Universität Augsburg
Fakultät für Angewandte Informatik

Entwicklung eines Frameworks für ein Audio-Indizierungs-Plugin in der Google-Desktop-Suche

Bachelorarbeit im Studiengang
Informatik und Multimedia

vorgelegt von

Lars – Christian Larsen

Februar 2007

Unter Betreuung von:
Prof. Dr. Rainer Lienhart
Lehrstuhl für Multimedia Computing
Universität Augsburg

Erstgutachter:
Prof. Dr. Rainer Lienhart
Institut für Informatik
Lehrstuhl für Multimedia Computing

Zweitgutachter:
Prof. Dr. Elisabeth André
Institut für Informatik
Lehrstuhl für Multimedia-Konzepte und Anwendungen

Danksagung Mein besonderer Dank gilt Prof. Dr. Lienhart für die Ermöglichung dieser Bachelorarbeit. Des Weiteren möchte ich Gregor van den Boogaart für die stets sehr kompetente Hilfe danken.
Außerdem möchte ich meiner Mutter danken, die mich stets bei meinem Studium unterstützt hat.

Übersicht

Das Ergebnis dieser Bachelorarbeit soll ihnen bei der täglichen Arbeit mit Audio-Dateien auf ihrem Computer helfen. Oft sucht man auf dem eigenen Computer ein ganz bestimmtes Lied und es ist recht schwer und dauert lange, dieses mit der Windows Suche zu finden. Meist reichen die Informationen, die man dann als Suchergebniss bekommt nicht aus, sondern man will auf den ersten Blick mehr über die gefundenen Dateien wissen. Dazu könnten z.B. das Album, das Veröffentlichungsjahr, die Samplerate oder die Bitrate der Datei gehören. Aber an manchen Tagen reichen diese Informationen auch nicht mehr aus, sondern man will mehrere Lieder des gleichen Genre finden, die zu der aktuellen Gemütslage passen. Dazu ist dieses Google Indizierungs Plugin gedacht, welches ihnen neben der einfachen Suche nach Dateien diese eben erwähnten Zusatzinformationen zur Verfügung stellt. Ein weiteres Feature ist dabei, dass das Genre nicht einfach nur aus den Tags ausgelesen wird, sondern on the fly von einem Klassifikator bestimmt wird.

Abstract

The result of this bachelor thesis should provide you with your daily work with audio files on your computer. Often you are searching for a certain audiofile on your computer, but the Windows search function is very slow. And often the search results are not very auxiliary, and you want to know more about the found files. For example the album, the publish date or the samplerate or the bitrate of the file. And on some days even these informations are not enough. It can be possible that you want to find song from one genre that fit you actual mind. For that is this google indexing plugin, which provides beside a normal search the just announced properties. Another feature ist that the genre is not just the genre from the tagdata from the audio file, but the genre which is provided with an classificator.

Inhaltsverzeichnis

Kapitel 1

1 Einleitung

Wo finde ich die Emailadresse meines Betreuers? Wo finde ich den Quellcode eines Praktikums in meinem Studium von vor 2 Jahren? Auf welcher Website stand der Beispielquellcode für ein Programm? Diese ganzen Informationen sind irgendwo auf meinem Computer gespeichert, aber wo? Diese Informationen per Hand zu suchen, indem man alle Ordner anschaut, in denen Musik gespeichert ist wäre die eine Möglichkeit. Die andere Möglichkeit ist, diese Informationen mit einer Suchfunktion zu suchen. Dafür hat Windows eine eigene Suchfunktion, mit der man die eigene Festplatte nach Audio-Dateien durchsuchen kann. Diese Suche ist aber sehr eingeschränkt und gerät sehr schnell an ihre Grenzen. Wenn diese Suche für jemanden persönlich immer noch nicht genau und vor allem schnell genug, ist kann man sich weiter umschauen und stößt als nächste Möglichkeit auf eine Desktop Suche.

1.1 Motivation

Zielsetzung meiner Bachelorarbeit war die Entwicklung eines Google Desktop Plugins. Dabei sollten die Ergebnisse eines vorherigen Praktikums, dass sich mit Music Genre Klassifikation beschäftigte, in eine praktische Anwendung integriert werden. Da kam die Idee, dieses in eine Desktop Suche zu integrieren und die Suchergebnisse mit weiteren Informationen über die Audiodateien zu versehen, welche die tägliche Arbeit mit Audiodateien vereinfacht.

1.2 Was ist eine Desktop Suche

Als Windows Anwender kennt man bestimmt die hauseigene, sehr langwierige Suchfunktion. Es gibt über 100000 Windows Anwendungen, welche mehr als 10000 Dateiendungen verwenden. Hat man eine sehr große Festplatte und sucht beispielsweise all mp3 Dateien auf dieser, kann das sehr lange dauern. Und meistens kommt die Windows Suchfunktion auch an ihre Grenzen, wenn man z.B. alle Textdateien nach einem bestimmten Wort durchsuchen will. Außerdem ist die Suche nicht besonders präzise und es gibt keine Operatoren, welche Suchfunktionen logisch verknüpfen lassen. Hier setzten so genannte Desktop Suchen an, welche dem schnellen Durchsuchen des Computers nach verschiedenen Inhalten dienen. Diese indizieren den gesamten Datenbestand auf dem Computer in einer vorherigen Idlephase des PCs. Das heißt sie durchsuchen ihn, lesen die Dateien ein, filtern diese und speichern die Ergebnisse dann in einer Datenbank ab. Diese Indizierung dauert zwar ziemlich lang und der Index, der dabei entsteht, kann sehr groß werden (~100MB), aber dies lohnt sich, da die Suche danach sehr schnell funktioniert. Die Programmoberfläche der Suchfunktion greift dann auf diese Datenbank zu. Somit ist es später in Echtzeit möglich, Ergebnisse für gewünschte Suchabfragen zu bekommen. Ein weiterer Vorteil von Desktop Suchsystemen ist die hohe Präzision, da sie die durchsuchten Inhalte nach Dateityp filtern und diese Informationen dann auch sinngemäß zuordnen. Der Index muss jetzt aber immer aktuell gehalten werden. Deshalb laufen alle Tools der Desktop Suche im Hintergrund und registrieren jegliche Änderungen im Dateisystem. Neue und geänderte Dateien werden sehr schnell in den Index mit aufgenommen. Ab und zu sollte man aber dennoch das System komplett neu indizieren, da manche Dateiänderungen für die Desktop Suche unbemerkt ablaufen und der Index dann nicht ganz aktuell ist.

Die Google Desktop Suche ist auch eine solche Desktop Suche, die die Suche nach Dateien, E-Mails oder anderen Dateien, auf dem eigenen Computer ermöglicht. Die Google Desktop Suche hat zudem noch einen großen Vorteil. Jeder hat schon einmal mit der Google Suchmaschine etwas im Internet gesucht. Nicht umsonnst gehört das Wort „googeln“ schon zu unserem Wortschatz. Die Google Desktop Suche hat genau dieselbe Oberfläche, nur durchsucht sie dabei die lokale Festplatte.

1.3 Google Indizierungsplugin

Die Google Desktop Suche ist ein lokal ausgeführter Service, der es erlaubt eigene Dateien zu indizieren und zu durchsuchen. Außerdem stellt Google ein Google Desktop Suche Software Development Kit (SDK) zur Verfügung, welches es ermöglicht, eigene Indizierungsplugins zu schreiben. So kann man z.B. Email Programme wie Lotus, Eudora oder Yahoo Mail durchsuchen. Oder weitere Formate wie Wordperfect, StarOffice und andere Dateiformate, die von der Google Desktop Suche standardmäßig nicht unterstützt werden, hinzufügen. So können nicht nur neue Dateiformate hinzugefügt werden, sondern die Suchergebnisse vorhandener Formate können detailreicher und genauer und den eigenen Bedürfnissen angepasst erweitert werden. So kann man z.B. ein Plugin entwickeln, welches nicht nur nach allen zip Dateien auf dem System sucht, sondern man könnte diese durch das Indizierungsplugin entpacken lassen und dann den Inhalt der zip Datei in den Suchergebnissen anzeigen. Somit würde man sich das manuelle extrahieren jeder einzelnen Datei auf der Suche nach etwas ersparen. Der Kreativität des Programmierers, beim Entwickeln von Plugins für neue Dateiendungen sind dabei keine Grenzen gesetzt. Außerdem kann man mit dieser API, die Google Desktop Suche in eigene Programme integrieren.[1]

Kapitel 2

2 Implementierung

Ich habe das Google Desktop Plugin in C++ entwickelt. Diese Programmiersprache eignet sich gut, da sie für die Arbeit mit großen Dateien etliche Geschwindigkeitsvorteile bietet und außerdem war es bei der Entwicklung der gesamten Bachelorarbeit immer wichtig Teile später einmal für andere Betriebssyteme wie Linux oder MacOS wieder zu verwenden.
Da die Google Desktop Suche Windows spezifisch ist und auch mit einem Windows eigenen Verfahren kommuniziert, habe ich mich zuerst in die Google Desktop API und in das Windows Component Object Model (COM) eingearbeitet.

