Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

MEAMM — Movie Experience and Metadata Manager

Group Project

Damiano Boppart Erwin Herrsche

dboppart@ee.ethz.ch heerwin@ee.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Raphael Eidenbenz
Prof. Dr. Roger P. Wattenhofer

January 19, 2012

Abstract

Currently there are few applications available that help users organize the film file collection that they store on their computer. The focus of this project was to design and start developing a new application that links relevant metadata to the users film files in his collection, providing her with a powerful tool to keep her collection organized and manageable. As this application was built from scratch, much of the effort went into planning and building a solid foundation that allows for easy extension beyond the extent of what was accomplished in this group project.

This project consist of the following parts:

- This paper, outlining the reasoning behind design decisions made concerning the implementation,
- The source code of the application that was developed, which is called "MEAMM".

The source code is available through the web-based hosting service for software development projects Gitorious[1] at this location: `https://gitorious.org/meamm/meamm/trees/StrangerThanFiction`[2].

[1]`http://gitorious.org/`

[2]Please note that this link references a specific version of the application, the commit tagged with `StrangerThanFiction`. Later version might be available, a version history can be found at `https://gitorious.org/meamm/`.

Contents

Chapter 1

Introduction

Both authors of this paper have made a habit of watching films on their computers in recent years. As our collection of film files started to grow, we came to see that there are practically no applications available that can assist us in managing a collection of film files. Common, repetitive tasks that would be easy to automate have to be done by hand. The following paragraphs describe a couple of these tasks.

Often, after having watched a film, we are interested in seeing the actors that have played in the film to see if they appear in other films worth watching. Or, we would like to find a song from the soundtrack and to see whether a sequel is in production.

Whenever we want to look up details on a specific film we have on disk, we typically have to go online and use an encyclopedia such as Wikipedia[1] or a site that catalogs movies such as IMDb[2] to do so. We have what we want to know more about right in front of us — the film file — and yet we have to go somewhere else to find out more about it — online. Even worse, it does not suffice to do it once for every film file, it has to be done every time we want to look something up.

An application that could retrieve the relevant metadata on a film and store it locally linked to a file would save us the trouble of entering the films names into a search field in the browser every time we want to look something up.

As our film file collections have grown beyond what one HDD can hold, determining whether we have a film on disk and if so, where exactly it is located becomes an increasingly involved job: Do I still have it on the HDD of my laptop computer? Or have I moved it to my desktop PC? Did I copy it to my external HDD that my friend just borrowed over the weekend? Is my USB thumb drive even big enough to hold my film file if I find it? Having an overview of one's film

[1] `http://www.wikipedia.org`

[2] `http://www.imdb.com`

file collection becomes increasingly difficult as the number of involved devices and HDDs increase.

In this world of removable storage devices and network shares, just because a file is not found at the path in the file system it was last seen does not automatically mean that it has been deleted. An application that could keep track of these volatile files in our film collection would be of great help in answering the above questions.

As we were unable to find any existing application that can help us with problems such as these, we decided to write our own. As the software engineering proverb goes: “Use the best tool for the job — If it doesn’t exist, write it yourself.”

Chapter 2
Design

2.1 Inspiration

After having decided to write our own application, our first step was to study existing ones to find ideas or concepts we could reuse. We examined applications for traits worth considering in the design of MEAMM. We broke the interesting applications into three categories, Music Players, Tag Editors and Media Collection Managers, which are discussed in detail in the following sections.

2.1.1 Music Players

Music players is the most important category of applications that has influenced MEAMM. This is due to the fact that for many applications in this category, metadata has traditionally played an important role in how the user interface is structured and how the application is used. Many music players include a library or catalog for the user to keep track of his collection of files, and browsing the collection using metadata is ubiquitous for these applications. In fact, many applications in this category have embodied the concept of navigation by metadata so deeply that using them as a “front end” for managing files is complicated. Things such as “copy the audio files in this playlist into a specific folder” or “move these files in my library to a different location on the file system” are uncomfortable to accomplish at best. Music players seem to try their best to make the user deal with files exactly once: when importing files into the internal library.

Notable examples of applications that fit this description are GMusicbrowser[1],

[1]`http://gmusicbrowser.org/`

Banshee[2], Winamp[3] and iTunes[4].

Today, many music players have the capability of managing video files alongside audio files. They are, however, not yet adept at managing film files as there are several important differences between video and audio files.

