On-demand Metadata Extraction Network (OMEN)

Daniel McEnnis, Schulich School of Music, McGill University, Montreal

June 2006

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of the Arts in Music Technology.

Your file Votre référence
ISBN: 978-0-494-28567-1
Our file Notre référence
ISBN: 978-0-494-28567-1

ABSTRACT **3**

RÉSUMÉ **4**

ACKNOWLEDGEMENTS **5**

1 INTRODUCTION AND MOTIVATION **6**

1.1 DATASETS 6
1.2 ALTERNATIVE APPROACHES 7
1.3 SUMMARY OF A SOLUTION 9
1.4 OVERVIEW OF THESIS STRUCTURE 10

2 RELEVANT WORK **12**

2.1 EXISTING DIGITAL PUBLISHING PROJECTS 12
2.2 EXISTING MUSIC ANALYSIS PROJECTS 14
2.3 EXISTING DISTRIBUTED DATABASE PROJECTS 15
2.4 EXISTING DISTRIBUTED COMPUTING PROJECTS 17
2.5 EXISTING BENCHMARKING PROJECTS 20

3 JAUDIO **23**

3.1 23
MULTI-DIMENSIONAL FEATURES 23
3.2 TYPES OF FEATURES 24
3.3 EMBEDDING JAUDIO 26
3.4 DEPENDENCY RESOLUTION 27
3.5 OVERVIEW OF SETTINGS 27
3.6 DESIGN DECISIONS 28

4 FEATURES **30**

4.1 TIME-DOMAIN FEATURES 31
4.1.1 Root Mean Square *31*
4.1.2 Relative Difference Function *31*
4.1.3 Zero Crossings *31*
4.1.4 Strength of Strongest Frequency via Zero Crossings *32*
4.1.5 Linear Predictive Coding *32*
4.2 MULTI-WINDOW TIME-DOMAIN FEATURES 32
4.2.1 Fraction of Low-Energy Windows *33*
4.2.2 Beat Histogram *33*
4.2.3 Beat Sum *33*
4.2.4Strength of Strongest Beat *33*
4.3 FREQUENCY-DOMAIN FEATURES 34
4.3.1 Power Spectrum *34*
4.3.2 Magnitude Spectrum *34*
4.3.3 Strongest Frequency via FFT Maximum *34*
4.3.4 Spectral Centroid *34*
4.3.5 Strongest Frequency via Spectral Centroid *34*
4.3.6 Method of Moments *35*
4.3.7 Spectral Rolloff *35*
4.3.8 Mel-Frequency Cepstrum Coefficients (MFCC) *35*
4.3.9 Peak Detection *35*
4.3.10 Partial-based Spectral Centroid *36*
4.3.11 Partial-based Spectral Smoothness *36*
4.3.12 Compactness *36*
4.4 HYBRID FEATURES 37
4.4.1 Spectral Variability *37*
4.4.2 Spectral Flux *37*

4.4.3 Simplified Spectral Flux *37*
4.5 METAFEATURES 38
4.5.1 Derivative *38*
4.5.2 Running Mean *38*
4.5.3 Standard Deviation *38*
4.5.4 Derivative of Running Mean *38*
4.5.5 Derivative of Standard Deviation *38*
4.6 GENERAL AGGREGATORS 39
4.6.1 Mean *39*
4.6.2 Standard Deviation *39*
4.6.3 MFCC *39*
4.7 SPECIFIC AGGREGATORS 39
4.7.1 Area Moments *40*
4.7.2 Multiple Feature Histogram *40*

5 WEB INTERFACES 41

5.1 RESEARCHER INTERFACE 41
5.2 LIBRARIAN INTERFACE 46
5.3 ADMINISTRATOR INTERFACE 50

6 NETWORK STRUCTURE 55

6.1 MASTER NODE 56
6.1.1 Web Services *57*
6.2 LIBRARY NODE 58
6.2.1 Web Services *58*
6.3 WORKER NODE 59
6.3.1 File Cache *60*
6.3.2 Web Services *60*

7 LEGAL ISSUES 62

7.1 CANADIAN COPYRIGHT LAW 62
7.2 USA'S COPYRIGHT LAW 62
7.3 CONCLUSION 63

8 CONCLUSIONS AND FUTURE WORK 64

APPENDIX A – USE CASE DIAGRAMS 65

APPENDIX B – JAUDIO DESIGN 69

B.1 GLOBAL STRUCTURE 69
B.2 DATAMODEL CLASS 69
B.3 EXECUTION OF METADATA EXTRACTION 70
B.3.1 Construction *71*
B.3.2 Dependency Resolution *72*
B.3.3 Extraction *73*

APPENDIX C – GLOSSARY 74

Axis *74*
Cache *74*
Dynamic-linked Library *74*
Ground Truth *74*
Servlets and JavaServer Pages *74*
Tomcat *75*
Web Services *75*
XML *75*

REFERENCES 77

Abstract

OMEN (On-demand Metadata Extraction Network) addresses a fundamental problem in Music Information Retrieval: the lack of universal access to a large dataset containing significant amounts of copyrighted music. This thesis proposes a solution to this problem that is accomplished by utilizing the large collections of digitized music available at many libraries. Using OMEN, libraries will be able to perform on-demand feature extraction on site, returning feature values to researchers instead of providing direct access to the recordings themselves. This avoids copyright difficulties, since the underlying music never leaves the library that owns it. The analysis is performed using grid-style computation on library machines that are otherwise underused (e.g., devoted to patron web and catalogue use).

Résumé

OMEN (On-demand Metadata Extraction Network, pour Réseau d'Extraction de Caractéristiques à la Demande) touche un problème fondamental en Recherche d'Information Musicale : l'inexistence d'un accès universel à un large ensemble de données musicales protégées par des droits d'auteur. La solution envisagée propose d'utiliser les grandes collections de musique numérisée disponibles dans de nombreuses bibliothèques. En utilisant OMEN, les librairies pourront effectuer une extraction de caractéristiques à la demande sur leur site, retournant aux chercheurs les valeurs calculées plutôt que de fournir un accès direct aux enregistrements eux-mêmes. Ceci évite les problèmes liés au droits d'auteur puisque la musique de laquelle sont extraites les caractéristiques ne sort par de la bibliothèque qui la possède. L'analyse est effectuée en utilisant des ressources de calcul en réseau sur les machines de la bibliothèque qui ne sont pas ou peu utilisées à ce moment (par exemple les machines dévouées à l'accès web ou au catalogue).

Acknowledgements

I would like to thank Ichiro Fujinaga for his unflagging support, especially the countless hours reading and providing insightful comments and suggestions through numerous revisions of this thesis. I would also like to thank Cory McKay for his assistance in implementing jAudio.