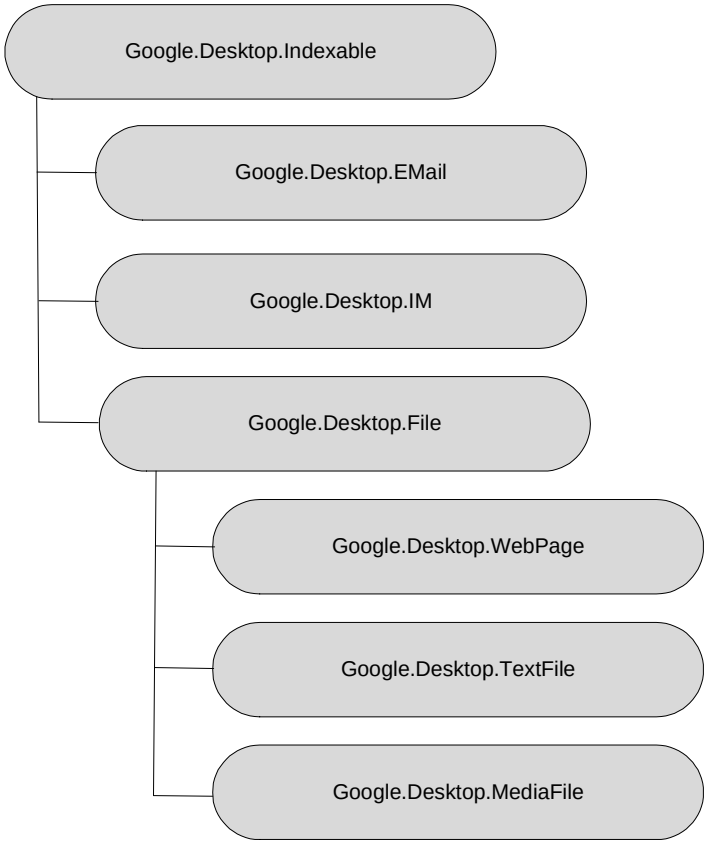
2.1 Google Desktop API

Google bietet eine API an, die in vier Teile aufgeteilt ist. Eine Index API, welche es ermöglicht Inhalte an Google Desktop zum indizieren und zum cachen zu schicken. Mit diesem Teil der API ist es möglich, Plugins für Dateiformate zu schreiben, die noch nicht unterstützt werden. Eine Query API, die es erlaubt Suchanfragen an die Google Desktop Suche aus eigenen Anwendungen heraus zu schicken. Eine Action API, mit der es möglich ist Standard Aktionen mit eigenen Aktionen zu überschrieben. Eine Event API, die es dem Plugin erlaubt Ereignisse zu empfangen, wenn neue Dateien indiziert werden.

2.1.2 Google Desktop Schema

Da es eine Menge verschiedener Dateiformate gibt, die verschiedene Eigenschaften haben, werden diese in der API grob in so genannte Schemata unterteilt. Die Google Desktop Suche bietet eine Reihe nicht erweiterbarer Schemata an, „Event Schemas“ genannt, die zum indizieren einer Datei verwendet werden können. Die Schemata sind hierarchisch geordnet. Es gibt Kinder Schemata die von ihren Eltern Schemata abgeleitet sind. Das

grundlegende Schema aller anderen ist das allgemeine Schema „Google.Desktop.Indexable“, welches grundlegende Daten für alle Dateien speichert. Zu den Daten gehören der Inhalt der Datei in einem Textformat und der MIME-Typ[1] des zu indizierenden Inhalts als „text/plain“ oder „text/html“ Format. In Verbindung mit dem Grund Schema können dann noch weitere, je nach Dateityp genauere verwendet werden. Diese sind von Google definiert und benutzen alle das Grundschema. Dazu gehören „Google.Dekstop.Email“, „Google.Dekstop.IM“, „Google.Dekstop.File“, „Google.Dekstop.WebPage“, „Google.Dekstop.TextFile“, „Google.Dekstop.MediaFile“. Die folgende Figur 1 zeigt die komplette hierarchische Struktur.



Figur 1: Google Desktop Search 1.0. event Schema hierarchy

2.2 Was ist das Windows Component Object Model (COM)

Das Microsoft Object Component Model (COM) ist ein Komponentenmodell, das von Microsoft entwickelt und 1993 ins Leben gerufen wurde. Die Idee ist, vorgefertigte und

[1] MIME-Typ besteht aus zwei Teilen: der Angabe eines Medientyps un der Angabe eines Subtyps. Beide Angaben werden durch einen Schrägstrich voneinander getrennt.

bereits getestete Bausteine (Komponenten) in den Entwicklungsprozess aufzunehmen, um daraus ein Anwendungssystem zusammenzusetzen. Softwarekomponenten sind wieder verwendbare Funktionseinheiten, die ihren inneren Aufbau abkapseln und über standardisierte Schnittstellen mit ihrer Umgebung kommunizieren.[4] COM ist eine Spezifikation, die Regeln für die Erstellung dieser Komponenten aufstellt, so dass diese Komponenten interoperabel[2] und dynamisch austauschbar sind und als „Black Box wieder verwendet werden können. [3: S 51] Dabei ist es egal, in welcher Programmiersprache eine COM-Komponente entwickelt wurde. Sie muss nur einheitlichen Standards unterliegen und ein spezielles binäres Format aufweisen. Binäres Format bedeutet, dass ein definierter Satz von Datentypen existiert, die in Bytefolgen umgesetzt werden können. Insbesondere gibt es einen festgelegten Mechanismus, wie ein bestimmter Block von Programmcode lokalisiert und aufgerufen werden kann.[4 – S.4]
Ziel solcher Komponentenmodelle ist es, die Softwareentwicklung enorm zu beschleunigen und die Qualität des Endproduktes zu steigern, da eben die Teile aus denen das neue Produkt zusammengebaut wird, schon getestet sind.

Eine COM Komponente ist definiert als eine Klasse (class), die ein oder mehrere Schnittstellen (*Interfaces*) implementiert. Dabei ist eine Klasse aber nicht im Sinne der klassischen Objektorientierung zu sehen. Sie ist vielmehr eine Sammlung von Methodenaufrufen, von denen die jeweils zusammengehörigen zu einem *Interface* zusammengefasst werden. Jede dieser Klassen hat dann eine eindeutige Kennung (*CLSID*), durch welche sie identifiziert werden kann. Auch die *Interfaces* haben eine solche Kennung, die *IID* genannt wird. [2]
Die Kommunikation mit einer COM-Komponente basiert auf den *Interfaces* dieser.
Unter Windows können diese Klassen aus z.B. Dlls (Dynamic Link Libraries) exportiert werden, die von allen 32-Bit Microsoft Betriebssystemen unterstützt werden.

Die Kommunikation mit einer Komponente basiert auf deren *Interface*. Dabei sind bei COM eine Menge Interfaces definiert, die man als Entwickler benutzten kann oder muss. Es ist aber genauso möglich, neue Komponenten selbst zu entwickeln oder existierende zu erweitern.

Zusätzlich verfügt COM über eine API in Form der so genannten COM-Library. Diese Bibliothek stellt einige Hilfsfunktionen für das Management von COM-Komponenten zur Verfügung.

[2] Interoperabilität: Kommunikations- und Interaktionsmechanismen zwischen beliebigen Applikationen sollen möglich sein.

Zusammenfassend kann man sagen, dass ein Entwickler durch den Einsatz von COM viele Möglichkeiten erhält. Er kann programmiersprachenunabhägig, versionsunabhängig, plattformunabhängig, objektorientiert Anwendungen entwickeln.

2.2.1 Architektur

COM basiert auf dem Client/Server Prinzip. Der Server ist dabei ein Objekt, welches wie schon vorher erwähnt, in einer COM-fähigen Programmiersprache geschrieben sein muss. Dieser COM-Server bietet mögliche aufrufbare Funktionen über ein *Interface* an. Der Client kann nun eine Instanz eines COM-Servers erzeugen und die vom COM-Server angebotenen Funktionen nutzen. Da die Programmiersprache mit der, der Server und der Client entwickelt wurden unterschiedlich sein kann, findet bei der Instanzierung und beim Aufrufen von Funktionen, die der Server zur Verfügung stellt, das so genannte Marshalling statt. Unter Marshalling versteht man das Entgegennehmen und Umwandeln einer Menge von strukturierten Datenelementen und/oder elementaren Werten in ein Format, welches es ermöglicht, diese in einer Nachricht an einen Empfänger zu schicken. In Fall von COM heißt das die austauschbaren Dateien in die Programmiersprachenunabhängige *IDL* (Interface Definition Language) umzuwandeln.[5]

2.2.2 Interfaces

Im Interface ist eine Menge von Operationen zusammengefasst. Dabei ist ein COM-Interface ein Zeiger auf eine VTable[3], die wiederum Zeiger auf Funktionen enthält. In COM-Interfaces wird eine Funktion durch ihren Namen, die Anzahl und Art der Übergabeparameter, dem Rückgabewert sowie einiger Zusatzinformationen beschrieben. Ein Interface hat außerdem eine systemweit einmalige Indentifikationsnummer, die GUID (Globally Unique Indentifier), welche es eindeutig indentifiziert. Somit macht es keine Probleme, wenn COM-Interfaces unterschiedlicher Hersteller den gleichen Namen haben.