Audio files are generally containers that permit the storage of metadata in addition to the audio content. In practice, music players make the supposition that this feature is made use of. This assumption reduces a music players library to a mere cache of data found within the files it tracks. However, this assumption does not generally hold for film files. Many containers do not offer the possibility to store metadata within the file, and in containers that do, the feature is simply never used. There is also no industry standard defining the format of such metadata. Music players are often not designed to retrieve metadata from anywhere but within the file, so files which do not incorporate metadata are considered not to have metadata at all. Another circumstance that is often not respected by these music players is that one film might consist of several files or one file might contain several films. For these reasons, music players were useful making decisions concerning the user interface, but not the back end.

2.1.2 Tag Editors

Tag Editors are applications that assist the user in editing the metadata stored in audio files. As the presence and quality of this metadata is crucial to music players as has been discussed above, some functionality found in the stand-alone tag editors in this category is often also found built into music players themselves. Many tag editors make it easy to use a web resource to retrieve metadata from to be stored in the audio files. Some offer the possibility to find the correct record in a web database based on incomplete metadata already present in the files.

Notable examples of applications in this category are Picard[5] and Mp3tag[6].

As tag editors are generally designed to handle audio files only, they make the same assumptions that music players have on the nature of the files they can handle. As has been discussed for music players, this makes tag editors a reference for user interface design decisions only.

[2]`http://banshee.fm/`
[3]`http://www.winamp.com/`
[4]`http://www.apple.com/itunes/`
[5]`http://musicbrainz.org/doc/MusicBrainz_Picard`
[6]`http://mp3tag.de/`

2.1.3 Media Collection Managers

Media collection managers, often also called DVD collection managers, are applications designed to catalog physical media that contains films such as DVDs or Blu-ray discs. The user interface in essence consists of a browser, similar to the user interfaces found in music players. Applications in this category are unaware of any files, as they are designed to organize physical objects. If a user were to use a Media Collection Managers to organize her video files it would require additional work, as she would have to manage the metadata stored by the collection manager in addition to managing the film files. Nevertheless, we consider applications of this category closest to what we envision MEAMM to be — a media collection manager that handles files where others handle references to physical media.

Notable examples of media collection managers include Griffith[7].

2.1.4 Conclusion

Even though each category of applications discussed above has severe shortcomings with respect to our goals, they have been of great help. Many design decisions regarding how the user interface of MEAMM is structured, or how a given task for the user is broken down into small steps, have been made a lot easier by the authors' familiarity with existing applications in the above categories.

2.2 Choosing a Metadata Source

One of the most important features of MEAMM is its ability to retrieve film metadata from a web-based database. We decided early on to structure MEAMM's code modularly in such a way that it would be able to support using multiple resources for metadata lookups. However, given the time constraints of this project, it was also decided that the implementation would only access one remote resource to begin with.

As the usefulness of MEAMM depends to a large part on the comprehensiveness and completeness of the metadata of the films it keeps track of, careful selection of the one data source to be used was of paramount importance.

From the list of possible candidates we compiled, we picked the three that looked the most promising and evaluated them in more detail.

[7] `http://griffith.cc/`

Table 2.1: Quantitative criteria of the evaluated web resources

Source	**Alexa rank**[c]	**# of films**[b]
IMDb	37	268571[c]
Linked Movie Database	3501526	85620[d]
Freebase	19463	192292[e]

[a] Alexa traffic rank, global; retrieved from `http://www.alexa.com/`
[b] The number of film entries in the resource. Only feature films considered where the statistics differentiated between types of films.
[c] retrieved from `http://www.imdb.com/stats`
[d] retrieved from `http://wiki.linkedmdb.org/Main/Statistics`
[e] retrieved from `http://www.freebase.com/view/film/film`

We considered the following criteria to evaluate the web resources:

- **Alexa global traffic rank and the number of feature film entries**
  These were the only two quantitative measures considered. We took them as a rough indicator of the comprehensiveness and completeness of the database, assuming that resources with more traffic and more film records were better.
- **Content licensing**
  The license of the content MEAMM was to retrieve and the terms of use of the interface to the service were also taken into consideration.
- **The API**
  Our most important criterion was, however, the design of the interface of the web resource to be used from within MEAMM. The nature of the interaction with the web resource and the data models employed obviously have a great impact on code complexity and performance of MEAMM.

The quantitative criteria are given in Table 2.1. A more detailed discussion of each candidate can be found in the following sections.

2.2.1 IMDb — `http://www.imdb.com/`

Given the above numbers, it is safe to say that IMDb is one of the most importance film reference works available online. It is also a website well-known to both authors and was therefore the most obvious candidate. However, IMDb does not offer any way for applications to directly interact with its service that would have been useful to us[8]. Also, the terms of use[9] were too restrictive for our taste. IMDb was therefore discarded as a candidate.