1 Introduction and Motivation

Music Information Retrieval (MIR) is a new and rapidly expanding research field, involving various tasks to analyzing and organizing music. Unfortunately, it is currently difficult for MIR researchers to effectively evaluate their algorithms. The primary obstacle to achieving this is the inability for the researchers to obtain uniform access to a large dataset of audio files that is representative of the music currently listened to. Access to music currently listened to is important for ensuring results are applicable to real world problems.

The On-demand Metadata Extraction Network (OMEN) seeks to resolve this difficulty by altering how MIR research is conducted—creating an international network of libraries that utilize existing CD collections and presenting these collections as a single unified dataset without violating copyright law or requiring MIR researchers to download the dataset. A prototype of this system has been implemented and deployed within the DDMAL lab in the music technology department of the McGill school of music.

1.1 Datasets

While the many different kinds of tasks in MIR have different requirements, there are some common factors. In general, MIR researchers want to make sure that their algorithms (accomplishing whatever task they are working on) will work on the music that a typical listener would listen to. Furthermore, many approaches to these tasks require a substantial number of pieces of music to work effectively. In addition, MIR researchers wish to make sure that their algorithms make as few assumptions about the

nature of the music analyzed as is feasible for their task, requiring a wide range of music as well. Finally, the dataset must be available for all MIR researchers to access.

Currently, there is a lack of quality datasets. There are two reasons for this difficulty: the amount of data required and copyright restrictions on the music. Since researchers require a large number of pieces for some algorithms as well as a diverse range of music, a large dataset is required. To put this requirement in perspective, most MIR researchers utilize "small" datasets consisting of approximately 20GB of data. This is roughly equivalent to 30 CDs of audio. These datasets are too small to contain a good cross-section of all music. Furthermore, most recorded music is copyrighted. As a consequence, every time a dataset is copied, all copyright holders in the collection must be compensated. This makes legally copying a large dataset prohibitively expensive as well as time consuming. If these datasets are to be downloaded over the Internet, we are presented with another potential problem related to network bandwidth. Thus, both the cost and difficulties involved in providing access to a good dataset mandates a different approach than copying the dataset for each researcher.

One solution to the dataset problem is to utilize the existing collections of CDs at libraries. The copyright costs have already been paid and the materials are already available to researchers who either use the materials in the library or check them out. However, this does not solve how to provide access to this dataset for MIR researchers around the world.

1.2 Alternative Approaches

Since copying the dataset is not feasible, alternative ways of allowing researchers access to the dataset should be explored. There are several models to choose from. One

option is a centralized site where MIR researchers send their algorithms to be executed on supercomputers from start to finish. Since the data is never copied, there is no copyright charge. However, it requires that one site acquire all the music in the dataset—a daunting financial challenge—and provide all computer time.

The other two approaches take advantage of the natural split between analysis and feature (metadata) extraction (see above). By extracting the features on site and sending the resulting metadata back to researchers for analysis, one also never copies data. This greatly lessens the computational load on the hosting sites since participating libraries only carry out a portion of the computation. Furthermore, this approach permits a distributed dataset—enabling use of multiple libraries' music.

One implementation of this approach is to pre-compute all features for all music in the dataset and place the results on the Internet for download. However, a number of tasks require moment-by-moment metadata that makes the metadata many times larger than the music that generated it. Furthermore, there are different parameters and settings for how features are collected that create a combinatorial explosion of versions of the metadata generated for each piece of music in the dataset in order to guarantee that MIR researchers can access the features with their desired parameters. This makes the metadata collection for each file so massive that it cannot be reasonably stored. Furthermore, calculating all these versions requires prohibitive amounts of computer time. These factors make pre-computing metadata unfeasible.

An alternative to pre-computing is on-demand metadata extraction. This process avoids the problems of pre-computing by only performing calculations when they are requested. As a result only the subset of metadata actually used by researchers is created.

While previously generated results can be cached for speed, they can also be discarded (and possibly recomputed later) if the hosting server runs low on disk space. However, while less extreme than pre-computing, on-demand computing still requires significant computer time from libraries. Fortunately, libraries already have this computing power available in the form of under-utilized computers provided to allow patrons to search the library catalogs and surf the Internet.

1.3 Summary of a Solution

In order to implement an on-demand metadata extraction network, one needs to solve a number of problems. One is that the distributed dataset needs an online web interface that would allow MIR researchers mechanisms for selecting what subset of the dataset to be used for their experiments. A library of feature extraction algorithms is needed for generating the features. There also needs to be a mechanism for implementing the dataset as a distributed database. Finally, grid computing is required to distribute the computation load inside libraries.

In order to address these problems, infrastructure is needed that will make it easy for libraries to deploy this service. OMEN (On-demand Metadata Extraction Network) provides this infrastructure. OMEN will consist of three levels. The first will be a global level linking the various participating libraries into a common portal website where researchers can access diverse datasets as a single entity. The second level will be the library portal where a master computer for the library will provide an archive of all music made available for analysis and coordinate all analysis conducted at the library. The third level will consist of 'worker' library computers that can perform analyses under the

supervision of the library's master computer using grid computing. The relationships between these levels of infrastructure are summarized in Figure 1.

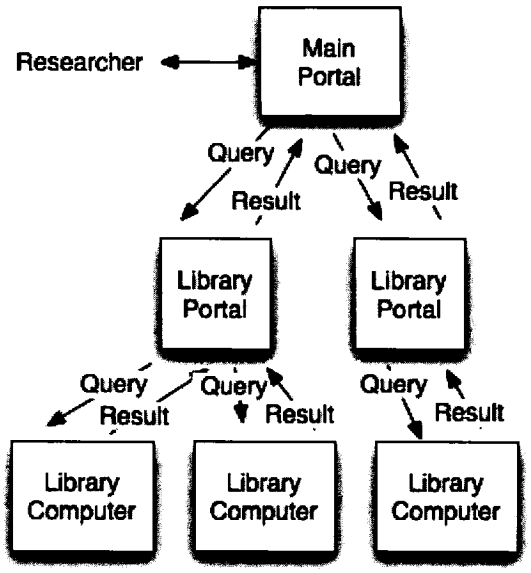


Figure 1: Relationships Between Infrastructure Levels

Analyses will be conducted using the jAudio (McEnnis et al. 2005) library. This infrastructure will permit librarians to merge their existing CD archives into a single large dataset that will be available to researchers for analysis as a single archive without requiring copying of the underlying datasets, solving a major problem for MIR researchers.

1.4 Overview of Thesis Structure

This thesis provides an in-depth description of OMEN. The first two chapters provide an introduction—a description of the problem and a description of relevant research. Chapters 3 through 6 provide an in-depth description of OMEN. The remainder of the thesis consists of appendices and a glossary—peripheral information pertinent to OMEN.

Chapter 2 describes relevant research in this area. This research is divided into five categories—digital publishing research, music analysis research, distributed database research, distributed computing research, and multimedia benchmarking research.