[3] VTable: Eine Tabelle die wiederum Zeiger auf Funktionen enthält, die vom COM-Objekt angeboten werden.

Ein Beispiel für ein COM-Interface, ist das IUnknown Interface welches in der Interface Definition Language (*IDL*) beschrieben ist. Es bildet die Basis für alle anderen Interfaces. Es muss von jeder COM-Klasse implementiert werden.[3 – S.52]

```
[
   object,
   uuid(00000000-0000-0000-C000-000000000046)
 ]
 interface IUnknown {

        HRESULT QueryInterface([in] GUID* rrid,
                               [out] void** ppvObj);

        ULONG   AddRef();
        ULONG   Release();
 }
```

Das Interface IUnknown definiert drei virtuelle Methoden, `AddRef`, `Release` und `QueryInterface`. Diese können über jede Schnittstelle aufgerufen werden. `QueryInterface` erwartet dabei zwei Parameter.[3] Eine GUID als Eingabeparameter, der durch [in] gekennzeichnet und einen Ausgabeparameter, der durch [out] gekennzeichnet ist. Dieser liefert bei Erfolg einen Zeiger auf die angeforderte Schnittstelle. Die Methode `AddRef` ist für die Erhöhung des internen Referenzzählers zuständig, z.B. bei einem erfolgreichen Aufruf von `QueryInterface`. `Release` vermindert dagegen den Referenzzähler um eins. Die Referenzzählung dient der Speicherverwaltung. [4]

2.2.3 GUID

Die Interface Indentifier (IID) und Class Identifier (CLSID) sind Beispiele für Globally Unique Identifier (GUID). Wie schon erwähnt, ist es wichtig, das COM-Komponenten eine eindeutige Kennzeichnung in Form einer dieser Indentifier haben.

Beispiel – GUID: {DECD4FAF-C3C2-43a6-BF2B-670BF7BC34E2}

Da es nicht sehr sinnvoll ist, diese GUID in einer weltweit agierenden, zentralen Organisation zu vergeben, kann man die GUID mit einem Algorithmus lokal auf dem eigenen Rechner berechnen. Dieser Algorithmus, der vom OSF[4] spezifiziert wurde, erzeugt GUIDs mit einer Länge von 128 Bit. Somit ist die Gesamtzahl der eindeutigen Schlüssel, mit einem Wert von 2^{128} so groß, dass die Wahrscheinlichkeit einen Schlüssel doppelt zu generieren gegen null geht. Dabei kennzeichnen die ersten 48 Bit des Schlüssels den Rechner, auf dem die GUID erzeugt wurde, die nächsten 60 Bit enthalten einen Zeitstempel, der die 100-Nanosekunden-Intervalle seit dem 15.10.1582 zählt. Die nächsten 4 Bit enthalten die GUID-Versionsnummer und die letzten 16 Bit werden anhand der Systemzeit berechnet.[3 - S.61]

In Visual Studio lassen sich diese GUIDs ganz leicht mit dem mitgelieferten Tool „Create GUID“, welches unter den Tools zu finden ist, generieren.

2.2.4 IDL

Die GUID Werte ermöglichen es, eine Schnittstelle eindeutig zu identifizieren. Es ist nötig zu wissen, welche Operationen diese Schnittstelle zur Verfügung stellt.

Es gibt diesbezüglich keine Vorgaben bei COM. Theoretisch kann jeder Entwickler die Schnittstellen in jeglicher beliebiger Form dokumentieren, als Text-Datei oder als HTML-Dokumentation. Es gibt aber auch eine so genannte Interface Definition Language (*IDL*) dafür. Im speziellen Fall von Microsoft wird diese auch MIDL (Microsoft IDL) genannt.

Bei der Google Desktop SDK ist eine „GoogleDesktopSearch.idl“ Datei mitgeliefert. Diese kann man in der Visual Studio Kommandozeile durch den Befehl MIDL „GoogleDesktopSearch.idl“ in eine Header-Datei (.h) umwandeln. Diese Header-Datei stellt dann alle Funktionen und GUIDs zur Verfügung, die benötigt werden, um mit der Implementierung des Plugins zu beginnen.[3 – S.61]

[4] Die Open Software Foundation ist ein Kosortium, das von HP, IBM und DEC gegründet wurde, um neue Industriestandards für UNIX zu enteickeln

2.2.5 Registry

Hat man eine COM-Komponente im System registriert, kann man Informationen darüber in der Registry des Computers abfragen. Diese Informationen ermöglichen dem Betriebsystem und dem Anwender spezifische Daten zu erfahren, die bei der Suche nach einer COM-Komponente nötig sind. Die Klassen, die durch ihre CLSID eindeutig identifiziert sind, werden in der Registry unter dem Schlüssel HKEY_CLASSES_ROOT\CLSID eingetragen.[4]

2.2.6 ATL – Die Active Template Library

Durch COM wurden nun die Anforderungen an COM-Komponenten aufgestellt. Will man als Entwickler mit dem Programmieren solcher Komponenten beginnen, wünscht man sich jedoch Unterstützung dabei. Um Unterstützung zu bekommen gibt es so genannte Frameworks, die dabei helfen sollen. Unter Windows sind das zum Einen die Microsoft Foundation Classes (MFC) und die ActiveX Template Library (ATL). Dabei ist die MFC aber eher für grafische Benutzeroberflächen konzipiert und hat deshalb eine Menge Ballast mit an Bord. Bei der ATL hingegen macht ausgiebigen Gebrauch von C++ Templates und Inline-Funktionen, so dass ein Includieren von den entsprechenden Header Dateien in den Quellcode ausreicht. Visual Studio bietet für die Entwicklung mit ATL Wizards an, die das Anlegen eines solchen Projekt enorm erleichtern.

2.3 Plugin DLL – COM (Übertragen auf Google Plugin)

Wie in Kapitel 2.2.1 schon erwähnt gibt es bei COM Objekten immer einen COM-Server und einen COM-Client. Im diesem Fall ist der Server die Google Desktop Suche. Diese stellt Interfaces zur Verfügung, die Methoden zum Anmelden des Plugins bei der Desktop Suche, Methoden zum Senden der ermittelten Daten einer neu indizierten Datei und Methoden zum Abmelden des Plugins, wenn es nicht mehr benötigt wird, enthalten. Will man jetzt das Plugin bei der Google Desktop Suche anmelden, muss man die CLSID dieser kennen, um eine neue Instanz von ihr zu erzeugen und sich bei ihr zu registrieren. Gleichzeitig braucht

man noch eine eigene CLSID, die man mit einem Tool wie in 2.2.3 erwähnt generieren kann, um den Einstiegspunkt im COM-Client, in diesem Fall das Plugin zu adressieren. War dieser Vorgang erfolgreich, wird die Google Desktop Suche von nun an für jede Dateiendung, für die für das Plugin registriert ist, in die Startmethode (`HandeFile`) springen und den Code darin durchlaufen. Wenn das PLugin wieder deregistriert werden soll, muss man ebenfalls wieder die CLSID der Google Desktop Suche und die eigene CLSID, mit der man das Plugin angemeldet hat wissen, um es wieder abzumelden.

Kapitel 3

3 Design

3.1 Wie leistet das Plugin etwas?

Was bekommt der Nutzer dieses Plugins und mit welchen Techniken und Hilfsbibliotheken werden diese Informationen zusammengetragen? Der Nutzer bekommt zu jeder Audio-Datei weitere Informationen, mit denen man auf den ersten Blick sagen kann um was für eine Art von Datei es sich hier handelt. Ist es z.B. ein Lied aus dem Album von einem bestimmten Musiker oder ist es vielleicht nur eine kleiner Jingle, der nur ein paar Sekunden lang ist.