2.2.2 Linked Movie Database — `http://linkedmdb.org/`

The Linked Movie Database licenses its content under the Creative Commons Attributions License or the GNU Free Documentation License depending on the source[10]. Its API consists of a SPARQL[11] endpoint: `http://data.linkedmdb.org/sparql`. As neither of the authors had been familiar with this type of interface, and we were unable to find any comprehensive documentation on it, we discarded the Linked Movie Database as a candidate in favor of Freebase.

2.2.3 Freebase — `http://www.freebase.com/`

Content on Freebase is licensed under the Creative Commons Attribution License[12]. Freebase offers multiple APIs[13]. The main advantage of Freebase over the other contenders is its extensive documentation found on the website[14]. In addition, anecdotal evidence collected during the work on MEAMM suggests that, at least for U.S. film productions, the extent of metadata available on Freebase is comparable to that offered by IMDb. As at this point we considered Freebase to be "as good as it gets" we refrained from evaluating further web resources.

2.3 Data Model

2.3.1 Artwork

An "artwork" in our data model is defined to refer to the metadata associated with a single motion picture production, and encompasses all versions of it. We defined the term "artwork" because the terms "film" and "movie" are commonly used to refer to both a film's metadata and a medium carrying a film such as a DVD or file.

[8]All interfaces apart from the website itself are listed on `http://www.imdb.com/interfaces`

[9]`http://www.imdb.com/help/show_article?conditions`

[10]`http://wiki.linkedmdb.org/Main/Licensing`

[11]SPARQL = SPARQL Protocol And RDF Query Language

[12]`http://wiki.freebase.com/wiki/License`

[13]`http://wiki.freebase.com/wiki/Developers`

[14]for instance `http://www.freebase.com/docs/mql`

Table 2.2: Predefined attribute types

attribute type	**integer value**	**string 0**	**string 1**	**string 2**
title	priority	title string	language[a]	description
date		date string[b]	country[c]	description
person	person id	function	(character name)	
genre	priority	genre string		
keyword	priority	keyword string		
URL		URL	description	
image	priority	URL	description	
origin		country[c]		
language	priority	language[a]		
length	length in seconds	version/description		
relation	artwork id	type	description	
map id		web resource id		

[a] Language code as specified by ISO 639-3
[b] Date string as specified by ISO 8601
[c] Country code as specified by ISO 3166-1 alpha-2

An artwork is a set of attributes. Each attribute has a type, an integer value, a source and up to four string values. The semantics of the integer and string values depend on the attribute type. The source is used to track the origin of the attribute. Any artwork may contain zero or more attributes of any attribute type. We defined a set of attributes which are listed in table 2.2 and can be used to describe almost anything related to artwork metadata. The attributes defined so far do not make use of the fourth string. It is reserved for future use.

Table 2.3: Predefined attribute types

attribute type	**integer value**	**string value**
name	priority	name
URL		URL
gender		gender
birthday		date[a]
death day		date[a]
map id		web resource id

[a] Date string as specified by ISO 8601

Person

We decided to track persons separately to be able to store some metadata for them in a similar manner to artworks. Person attributes are therefore very similar to artwork attributes, except they only have one possible string value. Table 2.3 shows our predefined attributes and their value interpretation.

2.3.2 Manila[15]

Manilas are used to group files independent of their actual location within a file system. This allows the program to be aware of atomic groups of files for various actions such as copying and playing. There may be no manila without a file in it, and no file without at least one manila.

2.3.3 File

A file may refer to an actual file on a file system or to a disk in a shelf. Files are stored with their size, location and file type. To solve the problem of keeping track of files on removable drives, the location is stored as a hookpath.

Hookpath

Keeping track of files on removable drives is not trivial. The absolute path of a file may change. There are technologies to keep track of a removable drives within

[15]as in Manila folder, a file folder made out of Manila paper

several operating systems. They all introduce several layers of abstraction. Hooks and hookpaths are a solution which operate on top of the file system, therefore not introducing any new abstraction layer and staying independent of the underlying system.

A hookpath is a method to store files as a relative path and a reference to the base directory. A hookpath has the following form:

hook:{UUID (hook)}/relative path

The hook is stored in a hookfile in the corresponding base directory. The path of a file can then be determined by taking the path of the hookfile and the known relative path from the hookfile to the actual file. If the path of the hook is unknown or invalid, when the hookfile is found again all the files depending on that hook will be automatically recovered. This means that if a hook is no longer present, the files are considered to be temporarily unavailable. Only if the hook is present and the file associated with that hook is not, is the file is considered to have been deleted.

The local storage has a hookcache to cache the mapping from hooks to directories.

Chapter 3
Implementation

The program is made with Qt 4[1], a multi-platform GUI toolkit. It has integrated database and network modules.