Chapter 3 describes jAudio. The major features and innovations of jAudio are discussed here. Chapter 4 describes the features present in jAudio. These features are divided into six categories—time domain features, multi-window time-domain features, frequency-domain features, and hybrid features, general aggregators, and specific aggregators.

Chapter 5 provides a description of the web interfaces provided by OMEN. These interfaces are the researcher interface, the library interface, and the administrator interface. Chapter 6 describes the different classes of computers in OMEN. These classes are Master Node, Library Node, and Worker Node.

Chapter 7 discusses legal issues involved in the deployment of OMEN. Chapter 8 gives an overview and discusses future work. Appendix A includes a collection of use-case diagrams that illustrate the execution steps in OMEN for different tasks. Appendix B provides a detailed description of how jAudio performs feature extraction. Appendix C provides a glossary of concepts and software packages described in the thesis.

2 Relevant Work

OMEN touches on a number of different disciplines. This includes publishing of datasets with their metadata, audio analysis software, distributed computing, distributed databases, and the efforts of other benchmarking projects to produce good datasets.

2.1 Existing Digital Publishing Projects

Constructing a friendly web interface to both access information about the OMEN dataset and load new audio files and their metadata into OMEN is an important part of making this dataset useful to MIR researchers. There exist a number of packages, both open source and commercial, that provide friendly interfaces for loading new pieces of data and their associated metadata as well as providing means to publish this data to the Internet. Some, like Greenstone and Streetprint, are open-source projects aimed at libraries. Others are traditional cataloguing software like Collectorz products and The Book Collection. Finally, there are new web-page commercial efforts such as LibraryThing.

Greenstone is a digital publishing tool created by Waikato University (Witten et al. 2000). It is a full-featured software package for creating a web site for searching and browsing a digital collection. It is geared towards libraries publishing existing digital collections, particularly libraries with extensive metadata about their holdings. It has a particular focus on text with a sophisticated search engine for searching text within a repository. The current version supports the importing of pre-existing metadata in MP3 music files. A new version, version 3 (Bainbridge et al. 2004), is currently in alpha and is implemented using web services that permit distributed libraries.

Streetprint was originally designed by the University of Alberta to publish British street literature (Ogle 2005). Similar to Greenstone, it provides an easy way to take a collection of existing digital documents with metadata and create a web site for interacting with the digital repository. Streetprint also provides support for a 'media type' for displaying multimedia content in a collection.

The Collectorz family of commercial products (Harte and Hoogerdijk 2006) also provides a way to take a collection and publish its contents. This product is geared towards home users who wish to catalogue (and publish online) their own collections. It has some relatively sophisticated mechanisms for acquiring metadata (such as scanning barcodes on CDs). Unfortunately, the web sites generated are static and have no search functionality.

The Book Collection software package (Book Collection 2006) also provides mechanisms to catalogue and manage collections of books. It is aimed at home users and small organizations with a variety of ways to acquire metadata on books using information such as barcodes or ISBN numbers. It differs from other commercial packages in that it provides mechanisms for creating a dynamic web site that uses the Book Collection database to provide searching and keep the content up to date.

LibraryThing (Spalding 2006) takes a different approach with an entirely web-based system where users upload information about their books to create an online record of their collection. It is aimed at users and provides various different means of searching the metadata present (both the metadata from the publishers and the metadata manually attached to the book entries themselves).

2.2 Existing Music Analysis Projects

While OMEN utilizes jAudio to extract metadata (features) from music files (see Chapter 3), there are a number of other systems that also complete a similar task. Each project approaches this extraction process differently.

Marsyas was first published in 2000 as C++ library for performing analysis on music files (Tzanetakis and Cook 2000). It is the first widely used library of this type and provided a library plus sample applications. Not all features present in the library are present in the sample applications. The design is modular—adding new features is relatively easy. However, unlike jAudio, changes require recompilation and dependencies between features must be explicitly calculated by the user.

Clam is a C++ application produced by the Universitat Pompeu Fabra (Amatrain and Arumi 2002). It is a GUI-based application that does provide metadata extraction capabilities. However, the main focus of the project is on analysis/synthesis.

M2K is a Java based application produced by the University of Illinois (Downie et al. 2005a). It is a GUI prototyping environment built on top of the D2K (ref). It is relatively easy to add new features to M2K without requiring a recompilation of the M2K application. Furthermore, applications constructed using M2K are interpreted so no compilation is needed when executing experiments. Unfortunately, while M2K is open source, D2K is a commercial application and only available with a commercial or academic license.

Maaate is a C++ application produced by the Commonwealth Scientific and Industrial Research Organization (Pfeiffer et al. 2005). Its primary purpose is to extract features already encoded in an MPEG-1 audio file. The system provides access to the

originally encoded features, all intermediate steps in the decoding process, and the final output. However, this extraction is geared more towards visualization than analysis.

The final analysis project is the implementation of the MPEG-7 metadata descriptors by Sikora (2004). While a description of these features exists (Quackenbush and Lindsay 2001) a more complete description of the features is located online (de Mallorca 2004). This application is not available for download. However, a web form is made available to submit an extraction request remotely. Source code for this implementation is not available.

2.3 Existing Distributed Database Projects

Since OMEN distributes its dataset across a number of different locations, it faces different challenges than a more traditional centralized dataset. There exist a number of different projects that tackle the problems inherent in accessing, indexing, and searching distributed datasets. The approaches vary widely, but all deal with distributed datasets of multimedia content. On one extreme is www.archive.org where all data is transferred from the source (the Internet) to a centralized network of computers. On the other extreme, a multimedia database leaves all data and metadata on the disparate servers, instead transforming queries into a distributed query forwarded to each databases for processing.

The www.archive.org database is run by the Internet Archive non-profit organization devoted to documenting the Internet (Internet Archive 2006). It utilizes a custom search engine to harvest data from the Internet and stores it in a large cluster of servers called the Wayback Machine. Multimedia data is also archived. Furthermore,

there is a separate audio site that permits downloads of bands that permit distribution of live performances.

Researchers can request shell access to the database cluster of the Wayback archive. While no analysis software is made available, users are permitted to upload their own analysis software. Unfortunately, there is no software provided for parsing the archive files or identifying which of the files contained in the archive file are music.

Another example is a prototype developed by Petkovic and Jonker (2004). This is a database system designed to process video using a collage of different analysis techniques, combining the results of these techniques into a single database system. The problems this prototype seeks to solve are the need to automatically extract and make available for querying a variety of dissimilar features and coping with massive storage requirements and extremely expensive insertion-time calculations. The prototype solves this problem by using distributed database servers that each includes metadata extraction software that is executed at the time that a video is added to the database. The metadata generated, however, is stored in a single central database.

The EGSO, VSO, and CoSEC solar data archives all seek to connect massive widely separated databases of solar images. EGSO is a coalition of European solar observatories. VSO is a coalition of US-based solar observatories. CoSEC is a corporate partnership featuring Lockheed Martin. All three of these projects face the same problem—how to identify what data researchers need from these massive widely distributed databases and get that data to the researchers in the form that they require.

