don10] DONNER, M.: *Puls Code Modulation (PCM) und Differential Puls Code Modulation (DPCM)*. <http://www.wilabs.ch/?page=dlmanager&id=39&mode=dl&select=kurzberichte>. Version: 2010
- [DS05] DITTMANN, J. ; STEINMETZ, R.: *Digitale Wasserzeichen*. http://www.giev.de/no_cache/service/informatiklexikon/informatiklexikon-detailansicht/meldung/digitale-wasserzeichen-34.html. Version: Juli 2005
- [ELK10] ELKO: *PCM - Pulscodemodulation*. <http://www.elektronik-kompodium.de/sites/kom/0312281.htm>. Version: 2010
- [FIP95] FIPS: *Secure Hash Standard (SHS)*, U.S. DoC/NIST, Publication, April 17 1995
- [Hen06] HENRICH, V.: *Medienverarbeitung mit Java in elementaren Einheiten*, Berufsakademie Mannheim, Studienarbeit, Juni 2006
- [HPG09] HUNDACKER, H. ; PÄHLER, D. ; GRIMM, R.: *URM - Usage Rights Management*, Poznan University of Economics Publishing House, in Nützel, J. and Alapan, A. (ed.), 'Virtual goods 2009, Nancy, France', Poznan University of Economics Publishing House, Poznan, Poland, Ausarbeitung, 2009
- [HR00] HÖSS, T. ; RIECK, T.: *WAV-Audio-Format*. <http://www.it.fht-esslingen.de/~schmidt/vorlesungen/mm/seminar/ss00/HTML/node107.html>.
Version: 2000
- [HUN] HUNDACKER, H.: *FORENSIC DIGITAL RIGHTS MANAGEMENT* , University of Koblenz-Landau, Ausarbeitung

- [IFP09] IFPI: *Digital Music Report 2009*. http://www.ifpi.org/content/section_resources/dmr2009.html. Version: Januar 2009
- [ITW10] ITWISSEN: *AAC-Kompression*. <http://www.itwissen.info/definition/lexikon/advanced-audio-coding-AAC-AAC-Kompression.html>. Version: 2010
- [Jä10] JÄNICKE, C.: *Grundlagen*. <http://www.bitworld.de/index.php/grundlagen/>. Version: Januar 2010
- [JAH10] JAHN, N.: *Winamp-Mediathek mit persönlicher Rechtekennzeichnung*, Universität Koblenz-Landau, Studienarbeit, 2010
- [Kap] KAPPES, A.: *Die Audiokodierung mp3*. <http://www.mp3encoding.de/>
- [Kli05] KLIMEK, H.: *Eine Web Service Schnittstelle für ein Web Service Entwickler-Portal*, Technische Universität Hamburg, Bachelorarbeit, Oktober 2005
- [Pus03] PUSCHER, F.: *Neue Wasserzeichen*, Ausarbeitung, 2003
- [REN02] RENATO, I.: *Open Digital Rights Language (ODRL)*. August 2002
- [RIA09] RIAA: *Standard Watermark Payload Configuration*, Paper, Version 1.0, März 2009
- [RTM02] ROSENBLATT, B. ; TRIPPE, B. ; MOONEY, S.: *Digital Rights Management – Business and Technology*. New York, 2002
- [SIT08] SIT, Das Fraunhofer-Institut: *Das Wasserzeichen*. <http://watermarking.sit.fraunhofer.de/Wasserzeichen/index.jsp>. Version: 2008
- [SIT09] SIT, Das Fraunhofer-Institut: *Wie funktioniert gehörangepasste Audiocodierung?* <http://www.iis.fraunhofer.de/bf/amm/products/mp3/mp3workprinc.jsp>. Version: 2009
- [SUN10] SUN/ORACLE: *The Java Tutorials*. <http://java.sun.com/docs/books/tutorial/sound/index.html>. Version: Januar 2010
- [Wil03] WILSON, S.: *WAVE PCM soundfile format*. <https://ccrma.stanford.edu/courses/422/projects/WaveFormat/>. Version: 2003