3.2 AudioIO

Da es eine Menge verschiedener Audio Formate gibt, war es mir wichtig eine Bibliothek zu finden, die in der Lage ist viele dieser zu öffnen. Man unterscheidet dabei unkomprimierte von komprimierten Formaten. Zu den unkomprimierten Formaten gehört z.B. *WAV*, bei dem das Signal als PCM[5] Format vorliegt. Dies hat aber den Nachteil, dass ein Lied durchschnittlich etwa 30-40 MB groß ist. Ein Beispiel für ein komprimiertes Audio-Format ist *MP3.* Dies ist ein Dateiformat zu verlustbehafteten Audiodatenkompression. Dies wird erreicht indem man Bereiche aus dem Lied, die der Mensch nicht hören kann herausfiltert und somit die Menge der Daten um ein Vielfaches reduziert. Dabei wird versucht, keine für den Menschen hörbaren Verluste zu erzeugen.
Eine Alternative ist *OGG*, das im Gegensatz zu *MP3* patentfrei und quelloffen ist. Bei *OGG* handelt es sich um ein Container-Dateiformat für Multimedia-Dateien. Die Dateien können gleichzeitig Audio-, Video- sowie Textdateien enthalten. *OGG* wurde entwickelt, um Multimedia Inhalte effizient zu speichern und zu streamen. Die Entwicklung dieses Containers wird von der Xiph.org Foundation vorangetrieben. Diese ist auch für den bekanntesten Audio-Codec[6] Vorbis bekannt, welcher dazu benutzt wird Audio-Dateien zu

[5] PCM: Pulse-Code-Modulation, eine Modulationsform, bei der ein analoges Signal binär codiert wird.
[6] Codec: Ein Verfahren bezeichnet man ein Verfahren, dass Daten digital codiert oder decodiert

codieren und zu decodieren. So erreicht man mit diesen verlustbehafteten Methoden etwa Dateigrößen um die 3–4 MB.
Ein weiteres komprimiertes, Audio-Dateformat ist *FLAC*. *FLAC* steht für „Free Lossless Audio Codec“, ist also ein verlustfreier Audio Codec. Im Gegensatz zu *MP3* und *OGG*, sind mit FLAC komprimierte Audiodateien absolut originalgetreu. Decodiert man ein FLAC-Datei und kodiert sie dann wieder hat man absolut keine Qualitätsverluste. Man schafft damit im Idealfall etwa eine Kompressionsrate von 2:1. Üblicherweise beträgt die Dateigröße aber etwa dreiviertel der Originaldatei.

Ein weiteres Format sind MIDI Dateien, bei denen die Datei nicht aus Musikinformationen besteht, sondern aus Schaltbefehlen zur Ansteuerung von Instrumenten oder einer Soundkarte.

Letztendlich hatte ich dabei zwei Bibliotheken zu Auswahl, die ich im Internet gefunden habe. Beide können eine Menge dieser unkomprimierten und komprimierten Formate öffnen. Dabei handelt es sich zum einen um die fmod[12] Bibliothek und zum anderen um die Audiere[11] Bibliothek. Beide können recht viele Dateiformate öffnen, aber die Audiere Bibliothek ist Open Source und unter der LGPL lizenziert, das heißt, dass man sie frei im eigenen Code verwenden darf. Die fmod Bibliothek hingegen ist ab einem bestimmten Punkt kostenpflichtig und war deshalb nicht so interessant.

Audiere ist eine high-level Bibliothek, die die Formate *OGG* Vorbis, *MP3*, *FLAC*, *WAV*, *AIFF*, *MOD*, *S3M*, *XM* und *IT* Dateien unterstützt. Für das Öffnen der *OGG* Dateien benutzt Audiere die OggVorbis[11] Bibliothek. Für das *FLAC* Format, die libFlac[15] Bibliothek. Für das Speex Format verwendet Audiere die libspeex[16]. Speex ist ein verlustbehafteter Audiocodec, der speziell auf die Platz sparende Speicherung von Audiodateien ausgelegt ist, die menschliche Sprache enthalten.
Dabei kann man die Dateien mit Audiere nicht nur öffnen um mit diesen zu arbeiten, sondern es ist auch möglich, die Dateien direkt abzuspielen. Unter Windows geht dies mit DirectSound oder WinMM und unter LINUX oder Cygwin mit OSS. Außerdem ist Audiere[13] auf verschiedenen Plattformen verwendbar. Unter Windows, LINUX, Cygwin und IRIX. Dies war für mich immer bei der Auswahl von Bibliotheken wichtig, da es so später möglich ist, Teile dieser Bachelorarbeit weiter zu verwerten.

3.3 Tags

Der Benutzer des Plugins bekommt zusätzlich Informationen zu Metadaten die in Audiodateien wie *OGG* und *MP3* gespeichert sind. Diese sind in Tags gespeichert, was so viel heißt wie Etikett, Beschriftung oder Anhänger. In diesen Tags sind etliche Zusatzinformationen zu einer Datei gespeichert. Dazu gehören z.B. Titel, Interpret, Album aber auch Audio Eigenschaften wie Bitrate, Samplerate oder die Anzahl der Kanäle der Datei. Um diese Informationen auszulesen verwende ich die Bibliothek taglib[10] von Scott Wheeler. Die Bibliothek unterstützt die das Auslesen und Schreiben der Metainformationen einer Menge bekannter Audioformate. Zur Zeit unterstützt taglib ID3[7] Tags in den Formaten ID3v1 und ID3v2. Außerdem können Vorbis Kommentare für *OGG* und Vorbis Kommentare für *FLAC* ausgelesen werden.
Dabei habe ich mich für diese Bibliothek entschieden, da sie mit einigen Vorteilen vom Autor angepriesen wird, die sich sehr viel versprechend anhören. So ist Taglib bis zu 6 mal schneller als id3lib[8] und 3 mal schneller als libvorbisfile[9], welche von der Xiph.org, den Entwicklern des *OGG* Formates stammt.

3.4 Performance und Thumbnails

Um die Geschwindigkeit des Indizierungsprozesses für Audio-Dateien nicht unermesslich lang zu gestalten, wurde zusätzlich eine so genannte Performance Bibliothek eingesetzt. Die IPP[15] (Intel Performance Primitives) Bibliothek, die Funktionen zur Verfügung stellt, die auf die Intel Prozessor Architektur zugeschnitten sind.

Wie sagt man so schön, ein Bild sagt mehr als tausend Worte. Deshalb war es wichtig, bei den Suchergebnissen auch eine visuelle Darstellung der Audio-Dateien zu haben. Der Benutzer bekommt einen Ausschnitt aus der Audio-Datei in Form eines Wellendiagramms angezeigt. Somit kann man die unterschiedlichen Musikstücke mit einem Blick recht schnell vergleichen. Um diese Bilder zu erzeugen wurde die Bibliothek OpenCV[16] verwendet, welche Funktionen zu Verfügung stellt, um die Bilder zu erzeugen und dann in einem komprimierten Format, wie PNG oder JPEG zu speichern.

[7] ID3: **Id**entify an MP**3**
[8] Id3lib: ist eine OpenSource Bibliothek zum Lesen und Schreiben von id3 Tags

3.5 Audio Hilfsbibliotheken

Zur Bestimmung des Genres wurde die Bibliothek libmgc[19] verwendet, welche parallel zu dieser Bachelorarbeit unter Linux entwickelt wurde. Ich habe dieses Projekt in ein Visual Studio Projekt portiert und daraus eine Bibliothek für Windows kompiliert. Die Bibliothek ist in der Lage ein Musikstück, welches momentan indiziert werden soll nach dem Genre zu klassifizieren. Um dies zu ermöglichen, wurde der Klassifikator den diese Bibliothek verwendet, vorher trainiert und die Trainingsdaten sind dem Plugin mitgeliefert. Somit ist es möglich, eine beliebige neue Datei nach ihrem Genre zu klassifizieren.
Da die Bibliothek in ihrer Grundform allerdings nur *WAV* Dateien lesen kann und ich den Quellcode vorliegen hatte, habe ich die Bibliothek zusätzlich noch so weit umgebaut, dass sie auch in der Lage ist, alle Dateiformate die von Audiere unterstützt werden zu verarbeiten. Dies fand ich wichtig, da doch die meisten Anwender eines solchen Google Plugins, aus Platzgründen wohl einen wesentlich höheren Anteil an komprimierten Formaten wie *OGG* oder *MP3* als an nicht komprimierten Formatem wie *WAV* auf ihrer Festplatte liegen haben.

Kapitel 4

4 Klassifikation (Music Genre Classification)

Was ist Klassifikation und wie setzt man sie ein? Bei der Klassifikation werden Objekte anhand bestimmter Merkmale in verschiedene Klassen eingeteilt.
Was bedeutet Klassifikation im Bereich der Musik? Dabei werden aus den Audio-Dateien *Features* extrahiert, anhand deren das Lied einer Gruppe zugeordnet werden kann. So könnte man z.B. Lieder nach weiblichen oder männlichen Gesang unterscheiden oder anhand ihres Rhythmus in z.B. Walzer, Rumba oder Tango unterteilen. Bei der für dieses Plugin verwendeten Musik Genre Klassifikation wird das Stück nach dem Musikbereich, dem Genre unterschieden. Hierzu gehören z.B. Classic, Country, HipHop, Jazz. Ein Genre enthält dabei Stücke ähnlichen Stils. Diese Genre Gruppen lassen sich dann in weitere Gruppen, so genannte Subgenre unterteilen. So hat z.B. die Rockmusik das Subgenre Heavy Metal dieses hat das Subgenre Black Metal und dieses das Subgenre Vinking Metal. Somit kann man theoretisch bis zu 16 Genres mit relativ guten Ergebnissen unterteilen. Man muss allerdings beachten, dass eine zu genaue Aufgliederung in die einzelnen Genres auch eine unschärfere Abgrenzung mit sich bringt.