EGSO and VSO utilize a central portal that collects the queries presented to it and then distributes these queries to all participating solar observatories (Messorotti et al.

2003, Gurman et al. 2005). One feature planned for EGSO but not yet implemented is to permit users to upload their own software for identifying what images to return. Finally, the relevant search results are returned.

CoSEC implements a different solution. It has a central broker that identifies all the different web services of each participating observatory (Hurlbert et al. 2001). Query requests are then forwarded directly to the observatories and the results of the search retrieved. In addition to search, the observatories also publish other analysis services that can be performed that are also accessible with a smart client.

Finally, there exists a prototype created by Berthold (2002) for her PhD thesis. This thesis focused on integrating multimedia objects into a distributed database system. The problem to be overcome includes massive amounts of data and the need for efficient access. The system assumes that all metadata is known in advance. What is unique about the solution presented is that the central access point to the database retains no data or metadata. All queries are transformed into a distributed query that is forwarded to each database system that then returns the results of the search to the controller.

2.4 Existing Distributed Computing Projects

OMEN utilizes distributed computing within each library to accelerate requests for feature extraction. There exist a number of projects that demonstrate different ways to accelerate on-demand metadata extraction through distributed computing. One possibility is massively parallel computing. Another possibility is through the use of clustering technology as demonstrated by the $(RS)^2I$ project. A third possibility is the use of peer-to-peer (p2p) technology such as used in sensor grids. P2p is a technology that allows a

network of computers to search for and exchange data without requiring a central computer to control who communicates with whom.

There are several examples of massively parallel computing where users donate time on ordinary desktop computers in order to facilitate a project. The first is the Great Internet Mersenne Prime Search (GIMPS), closely followed by distributed.net. Nearly all other massively parallel computing projects are now conducted using an application framework known as Berkeley Open Infrastructure for Network Computing (BOINC) that provides a toolkit and common infrastructure for constructing massively parallel computing projects. While these projects differ greatly in purpose, their solutions for handling distributed computing are standardized with the common infrastructure (Anderson 2004).

GIMPS started in January of 1996 searching for Mercenne primes (Lawton 2000). Each worker computer is assigned a number to analyze and returns the result of that calculation to a central server.

Distributed.net has two projects (Curzio et al. 2006). One project is cracking codes. Each computer is assigned a set of keys to process and communicate to the server if any of the keys decrypt the message. The other is calculating optimum Golomb rulers. Each computer evaluates whether each potential Golomb ruler in their assignment are Golomb rulers and then returns the results to the central server.

The BOINC framework is based on the seti@home project and provides mechanisms for easily creating and distributing a significant number of massively parallel projects beyond seti@home including Folding@Home and climateprediction.net (Anderson 2004). Participating projects provide their own data server and client code

while the maintainers of the BOINC project maintain the scheduling and management system for all projects that utilize their framework. All data and executables for analyzing the data for a task is first transmitted to a participating computer. Then the analysis is conducted. Finally, the results are returned to projects data server where the results are integrated into that project's data repository.

The $(RS)^2I$ project (Bretschneider and Kao 20002) uses a different approach to distributed computing. This project analyzes a large database of satellite images where the features used for searching may not be known until a query is received. Furthermore, the size of the image to be analyzed is variable, vastly increasing the computational complexity. To solve this problem, the $(RS)^2I$ project caches some metadata on the central server and employs a Beowulf cluster—a group of (usually identical) computers that are connected to one another by high-speed network connections—to perform feature extraction on-demand. To speed the process further, the set of images to be searched is pruned using metadata, then the resulting dataset is distributed around the cluster for processing.

Another approach is a fully p2p solution presented by Li et al. (2003). Their system for querying a grid of sensors connected by wireless, battery powered transmitters makes use of p2p techniques for transmitting queries to the relevant sensors and returning the results of the query. These sensors collect information, but the batteries would drain too quickly if the data were returned to the central source—especially since communication is the largest drain on the battery. In order to process these queries effectively and without draining the batteries, Li et al. developed a system for creating metadata at the nodes and accessing this metadata effectively without a central controller.

Finally, there is Globus, a toolkit for building grid applications (Foster et al. 2002). While not a distributed application in and of itself, it provides a set of tools for constructing applications that use grid computing that solves many of the problems of coordinating collaboration in distributed computing between organizations with different security and usage policies. A sample of these tools includes modules for ensuring security, modules for secure data distribution, and modules for scheduling of computation.

2.5 Existing Benchmarking Projects

Constructing good multimedia datasets is a common problem that all multimedia benchmarking projects must address. For each benchmarking project, the datasets must be large enough to provide adequate training sets for the algorithms tested as well as diverse enough to reach conclusions that are generalizable. There are a number of different competitions testing the performance of various multimedia search and classification algorithms involving various types of multimedia that have dealt with this problem. There is the TRECA video track, the Bencahathlon competition, the ImageCLEF competition, and the MIREX competition.

The TREC Video track was present in the 2001 and 2002 TREC conference (Smeaton and Over 2002). It expanded into its own competition separate from TREC starting in 2003 (Over et al. 2005). Initially, the database was constructed from video downloaded from the Internet Archive and Open Video Project. The database was later expanded to include copyrighted data available only to the participants. Initially, the competition tested a number of algorithms for their ability to match semantic descriptions to video shots (segments of uninterrupted video from a single camera) that were pre-

calculated. Each entry ran their algorithms against the database, marking video shots as either containing the query or not. Later competitions added additional tasks to the competition.

The Benchathlon network (also known as BIRDS-I) was a competition created and sponsored by HP. Initially, its goal was to evaluate algorithms for use in image finding tools used on mobile devices (Gunther and Beretta 2001). It used a single database that consisted of hand categorization of images acquired from the Internet. This was the basis for the first competition in 2001 at the Internet Imaging Conference. The project expanded into Benchathlon with additional contributions from other sources, leading to the Benchathlon competitions in 2002 and 2003 also at the Internet Imaging Conference.

ImageCLEF is another image benchmarking competition that is part of the Cross Language Evaluation Framework (CLEF) competition (Clough et al. 2004). It uses a single database consisting of copyright free images of St. Andrews University. The competition is unusual in that metadata is included in the competition—the captions for the images—but queries are placed in another language.

MIREX is the music information retrieval competition held at the International Conference on Music Information Retrieval (ISMIR) in 2005 (Downie et al. 2005b). It was preceded by the Audio Description Contest at ISMIR 2004. Each of the ten contests has their own database stored at the International Music Information Retrieval Systems Evaluation Laboratory (IMIRSEL) (Downie 2004). The data consisted of contributions made by participating researchers, some of which had copyright restrictions. This caused

difficulties distributing representative test sets that researchers could use to test their algorithms before submission.