To abstract the access to the local storage and web resources there are two base classes (`LocalStorageBase` and `WebResourceBase`) which define the interface. These interfaces are implemented for a SQLite local storage and the Freebase web resource.

SQLite is a serverless SQL database. The `SQLiteStorage` class acts as a wrapper between the "semantic web"-like data model and the relational database back end.

3.1 The Interface to Freebase

Of the APIs Freebase offered, we chose to use the one based on MQL[2]. This interface uses queries and responses that are structured as JSON[3] objects. They are sent and received using HTTP requests and responses.

A simple example query is shown in Listing 3.1, the corresponding response is shown in Listing 3.2.

Generally speaking, a query consists of an object (represented using JSON) with some known properties (i.e. key-value-pairs where the value has a sensible value) and keys with values that are meaningless (i.e. `null` or {}). The "empty" values are then replaced by data from the database, and an object with the same keys is returned.

This query uniquely identifies an object in Freebase's database by specifying

[1]`http://qt.nokia.com/`
[2]MQL = Metaweb Query Language
[3]JSON = JavaScript Object Notation

Listing 3.1: A simple query example

```
[{
  "type": "/film/film",
  "id":   "/en/the_dark_knight",
  "name": [{
    "lang|=": [
      "/lang/en",
      "/lang/de",
      "/lang/fr"
    ],
    "lang":     null,
    "value":    null,
    "optional": true
  }]
}]
```

its `type` as `/film/film` and its `id` as `/en/the_dark_knight`. It asks for a possibly empty array of objects that have the key `name`, whose value is an array of objects. Each of these `name` objects has the key `lang` with a value of either `/lang/en`, `/lang/de` or `/lang/fr`. For each `name` object that matches these requirements, the `name` object is to be returned containing exactly the two keys `lang` and `value` with their associated values retrieved from the database.

Freebase returns the actual response to the query together with some status information. The actual response is the value of the key `result`. It can clearly be seen that the structure of the object with the present keys exactly resembles that of the query, the keys `lang|=` and `optional` excluded.

In MEAMM the responses from Freebase are parsed using the `QScriptEngine` class.

Listing 3.2: The result of the query

```
{
  "code":           "/api/status/ok",
  "result": [{
    "id":   "/en/the_dark_knight",
    "name": [
      {
        "lang":  "/lang/en",
        "value": "The Dark Knight"
      },
      {
        "lang":  "/lang/fr",
        "value": "The Dark Knight : Le Chevalier noir"
      }
    ],
    "type": "/film/film"
  }],
  "status":         "200 OK",
  "transaction_id": "cache;cache03.p01.sjc1
     :8101;2012-01-18T22:17:54Z;0001"
}
```

Chapter 4

Discussion of Results

Although far from having all the features its users could desire, MEAMM at this stage is already sophisticated enough that it might be a useful tool to some users in practice. We successfully managed to implement some features related to every aspect of the intended use case.

Comparing the current version of MEAMM with the requirements originally specified in the agreement of this group project[1], we must however conclude that our original projection concerning the capabilities of MEAMM at the end of this project were a bit too optimistic. This is due to the fact that we invested more time in the planning and design phase which cut into the time left for actual implementation. However, we see this meticulous planning justified in the fact that during implementation we never encountered any serious design flaws made during the planning phase. Our data and logical structure devised for MEAMM proved to be sound. Being able to rely on a solid base pays off as a software project as it grows. We succeeded in establishing such a solid base.

[1]`disco.ethz.ch/theses/fs11/meamm.pdf`

Chapter 5
Outlook

We plan to continue to develop MEAMM in our spare time and implement the still missing features. Most important is polishing up the user interface, which would make the user experience smoother. More differentiated metadata should be imported from the web resource. Also export functionality for search results or selections is required. We would also like to see searching or adding an artwork based on a filename and adding directories to manilas. Also, the user should be able to perform actions on manilas, such as copying a manila. The hook handling would also benefit from refinement.

Some outstanding work might even be extensive enough as to warrant a group project in their own right:

- Extracting technical information from files
- Automatically adding artworks based on file (meta)data
- Automatically updating of manually-added artworks with incomplete data with data from a web resource
- Implementing importing metadata from a web resource other than Freebase

Chapter 6
Acknowledgements

The authors would like to thank our tutors Raphael Eidenbenz and Prof. Dr. Roger P. Wattenhofer: who accepted our proposal for this group project, allowing us to pursue our personal interests whilst getting credit for conducting academic work for the ETHZ; did not constrict us through strict deadlines or unnecessary bureaucracy; always made time for meetings on very short notice.

Damiano Boppart would also like to thank Philip Stark, who, through insightful contributions at a very, very early stage ensured that MEAMM matured from a vague dream to a proposal for a group project.