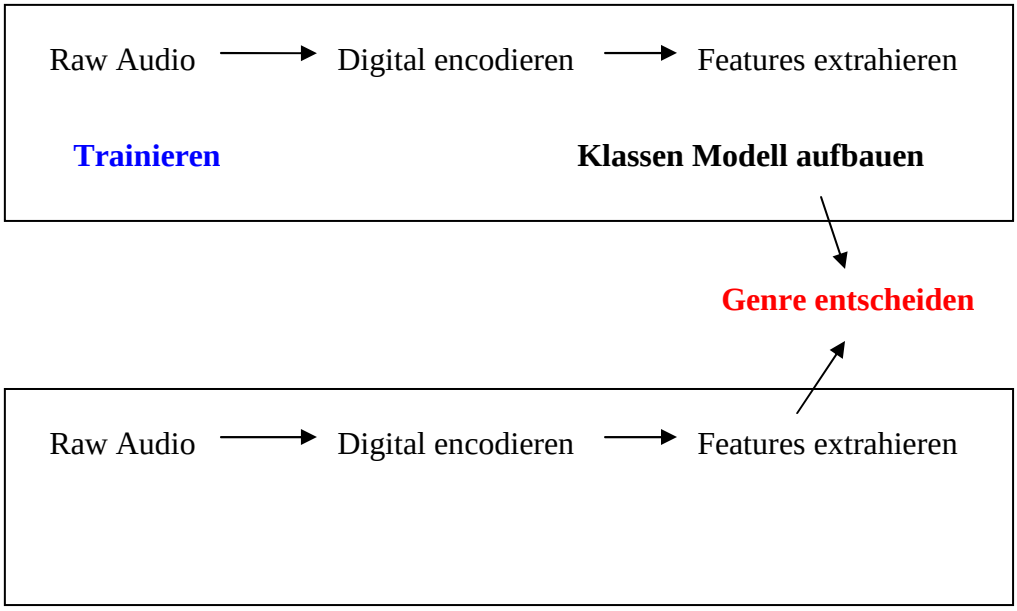
Da Musikdatenbanken, bei z.B. dem iTunes Store oder anderen vergleichbaren Online Musikläden immer größer werden und es mühsam ist, diese manuell zu katalogisieren, macht eine praktische Anwendung einer solchen Klassifikation großen Sinn. Genauso sinnvoll ist diese Einteilung aber auch auf dem heimischen PC, wenn man einfach nur auf der Suche nach einem Musiktitel ist, der zu der aktuellen Gemütslage des Benutzers passt.

4.1 Features

Allgemein ist ein Feature ein Merkmal, das eine bestimmte Eigenschaft von etwas beschreibt. Bei der in diese Bachelorarbeit verwendeten Klassifikation werden die folgenden Features verwendet. So genannte *Rhythmus Features*. Diese beschreiben eine wiederkehrende Folge von Schlägen und Pausen in einem Musikstücks. Dabei wird in der Musik die Anzahl der Schläge als BPM (beats per minute) bezeichnet. So hat z.B. HipHop viel markantere Beats als Kammermusik. Des Weiteren werden *Pitch Features* verwendet. Der Pitch sagt etwas über die Tonhöhe eines Musikstückes aus. So haben verschiedene Musikstücke aus verschiedenen Genres ganz charakteristische Tonhöhen. Ein weiteres Feature sind die *Timbral Texture* Features. Durch Timbre Features ist man in der Lage, Mischungen aus Stücken mit ähnlichen Rhythmus und ähnlichem Pitch zu unterscheiden. Ein weiteres Feature ist das *Low Energy* Feature. Dieses Features misst die RMS-Energy eines Stückes. Bei einem immer gleich bleibenden Stück sollte der Wert etwa 50 Prozent erreichen. Hat ein Stück allerdings mehr Dynamik sollte dieser Wert höher werden. Und das letzte Feature sind die *Mel-Frequency Cepstral Coefficients (MFCC)*, welche standardmäßig in der Spracherkennung verwendet werden. Die Koeffizienten sind ein weiteres gutes Merkmal um das Spektrum der Schwingungen einer Audiodatei zu repräsentieren.

4.2 Klassifikator

Ein Klassifikator funktioniert immer folgendermaßen. Zuerst muss man ihn trainieren. Das heißt, er wird auf schon bekannte und klassifizierte Dateien angewandt und speichert sich dann die Merkmale dieser in bestimmten Strukturen. Danach kann der Algorithmus (Klassifikator) anhand der erlernten Strukturen einen neuen und bisher unbekannten Fall aufgrund der beobachteten Attribute und deren Ausprägungen in eine der bekannten Klassen einordnen. In der Abbildung 2 zeigt das Schema der grundsätzlichen Funktion eines Klassifikators.



Einige sehr bekannte Klassifikatoren sind der Bayes, der KNN (k-nearest neighbour), der k-means und der support vector maschines Algorithmus. Diese Algorithmen haben alle unterschiedliche Arten, beim Trainieren die unterschiedlichen Klassen aufzubauen und eignen sich dementsprechend für verschiedene Klassifikationsgebiete.

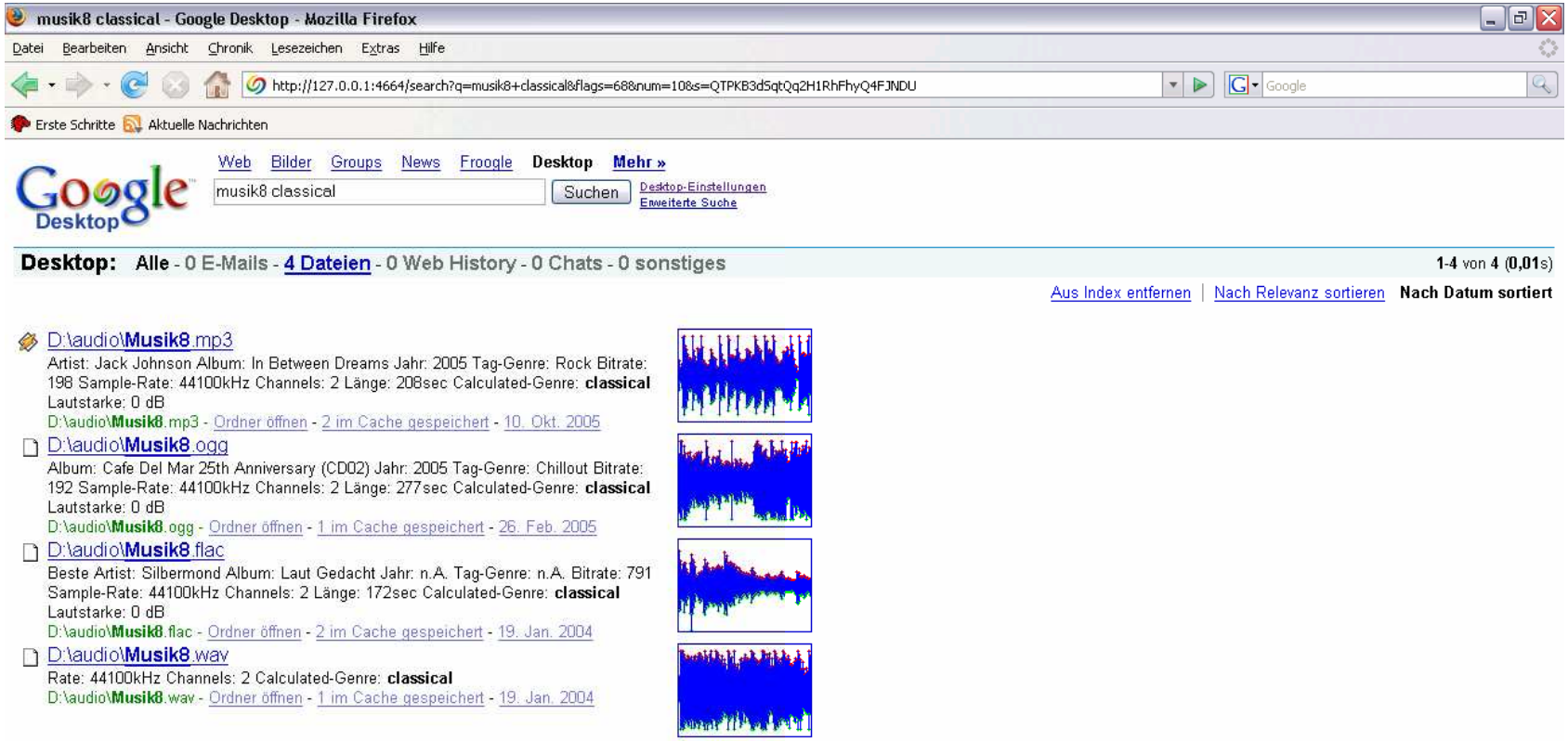
Weitere Informationen zum Thema Klassifikation und Feature Extraktion enthält die Bachelorarbeit von Florian Lingenfelser, der die libmgc zur Music Genre Klassifkation entwickelt hat.[19]