3 jAudio

jAudio is a framework for feature extraction designed to eliminate duplication of effort in calculating features from audio signals (McEnnis et al. 2005). This system meets the needs of MIR researchers by providing a collection of analysis algorithms that are suitable for a wide array of MIR tasks. In order to provide these features with a minimal learning curve, the system implements a GUI that makes the process of selecting desired features straightforward. A command-line interface is also provided to manipulate jAudio via scripting, and jAudio may also be used as a dynamic linked library (a collection of sub-programs) for use in other applications. Design decisions and their consequences are discussed further at the end of the chapter.

OMEN utilizes jAudio to provide feature extraction capabilities. jAudio has a number of attributes that make it ideal for this purpose. jAudio provides support for multi-dimensional features. Furthermore, jAudio provides multiple types of features—features, metafeatures, and aggregators. In addition, jAudio was designed to be used as an embedded library, thus it has a sophisticated dependency resolution algorithm, intuitive settings for controlling extraction parameters, and an XML configuration file that permits users to add new features to it without modifying the dynamic linked library.

3.1 Multi-dimensional Features

Features extracted by jAudio are not restricted to one dimension. This is especially important for features such as MFCC's (see Chapter 5) that typically have many dimensions. Furthermore, in jAudio, the number of dimensions for a particular feature may not be known prior to extraction. This is especially useful for features such

as Magnitude Spectrum (see Chapter 5) where the number of dimensions is dependant on the window size. However, while the number of dimensions can vary based on the settings used, it cannot vary based on the input audio signal.

3.2 Types of Features

jAudio extracts three different classes of features—features, metafeatures, and aggregators. Features, including multi-dimensional features, are the fundamental type of information that jAudio processes and are calculated for each window. Metafeatures provide a template for creating new features derived from other features. These derived features are also calculated every window. Aggregators provide a way to control how jAudio produces per-song output from the per-window data.

Features are the basic unit of feature extraction. Features produce output on a per-window basis. Their input can be the digital representation of the audio signal being analyzed, the results of previous calculations, or both. All features, which require more than one window of data before output is possible, use a sliding window that permits jAudio to output for each window for these features. The output can be either uni-dimensional or multi-dimensional and the number of dimensions can be determined at execution time.

Metafeatures are feature templates that are automatically applied to all features to create new derived features. The primary purpose of metafeatures is to allow developers to produce common variations of features without having to explicitly code these variations for each feature. Derived features function exactly like regular features, and are also output on a per-window basis. jAudio provides three metafeature classes—Derivative, Mean, and Standard Deviation. In addition, metafeatures can be chained

together to create new metafeatures. jAudio provides two such chained metafeatures—Derivative of the Mean, and Derivative of the Standard Deviation. These five metafeatures can be used to provide up to five additional features for each standard feature. An example of this in practice: the feature Root Mean Square (RMS) can be automatically expanded from one to six features—RMS, Derivative of RMS, Running Mean of RMS, Running Standard Deviation of RMS, Derivative of the Running Mean of RMS, and Derivative of the Running Standard Deviation of RMS.

Aggregators are functions that collapse a sequence of vectors into a single vector or into a smaller sequence of vectors. This is similar to aggregate features introduced by Begrestra et al. (2006). Compared to metafeatures, aggregators produce output only on a per-song basis, not on a per-window basis. Aggregators take as input a sequence of vectors—the extracted windowed feature values over the entire input file—and output a single vector in its place. In essence, information on how a feature varies with time is collapsed into a single feature vector for each input file in its entirety. For the Standard Deviation aggregator, for example, each dimension of the output vector is the standard deviation of the values for that dimension in the input vector across all windows of the song. It is also possible to design aggregators that encapsulate more sophisticated and detailed information on how a feature varies from window to window.

Aggregators come in two varieties. The first is a function that is applied to the output of every feature to be saved. Mean and Standard Deviation are examples of this. The other type is a targeted aggregator that is applied only to one or more specific features. An example of this kind of aggregator is the AreaMoments Aggregator that, for a given order of input features, treats their combined sequence of vectors as a two-

dimensional image matrix and calculates two-dimensional moments from this matrix. This second type of aggregator is useful for acquiring information about how the features in a collection of features change together over time, a potentially very meaningful type of information that cannot generally be accessed with alternative feature extractors.

Users of jAudio can create their own aggregators of either type. Currently, jAudio has three aggregators that are applied to the output of every feature: Mean, Standard Deviation, and MFCC. There are also two features that are applied selectively: Multiple Feature Histogram and Area Moments.

3.3 Embedding jAudio

jAudio is designed so that it can be used either as a standalone application or as a dynamically linked library that can be embedded in other applications. Special emphasis has been placed on making it easy to add new features and aggregators to jAudio. jAudio provides an XML configuration file to identify and dynamically load all features and aggregators. In order to add a new feature or aggregator, one need only add a reference in the XML configuration file. There is no need to recompile jAudio after doing so.

In order to further accommodate the addition of features at runtime, jAudio automatically searches a plugin folder containing compiled Java classes for the implementation of the features described in the XML configuration file. The location of the plugin folder is specified in the first line of the XML configuration file as a URL. By using a URL instead of a path, it is possible to use a web site or other remote location to acquire the class files needed to construct features and aggregators. By modifying the configuration file and adding corresponding compiled Java classes to the plugin folder, it is possible to add new features while jAudio is still running. The compiled Java classes

that implement these features do not have to be in the plugin folder. Features whose implementations are already made available to jAudio through other means (i.e., via the classpath) are incorporated automatically.

3.4 Dependency Resolution

An additional unique feature of jAudio is its dependency resolution capabilities. Users do not have to know what the dependencies of a feature are when choosing which features to extract. Before the extraction begins, jAudio silently determines which additional features must be calculated in order to satisfy all the dependencies of the chosen features. These additional features are only used internally, and do not appear in jAudio's output. All features to be included are then automatically ordered to ensure that a feature is only calculated after all of the features that it depends on have first been calculated. An example of this is the Moments feature—it depends on the Magnitude Spectrum feature. As a result, for every window in which Moments is calculated, it occurs after Magnitude Spectrum is calculated. This algorithm is described in detail in Appendix B.