Kapitel 5

5 Evaluierung

5.1 Benutzung

Ist das Plugin auf dem Computer installiert (ein Installationsanleitung finden sie im Anhang), indiziert die Google Desktop Suche von nun an folgende zusätzliche Dateiformate: **WAV, AIFF, MP3, OGG, FLAC und SPX**.
Als Inhalt der Suchergebnisse werden diverse Daten angezeigt. Zum einen werden die Tags der Audiodateien ausgelesen, wie z.B. Artist, Album, Jahr. Außerdem werden eine Reihe von Audioeigenschaften angezeigt. Die Bitrate, die Samplerate und die Kanäle. Ein weiteres Feature, welches angezeigt wird, ist das Musicgenre des entsprechenden Musikstückes. Dabei handelt es sich aber nicht um das Genre, welches in den Tags der Audiostücke gespeichert ist. Das Genre wird durch einen vorher trainierten Klassifikator anhand der Eigenschaften des Musikstückes bestimmt. Somit kann man je nach momentaner Stimmung, die Musiksammlung schnell nach allen Lieder die das Genre, welches zur aktuellen Genesung passen, durchsuchen
Um einen Eindruck zu bekommen wie so ein Suchergebniss aussehen kann, sehen Sie in der folgenden Abbildung eine Beispielsuche. Ich habe hier nach allen Dateien, die den Namen Musik8 und das Genre classical haben, gesucht. Wie man sieht werden zu den Formaten *MP3*, *OGG* und *FLAC* die Taginformationen, Informationen zu den Audio Eigenschaften und ein Thumbnail angezeigt. Bei der *WAV* Datei sieht man Informationen zu den Audioeigenschaften und ein Thumbnail. Das Thumbnail zeigt bei einer Audiodatei mit einer Samplingrate von 44100KHz einen Auschnitt von etwa 2,5 - 3 Sekunden. Außerdem haben alle Suchergebnisse, die Zusatzinformation Calculated-Genre, welche das mit dem Klassifikator errechnete Genre anzeigt. Manchmal kann es vorkommen, dass sie nicht alle Informationen lesen können, da diese nach der Hälfte aufhören. Sollte dies der Fall sein, einfach auf das Thumbnail klicken und der Rest wird sichtbar. Mehr Informationen zur Installation und zum Ausprobieren des Plugins finden sie im Anhang A3 Installation des GDP.


musik8 classical - Google Desktop - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://127.0.0.1:4664/search?q=musik8+classical&flags=68&num=10&s=QTPKB3dSqtQq2H1RhFhyQ4FJNDU
Erste Schritte Aktuelle Nachrichten
Web Bilder Groups News Froogle Desktop Mehr »
Google Desktop
musik8 classical
Suchen
Desktop-Einstellungen
Erweiterte Suche
Desktop: Alle - 0 E-Mails - 4 Dateien - 0 Web History - 0 Chats - 0 sonstiges
1-4 von 4 (0,01s)
Aus Index entfernen | Nach Relevanz sortieren Nach Datum sortiert
D:\audio\Musik8.mp3
Artist: Jack Johnson Album: In Between Dreams Jahr: 2005 Tag-Genre: Rock Bitrate: 198 Sample-Rate: 44100kHz Channels: 2 Länge: 208sec Calculated-Genre: classical Lautstarke: 0 dB
D:\audio\Musik8.mp3 - Ordner öffnen - 2 im Cache gespeichert - 10. Okt. 2005
D:\audio\Musik8.ogg
Album: Cafe Del Mar 25th Anniversary (CD02) Jahr: 2005 Tag-Genre: Chillout Bitrate: 192 Sample-Rate: 44100kHz Channels: 2 Länge: 277sec Calculated-Genre: classical Lautstarke: 0 dB
D:\audio\Musik8.ogg - Ordner öffnen - 1 im Cache gespeichert - 26. Feb. 2005
D:\audio\Musik8.flac
Beste Artist: Silbermond Album: Laut Gedacht Jahr: n.A. Tag-Genre: n.A. Bitrate: 791 Sample-Rate: 44100kHz Channels: 2 Länge: 172sec Calculated-Genre: classical Lautstarke: 0 dB
D:\audio\Musik8.flac - Ordner öffnen - 2 im Cache gespeichert - 19. Jan. 2004
D:\audio\Musik8.wav
Rate: 44100kHz Channels: 2 Calculated-Genre: classical
D:\audio\Musik8.wav - Ordner öffnen - 1 im Cache gespeichert - 19. Jan. 2004
musik8 classical
Suchen
Google Desktop – Startseite - Verlauf durchsuchen - Indexstatus - Datenschutz - Info - © 2006 Google

5.2 Ausblicke

In der Zukunft möchte ich das Plugin noch um folgende Funktionen erweitern.

Wie in Kapitel 3.2 schon erwähnt, gibt es neben Audiodateien, die in unkomprimierten oder in komprimierten Formaten vorliegen, noch *MIDI* Dateien, welche nur Steuersignale enthalten. Bei der Suche nach Bibliotheken zum Öffnen von Audiodateien, habe ich zusätzlich noch die Bibliothek TiMidity++ [18] gefunden, die in der Lage ist, diese *MIDI* Dateien im Speicher in *WAV* Dateien umzuwandeln. Mir war es bis zum jetzigen Stand noch nicht möglich einen Windows Port dieser Bibliothek zu kompilieren um diese auch zu verwenden. Ich möchte diese aber im nächsten Schritt noch in das Plugin einbauen, so dass für die MIDI Dateien auch das Genre bestimmt und Thumbnails erzeugt werden können.

Da die Auschnitte der Thumbnails momentan einfach nur die Mitte einer Audiodatei darstellen, sind sie nicht so aussagekräftig für die Datei. Es könnte z.B. passieren, dass an dieser Stelle zufällig eine Stille oder eine eher leisere Passage vorliegt. Somit ist das Thumbnail dann natürlich nicht aussagekräftig für dieses Lied.
Deshalb möchte ich das Plugin noch um die Möglichkeit erweitern den Refrain in einem Lied zu erkennen. Dies wurde im vorigen Jahr schon in einem Praktikum an diesem Lehrstuhl entwickelt. Somit hätte man immer ein aussagekräftiges Stück eines Liedes.

Da ich die libmgc verwende, muss momentan jede Audiodatei komplett in den Speicher geladen werden. Die libmgc war ursprünglich so implementiert, dass man ihr eine wav Datei mitgeben musste und diese wurde dann komplett in den Speicher geladen und dann bearbeitet. Ich habe die Bibliothek schon soweit erweitert , dass man ihr keine Audiodatei sondern einen Buffer übergibt, in dem die Daten einer schon geöffneten Datei gespeichert sind. So ist es immerhin schon möglich, weitere Formate zu klassifizieren, die von der Bibliothek Audiere unterstützt werden. Im nächsten Schritt möchte ich die libmgc noch soweit ändern, dass diese in der Lage ist, die Audio-Dateien in Blöcken zu öffnen und zu bearbeiten, da die momentane Speicherauslastung zu groß ist.
Ein weiteres Problem das Aufgetreten ist, ist das manche Features, die die libmgc berechnet jetzt nicht mehr stimmen, da die Bibliotheken, die die Audio-Dateien öffnen die mit verschiedenen Werten machen. Die libsndfile, die ursprünglich in der libmgc verwendet wurde öffnet die Werte der Audio-Datei mit kleinen Float-Werten. Die Audiere Bibliothek hingegen öffnet die Datei mit sehr großen Integer-Werten. Dies führt bei der Berechnung mancher Features zu Problemen. Daher sind momentan die Klassifizierungsergebnisse sehr

dürftig, da durch die fehlerhaften Werte die Ergebnisse sehr verfälscht werden. Im nächsten Schritt müssen diese Features noch einmal überarbeitet und normiert werden, dass sie auch für die Audiere Bibliothek aussagekräftige Werte liefern.[20]

5.3 Schluss

Ich habe dieses Google Plugin entwickelt um die tägliche Arbeit mit Audio-Dateien zu erleichtern.
Um dies zu erreichen habe ich mich in viele Bibliotheken und Techniken, des C++ Programmierens eingearbeitet und mit diesen auseinander gesetzt. Angefangen beim Einlesen in immer neues APIs, dem Einbinden von neuen Bibliotheken und das Ausprobieren dieser über das kompilieren dieser für Windows. Genauso habe ich sehr viel mit Visual Studio gearbeitet und dieses kennen gelernt. Oft hat sich dies als gar nicht so leicht herausgestellt, da viele Probleme aufgetreten sind, die ich lösen musste.
Für mein weiteres Studium allerdings war diese Bachelorarbeit sehr lehrreich, da ich durch einen „Sprung ins kalte Wasser“ sehr viel Nützliches angeeignet habe, was mir zukünftig sehr hilfreich sein wird.

Anhang A

Anhang

Der Anhang gibt zuerst eine kurze Erläuterung wie ein Visual Studio Projekt für eine Google Desktop Suche angelegt werden muss, danach wird die Installation des Plugins beschrieben und es gibt eine Beschreibung des CD Inhaltes.