3.5 Overview of Settings

jAudio uses a number of user definable settings that influence how features are extracted. These settings can be roughly grouped into three areas—file types and sample rate, window size and window overlap, and output types.

jAudio supports a wide variety of file types by utilizing the JavaSound libraries included in Java. This makes it possible to process Wave files, AIFF files, Au files, and Snd files. In addition, jAudio utilizes an additional library to provide support for MP3

files. Through use of Tritonus Java plugins (Tritonus 2005), jAudio also has support for re-sampling the input signal to a new sample rate. jAudio allows users to determine this sample rate, which is then held constant for all files analyzed—regardless of the sampling rate of the input files.

jAudio also provides control over the size of the analysis window and the degree of overlap between adjacent windows. This setting is set once for all features and all files. (Permitting separate window sizes for each feature interferes with dependency resolution since features can be dependant on other features with a differing window size.) By default, the analysis window is 512 samples in length with no overlap.

jAudio also permits users to choose one of two output file formats—ARFF or ACE. These can serve as input for machine learning packages. ARFF is the data format used as input by the Weka machine learning system (Witten and Frank 2005). ACE is the input format used by the ACE machine learning system (McKay et al. 2005). The user may also choose whether output should be generated per-file (using aggregators) or per-window (not using aggregators). Due to limitations in the ARFF format, it is not possible to produce both per-window and per-file output at the same time when producing ARFF output.

3.6 Design Decisions

jAudio was designed to meet a number of different criteria. The system should provide a plugin interface for adding features and aggregators. It should provide command-line access, provide a GUI interface, and be easily embedded in other applications. Setting up feature extraction should be quick and simple. In particular, users

should not need to know the dependencies of the features they use. Furthermore, the computational burden should be reasonable and the code should be portable.

A number of implementation decisions were made to accommodate these design goals. In order to make configuring feature extraction simple and quick, features are chosen from a list instead of by constructing patches in (more complicated) graphical interfaces such as those used in CLAM (Amatrain and Arumi 2002) and M2K (Downie et al. 2005a). Similarly, window size and sample rate are fixed across all features. Allowing per-feature settings would require a more complicated interface that would be counter to simplicity. Automatic dependency resolution also resolves the biggest complication with existing systems—the need to explicitly include features whose results are not wanted but which are required for other features. By configuring each feature at most once, each feature need only be calculated once regardless of the number of times it is referenced, greatly reducing computational costs. By using Java, portability can be maintained. While there might be a concern over performance, Moreira et al. (2000) demonstrate that properly optimized Java code can achieve performance comparable to optimized Fortran.

4 Features

OMEN utilizes jAudio as a library for metadata extraction. jAudio provides implementations for a number of different features used to generate metadata. Features produce output for each window of the signal. Each of these features can be subdivided into four broad categories: those features that depend only on one window of a time-based representation of the signal (time-domain features), those features that depend on a vector of windows of a time-based representation (multi-window time-domain features), those that depend only on the frequency representation of a signal (frequency-domain features), and hybrid features that depend on changes in the frequency representation of a signal over time. Furthermore, jAudio provides aggregators that come in two varieties: general aggregators (that are applied to the output of all features) and specific aggregators (that are only applied to the output of specific features).

OMEN has an interface designed to make creating new features and aggregators as easy to create as possible while maintaining flexibility. In order to accomplish this, each feature has a standard description class where developers define information such as what other features have. Similarly, aggregators have a description class that defines the content of the feature. Keeping this information in the feature's or aggregator's source code (rather than a separate configuration file) eases development. Features have an extractFeature method that has as a parameter the PCM representation of the current window as well as all dependencies defined in the description. The Aggregators' aggregate method has as a parameter a list of the output generated by the relevant features' extract method for all windows. This structure shifts all decisions about whether or not there is enough information to compute a feature or include a window in the data

provided to an aggregator to jAudio's feature extraction routines. This is possible because jAudio is restricted exclusively to extracting features and so can fix decisions that would otherwise require researcher input.

4.1 Time-Domain Features

The output of time-domain features is calculated from exactly one window of the time-domain representation of a signal. Unless otherwise stated, the output each feature is a single value per window.

4.1.1 Root Mean Square

Root mean square (RMS) is the average energy per sample of one window of the signal.

$$\text{RMS} = \sqrt{\sum_{n=1}^{N} x_n^2}$$

Where x_n is the nth sample of the window.

4.1.2 Relative Difference Function

This feature is calculated by taking the log of the absolute value of the derivative of RMS (Klapuri 1999). For extremely small or zero values of the derivative, an arbitrarily small value is supplied instead.

4.1.3 Zero Crossings

This feature measures how many times the signal value crosses zero in a given window. The definition of 'crossing zero' is when the signal changes sign or changes from non-zero to zero.

4.1.4 Strength of Strongest Frequency via Zero Crossings

This feature transforms the unit of the zero-crossings feature from number of crossings per window into Hertz:

$$\text{Frequency in Hz} = \left(\frac{Z}{2}\right)\left(\frac{S}{N}\right),$$

where Z is the number of zero crossings, S is the sample rate, and N is the number of samples in the window. The output of this feature is an approximation of the pitch of the signal if the signal is monophonic.

4.1.5 Linear Predictive Coding

Linear predictive coding (LPC) provides a set of feedback filter coefficients that provides a compact representation of a window of samples. LPC is calculated as follows. First, create a matrix from the window data that is $L(W + L)$ where L is the number of coefficients and W is the length of the window of data. The first column is the contents of the window zero-padded with the data in each subsequent column shifted 1 row towards the bottom of the column (adding an additional zero at the top of the column) so that the last column starts with L zeros followed by the contents of the window. Then multiply the transpose of this matrix by itself to create a LxL autocorrelation matrix. This matrix is inverted and multiplied by a matrix that has an order $Lx1$ where the first entry is 1, and the rest 0. The resulting matrix is the LPC coordinates (Jackson 1999).

4.2 Multi-Window Time-Domain Features

These features are calculated over multiple windows of the time-domain signal. Unless otherwise noted, these features use a default super-window of 100 windows and output a

single value per window. A super-window is a window that consists of other windows. The length of all super-windows can be altered.

4.2.1 Fraction of Low-Energy Windows

This feature calculates the average value of the RMS over its super-window, and then calculates the percentage of individual windows whose RMS value is less than this mean value. This feature is useful for determining whether a signal has a stable, constant RMS, or has a tendency to have strong spikes.

4.2.2 Beat Histogram

This feature performs a Fast Fourier Transform (FFT) (McClellan et al. 1999, 371–4) against a super-window of RMS values in order to construct a histogram representing rhythmic regularities. A longer super-window of 256 windows is used to capture longer rhythmic regularities. This feature outputs a vector of values equal in length to the length of the super-window. This is used as a base feature for determining best tempo match.

4.2.3 Beat Sum

This feature is dependant on the beat histogram. It is calculated as the sum of all the entries in the beat histogram's vector. This gives some measure of the prevalence of regular beats occurring in the signal.

4.2.4Strength of Strongest Beat

This feature is dependant on the beat histogram feature. Its value is the maximum value in the beat histogram.

4.3 Frequency-Domain Features

These features utilize the output of the FFT (McClellan et al. 1999, 371–4) provided in jAudio. All of these features are dependent on either the magnitude spectrum feature or the power spectrum feature.

4.3.1 Power Spectrum

This feature first calculates the FFT of a window of samples, then a single vector is created from its real and imaginary components by squaring both the imaginary and real part, and then adding the result together.

4.3.2 Magnitude Spectrum

This feature is the square root of the power spectrum.

4.3.3 Strongest Frequency via FFT Maximum

This feature provides the index of the maximum value in the power spectrum.

4.3.4 Spectral Centroid

This feature is the "center of mass" of the power spectrum. It is calculated by multiplying each value in the power spectrum vector by its index in the vector and dividing this result by the sum of all entries in the vector. The resulting spectral centroid is the index in the vector where exactly half of the energy in the power spectrum is located in indices above the spectral centroid and exactly half below.

4.3.5 Strongest Frequency via Spectral Centroid

This feature transforms the unit of the spectral centroid from a vector index to Hertz.

4.3.6 Method of Moments

This feature consists of the first four statistical moments of the magnitude spectrum. This includes the area (zeroth order), mean (first order), spectral centroid (second order), spectral skew (third order), and spectral kurtosis (fourth order). These features describe the shape of the spectrograph of a given window (Fujinaga 1997).

4.3.7 Spectral Rolloff

This feature provides similar information to spectral centroid, but instead of providing the index where 50% of the energy is at a lower indices in a given power spectrum vector, this value calculates the index where 85% of the energy is located at a lower indices in the power spectrum vector. This default cutoff of 85% can be altered.

4.3.8 Mel-Frequency Cepstrum Coefficients (MFCC)

This feature is a concise description of a spectrum that is calculated from the magnitude spectrum (Bolger et al. 1963). MFCCs are calculated using the following steps: calculate the logarithm of each element of the power spectrum vector. Then convert the vector into Mels (a perceptually-based frequency scale). Finally, take the discrete cosine transform of this vector.

4.3.9 Peak Detection

This implementation is a simple algorithm whose output provides a vector of the peaks present in the frequency-representation of a signal. Peaks are defined as a local maximum in the magnitude spectrum where the peak's value is greater than one-tenth the value of the global maximum. It should be noted that this feature does not track peaks.

4.3.10 Partial-based Spectral Centroid

This feature is based on McAdam's (1999) spectral centroid algorithm used for instrument classification. It is calculated by multiplying each value in the vector of peaks by its index in the vector and dividing this result by the sum of all entries in the vector. This feature differs from a traditional spectral centroid in that peaks (listed in the paper as partials) are numbered sequentially and the spectral centroid is calculated on this vector of peaks rather than the power spectrum. In MPEG-7, this feature is called harmonic spectral centroid (Quackenbush and Lindsay 2001).

4.3.11 Partial-based Spectral Smoothness

This feature is an implementation of McAdam's (1999) Spectral Smoothness algorithm that operates on a vector of peak values. Using this vector, the spectral smoothness is calculated as follows:

$$\text{Spectral Smoothness} = \sum_{n=1}^{N-1} \log(P[n]) - \frac{\log(P[n-1]) + \log(P[n]) + \log(P[n+1])}{3}$$

where $P[n]$ is the nth peak vector and N is the length of the peak vector. It should be noted that if less than three peaks are present, this feature does not return a value.

4.3.12 Compactness

This feature is derived from McAdam's Spectral Smoothness (McAdams 1999). The difference between these features is that, instead of operating on a vector of partials, this feature operates on the magnitude spectrum. It is calculated as follows:

$$\text{Compactness} = \sum_{n=1}^{N-1} \log(M[n]) - \frac{\log(M[n-1]) + \log(M[n]) + \log(M[n+1])}{3}$$

where $M[n]$ is the nth entry in the magnitude spectrum's vector and N is the length of the vector. This feature is used to measure the degree of noise in a signal.

4.4 Hybrid Features

These features use a combination of both time-domain information and spectral information.

4.4.1 Spectral Variability

This feature is considers the magnitude spectrum as a population and calculates the standard deviation over the magnitude spectrum.

4.4.2 Spectral Flux

This feature is an implementation of McAdam's (1999) algorithm for spectral flux and is calculated using a vector of peaks. It is implemented as the Pearson's autocorrelation between two consecutive vectors of peaks. Peaks are matched by order of occurrence starting from the lowest frequency peak. If the number of peaks differs in consecutive windows of the frequency representation of the signal, the unmatched peaks are discarded. In MPEG-7, this feature is called harmonic spectral deviation (Quackenbush and Lindsay 2001).

4.4.3 Simplified Spectral Flux

This feature uses the sum of the squares of the difference between consecutive magnitude spectrum windows. It is used to describe the rate of change in the spectral content of the signal. It is identical to spectral flux except that it is based on the magnitude spectrum instead of a vector or peaks.

4.5 Metafeatures

Metafeatures are feature templates that can use the output of any feature and create a new feature from it. Like features, metafeatures produce output for each window of the digital signal. Currently, there are three metafeatures defined in jAudio: Derivative, Standard Deviation, and Mean. Furthermore, these metafeatures can be chained, creating more complicated features. In jAudio, five new features are generated from every feature.

4.5.1 Derivative

This metafeature implements a simple discrete derivative, returning the difference between adjacent values.

4.5.2 Running Mean

This metafeature provides the arithmetic mean of an arbitrary feature across a super-window.

4.5.3 Standard Deviation

This metafeature provides the standard deviation of an arbitrary feature across a super-window.

4.5.4 Derivative of Running Mean

This metafeature provides change in the running mean across a super-window. This feature is useful for tracking trends in any output that is inherently noisy.

4.5.5 Derivative of Standard Deviation

This metafeature provides change in the standard deviation across a super-window. This feature is useful in tracking changes in variability of a feature.

4.6 General Aggregators

General Aggregators are functions that take the output of features (which generate output for each window) and transform this into a single output per input file. General aggregators are automatically applied to every feature. However, general aggregators can only utilize the output of a single feature at a time.

4.6.1 Mean

This aggregator calculates the mean value of the underlying feature. If the feature is multidimensional, the mean is calculated for each dimension separately.

4.6.2 Standard Deviation

This aggregator calculates the standard deviation of the underlying feature. If the feature is multi-dimensional, the standard deviation is calculated for each dimension separately.

4.6.3 MFCC

This aggregator treats the underlying feature as a digital audio signal. By default, the feature output is treated as having a sample rate of 16KHz. The algorithm for calculating MFCCs for this aggregator is identical to the algorithm for the MFCC feature. If the feature is multidimensional, the calculations are performed on each dimension separately.

4.7 Specific Aggregators

Specific Aggregators are applied only to specific features instead of all features. Also, specific aggregators can combine the input of more than one feature at a time. All aggregators of this type must specify which features are to be used as input.

4.7.1 Area Moments

This aggregator first flattens all the features into a single array of output, concatenating the dimensions of the source features in order to create a single matrix of output values. Then the source is treated as a two-dimensional image and the two-dimensional statistical moments are calculated utilizing the same algorithm as the feature.