A.1 Visual Studio Projekt für GDP

Folgende Anleitung habe ich aus der Google Desktop Plugin SDK. Ich habe diese allerdings an manchen Stellen, die ich etwas unklar fand mit roter Schrift noch etwas erweitert.

Writing and Distributing Your GD Plug-in

Developing a GD Plug-in With Visual Studio

You can use Visual Studio 2003 or Visual Studio 2005 to create a class template for your plug-in, such that you only have to fill in registration with GD and, if necessary, how your HandleFile method is implemented.

The steps needed are:

1. Launch Visual Studio 2003. Select **File >New Project**. Select **ATL project**.
    1. Give the project an appropriate name
    2. Click **OK**.
2. Select the **Application Settings** tab.
    1. De-select **Attributed**, and leave all other settings alone.
    2. Click **Finish**.
3. Your new project will open in a new solution. Right-click your project.
    1. From the **Add** submenu, select **Add Class**.
    2. From the resulting dialog, select **ATL Simple Object**.
    3. Click **Open**.
    4. Name your new class, for example **MyCrawlPlugin**.
    5. Select the **Options** tab.
    6. Select threading model **Both**.
    7. Check the checkbox next to **ISupportErrorInfo** to enable it.
    8. Click **Finish**.
4. Your source file will open in Visual Studio.
5. Go to **Class View**.
    1. Navigate to your main interface, for example **IMyCrawlPlugin** for your **MyCrawlPlugin** class.
    2. Right click, and from the **Add** submenu select **Add Method**.
    3. Add your **HandleFile** method, specifying both the BSTR RawFullPath and IDispatch* IFactory arguments.

4. Click **Finish** when done.

You now have a template for your class, so only GD registration and the implementation of the HandleFile method remains. And what is also very important is that you now have a CLSID called IDR_MYCRAWLPLUGIN which you need to register your HandleFile Method at the Google Desktop Search. With this CLSID the Google Desktop Search can find the entry point for this Method.

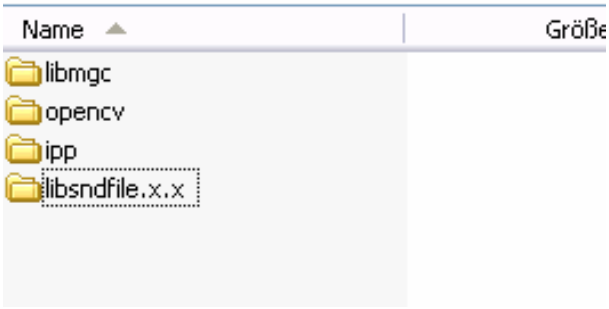
A.2 Visual Studio Projekt für libmgc

Aus den Source Dateien von Florian Lingenfelser habe ich eine dll für die Bentutzung unter Windows gemacht.

Diese Schritte werden benötigt:

1.) Starten Sie Visual Studio. Wählen Sie Datei → neues Projekt. Wählen Sie
   1.) Wählen Sie einen geeigneten Namen wie z.B. libmgc
   2.) Klicken Sie auf Application Settings und wählen Sie dll aus.
   3.) Klicken Sie fertig.

2.) Klicken Sie mit der rechten Maustaste ihr Projekt an und wählen Sie existierende Dateien hinzufügen.
3.) Klicken Sie mit der rechten Maustaste ihr Projekt an und wählen Sie die Eigenschaften. Gehen Sie zu C++ → Recompiled Headers und wählen Sie keine Precompiled Headers aus.
4.) Klicken Sie mit der rechten Maustaste auf die Eigenschaften ihres Projektes auf Eigenschaften. Gehen Sie zu C++ → Allgemein → Zusätzliche Include Pfade. Fügen Sie folgende Pfade Hinzu, wobei von folgender Verzeichnisstruktur ausgegangen wird.



- „..\opencv\ml\include“
- „..\opencv\cxcore\include“
- „..\opencv\cv\include“
- „..\opencv\otherlibs\highgui“
- „..\IPP\5.1\ia32\include“
- „..\libsndfilex.x“

5.) In der featureExtraction.cpp habe ich folgende Zeilen hizugefügt, da die Funktion log2f nicht Windows konform ist.

```
#ifdef WIN32
      #define M_LOG2_E        0.693147180559945309417
      #define log2f(x) (logf (x) / (float) M_LOG2_E)
#endif
```

Um alles include die die Datei dirent.h includieren habe ich auch den #ifdef Befehl geschrieben, da dieses include nur nötig ist für die fft und diese gibt es unter Windows nicht. Daher ist es in der momentanen Version der Bibliothek auch nicht nötig diese unter Windows zu trainieren.

```
#ifdef LINUX
      #include „dirent.h“
#endif
```

6.) Jetzt müssen noch die Drittbibliotheken gelinkt werden.

- libsndfile (Die Bibliothek libsndfile.lib muss erst noch kompiliert werden. Eine ausführliche Anleitung dazu finden Sie in der Readme.txt der Bibliothek)
- ippi.lib, ipps.lib, ippcore.lib, ippsc.lib, ippsr.lib (Intell IPP Bibliotheken)
- cv.lib, ml.lib, cxcore.lib (OpenCV Bibliotheken)

A.3 Installation des GDP

Die Installation habe ich mit NSIS einem Open Source Installer Tool realisiert. Diese kopiert alle nötigen Dateien an die richtige Stelle und registriert das Audio Indizierungsplugin bei der Google Desktop Suche. Um dies erfolgreich abzuwickeln sind folgende Schritte notwendig.

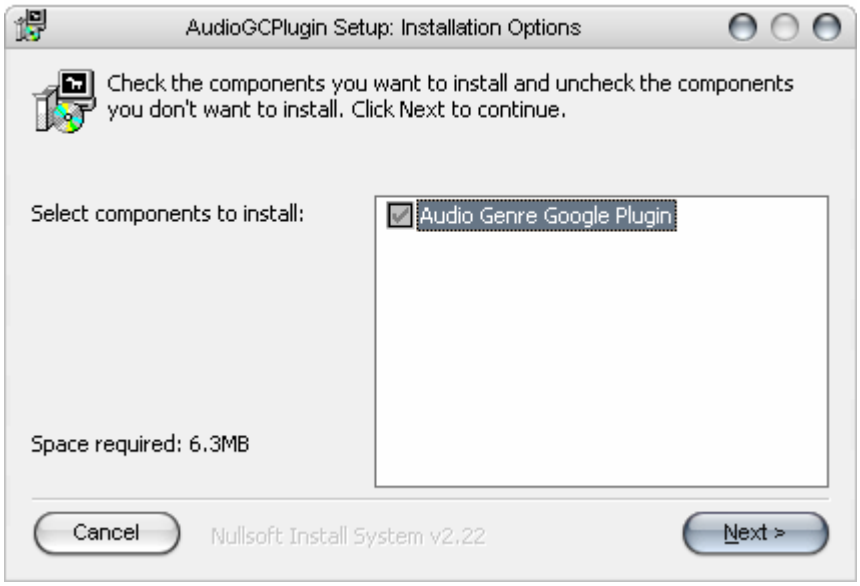
1.) Google Desktop Suche installieren

- diese kann entweder hier heruntergeladen werden: http://desktop.google.com/de/index.html
- oder direkt von der CD aus dem Verzeichnis GDS installiert werden

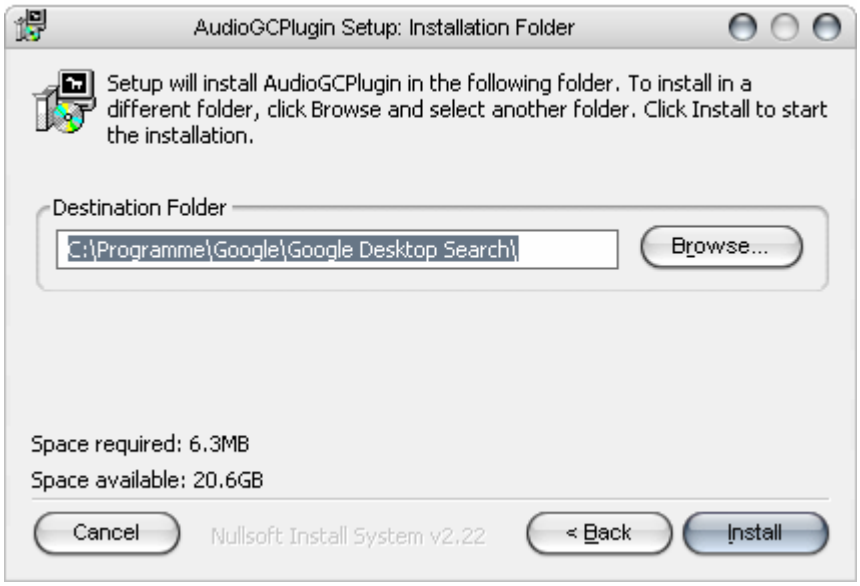
2.) Da das Plugin die Intel IPP Bibliothek verwendet müssen vorher die erforderlichen DLL Dateien, die auf der CD im Verzeichnis Intel-IPP-Dlls zu finden sind in das WINDOWS\SYSTEM32 Verzeichnis kopiert werden.

3.) Google Desktop Plugin installieren

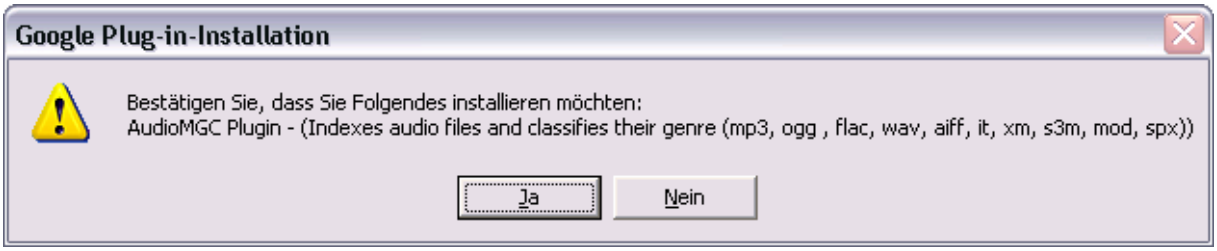
- dazu den Installer im Root der CD ausführen

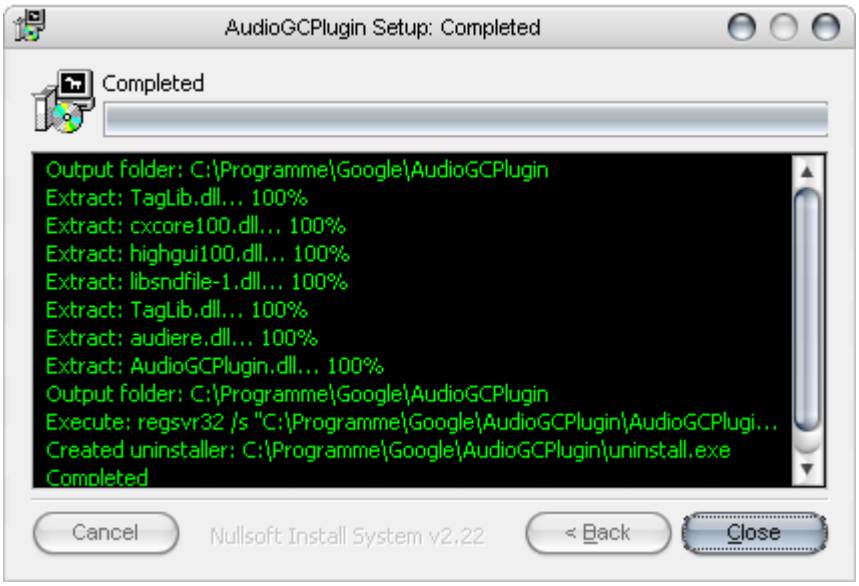


Next klicken



Ein Verzeichnis aussuchen in dem das Plugin installiert werden soll und dann auf **Install** klicken. Es ist wichtig, dass sie das Plugin wirklich genau in den Google Desktop Search Pfad installieren, da sonst die benötigten Trainingsdaten für den Klassifikator nicht gefunden werden.





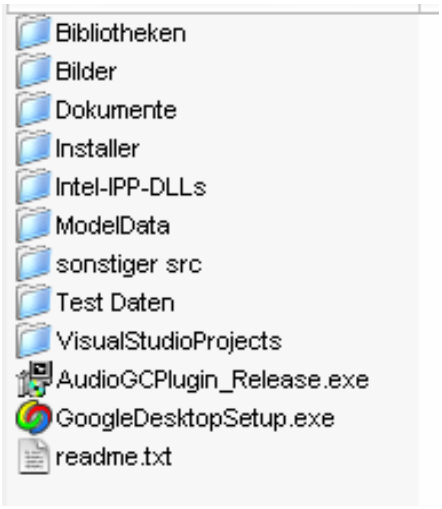
Sie werden noch gefragt ob die damit einverstanden sind, dass das Plugin auf ihrem System registriert wird. Klicken sie **OK** und dann können sie den Installer aud **Close** schließen

4.) Kopieren Sie alle Dateien in dem Verzeichnis ModelDat in das Root ihrer System Festplatte z.B.: Sollten Sie ihr Windows auf c:\ installiert haben kopieren Sie alle Model-Dateien nach c:\

5.) Die Google Desktop Suche neu indizieren oder sie kopieren einfach die Musikdateien, die auf der CD im Verzeichnis „Test Daten“ zu finden sind auf ihre Festplatte und warten 1 bis 2 Minuten. Danach können Sie nach Musik8 suchen und bekommen das gleiche Suchergebniss wie in Kapitel 5.1 beschrieben.

A.4 CD – Inhalt

Im Root Verzeichnis der CD finden sie folgende Verzeichnisse:



- Bibliotheken: enthält die originalen Bibliotheken
- Bilder: enthält alle Bilder, die ich für die Bachelor Arbeit verwendet habe
- Dokumente: enthält alle Dokumente, die ich elektronisch aus dem Internet habe
- Installer: enthält alle Dateien die nötig sind um einen Installer zu erstellen.
- Intel-IPP-DLLS: enthält die dlls der Intel IPP Bibliothek
- ModelData: enthält die trainierten Modelle für den Klassifikator
- Sonstiger src: enthält ein Implementierung in der die OGG Vorbis und die libmad Bibliothek benutzt werden
- Test Daten: enthält zu jeden Audio Format das vom Plugin unterstützt wird eine Testdatei
- VisualStudioProjects: Enthält das Visual Studio Projekt für das GDP in dem Unterverzeichnis *AudioGCPlugin* und das VisualStudioProjekt für die libmgc im Verzeichnis *libmgc.* Außerdem enthält es alle Bibliotheken, die zum kompilieren benötig werden noch einmal in ausgepackter Form.
- AudioGCPlugin_Debug.exe: installiert das Google Desktop Plugin als DEBUG Version bei der MessageBoxen aufpoppen.
- AudioGCPlugin_ohneKlassifikator: installiert das Google Desktop Plugin ohne den Klassifikator
- AudioGCPlugin_Release.exe: installiert das GoogleDesktopPlugin als RELEASE Version ohne MessageBoxen.
- GoogleDesktopSearchSetup.exe: installiert GoogleDesktopSearch auf dem Computer.

Literaturverzeichnis

1. **Google Desktop..** Google Desktop SDK. http://desktop.google.com .[Online] 2007., http://desktop.google.com/dev/searchapi.html

2. **Griffel, Frank.**. Componentenware: Konzepte und Techniken eines Softwareparadigmas. Dpunkt.verlag: 1. Auflage – Heidelberg 1998

3. **Gruhn, Volker und Thiel, Andreas..** Komponentenmodelle: DCOM, JavaBeans, Enterprise JavaBeans, CORBA. Addison-Wesley Verlag: 1. Auflage – München 2000

4**. Glogowski, Adam..** Microsoft COM-Familie, [Online] 2007 http://swe.uni-duisburg-essen.de/de/education/ws0506/cbsd/ausarbeitungen/COM-Familie.pdf

5. **Microsoft Cooporation..** msdn Deutschland. http://microsoft.com [Online] 2007, http://www.microsoft.com/germany/msdn/library/components/TippsZumUmgangMitDerIDL.mspx?mfr=true

6. **Tzanetakis, George..** Automatic Music Genre Classification of Audio Signals [Online] 2007, http://www-cse.ucsd.edu/classes/fa01/cse291/Audio.ppt

7. **Tzanetakis, George und Essl, Georg und Cook, Perry..** Automatic Musical Genre Classification Of Audio Signals [Online] 2007, http://ismir2001.ismir.net/pdf/tzanetakis.pdf

8. **de Castro Lopo, Erik..** libsndfile. http://www.mega-nerd.com/libsndfile/ [Online] 2007

9. **the Xiph open source community..** libvorbis. http://xiph.org/vorbis/ [Online] 2007

10. **Wheeler, Scott..** taglib. http://developer.kde.org/~wheeler/taglib.html [Online] 2007

11. **Audiere..** High-level audio API http://audiere.sourceforge.net/home.php [Online] 2007

12. **fmod..** music&sound effects system http://www.fmod.de/ [Online] 2007

13. **FLAC..** free lossless audio codec http://flac.sourceforge.net/ [Online] 2007

14. **speex..** a free codec for free speech http://speex.org/ [Online] 2007

15. **Intel Corp..** Intel Performace Primitives (IPP). http://www.intel.com. [Online]

16. **Intel Corp..** Open Source Computer Vision Library. http://www.intel.com. [Online] 2007., http:://www.intel.com/technology/computing/opencv/index.htm

17. **Intel Corp..** MKL. http://www.intel.com. [Online]

18. **TiMidity++..** software synthesizer http://timidity.sourceforge.net/ [Online] 2007

19. **Lingenfelser, Florian..** Bachelor Arbeit „Music Genre Classification“, Universität Augsburg 2007

20. **Castro de Lopo, Erik..** libsndfile http://www.mega-nerd.com/libsndfile/ [Online] 2007