4.7.2 Multiple Feature Histogram

This aggregator is designed to track concurrent changes between features over an input signal. Each dimension is separately divided into bins. (i.e., 2 bins would be bottom 50% of values and top 50% of values). Each window is assigned an identifier according to the combination of bin values for each dimension in the input array for that window. The output is a histogram of these identifiers.

5 Web Interfaces

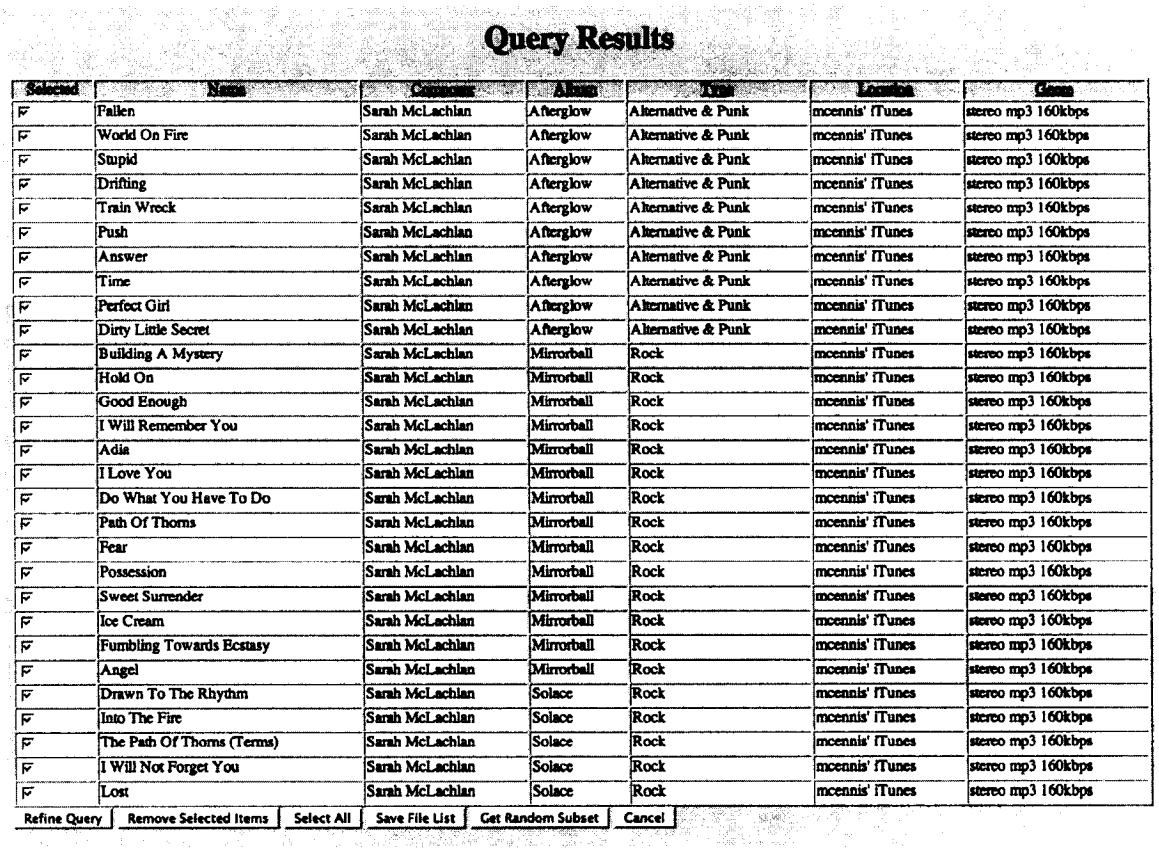
OMEN provides three different web interfaces for each of the three different kinds of users that will use it. Each of these interfaces serves a different purpose and meet different design criteria. The first type is a researcher. This is a person that will conduct experiments that will use the content of the library's collection. The second web interface is the Librarian Interface. This interface is for librarians to control access to their archive and to keep the centrally stored metadata about the music library's collections up to date. The final interface is the web Administrator Interface. This interface is for maintenance of the structure of OMEN itself. This includes managing users, managing music libraries, and maintaining and updating OMEN.

5.1 Researcher Interface

The primary purpose of the Researcher Interface is to make it as easy as possible for researchers to access cached metadata or request the extraction of new metadata. To accomplish this, the Researcher Interface provides access to the ability to search against the combined metadata from the various libraries. In addition, the Researcher Interface permits the user to either upload or generate settings for analysis, submit a request for metadata extraction, delete previously created data, view settings, file lists, or results already created, submit a new feature for inclusion in OMEN, and change the settings on their account. Researchers can create an account by registering using a link from the login page.

The search query interface consists of a simple keyword search against all metadata fields. However, the Researcher Interface also includes the ability to remove

entries from the list, select a random sample of the current file set, and to save the file list as a named query (see Figure 2).



Query Results

Selected	Name	Composer	Album	Type	Location	Comm
☑	Fallen	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	World On Fire	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Stupid	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Drifting	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Train Wreck	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Push	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Answer	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Time	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Perfect Girl	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Dirty Little Secret	Sarah McLachlan	Afterglow	Alternative & Punk	mcennis' iTunes	stereo mp3 160kbps
☑	Building A Mystery	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Hold On	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Good Enough	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	I Will Remember You	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Adia	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	I Love You	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Do What You Have To Do	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Path Of Thorns	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Fear	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Possession	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Sweet Surrender	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Ice Cream	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Fumbling Towards Ecstasy	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Angel	Sarah McLachlan	Mirrorball	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Drawn To The Rhythm	Sarah McLachlan	Solace	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Into The Fire	Sarah McLachlan	Solace	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	The Path Of Thorns (Terms)	Sarah McLachlan	Solace	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	I Will Not Forget You	Sarah McLachlan	Solace	Rock	mcennis' iTunes	stereo mp3 160kbps
☑	Lost	Sarah McLachlan	Solace	Rock	mcennis' iTunes	stereo mp3 160kbps

Refine Query | Remove Selected Items | Select All | Save File List | Get Random Subset | Cancel

Figure 2: Researcher Search Query Interface

Named queries are a mechanism to allow a researcher to save a list of files for later use in experiments. These file lists are listed by hyperlinked name in the researcher's homepage and can be later viewed by clicking on the name. By allowing researchers to save queries independently of the features to be extracted, researchers can reuse a common file list (i.e., for benchmarking purposes) even if researchers are not using the same features.

Also, there is a mechanism for storing settings for use in experiments. These settings can be loaded into OMEN in one of two ways. One can use the *Upload Settings*

interface to upload a jAudio settings file created using the jAudio Java application, or one can use the *Create Settings* interface on the homepage to explicitly create a settings file (see Figure 3). In both cases, the researcher supplies a name for the settings. The settings are then stored and displayed by name on the researcher's homepage. This feature allows researchers to store settings that are applicable for their research without tying those settings to a particular data set.

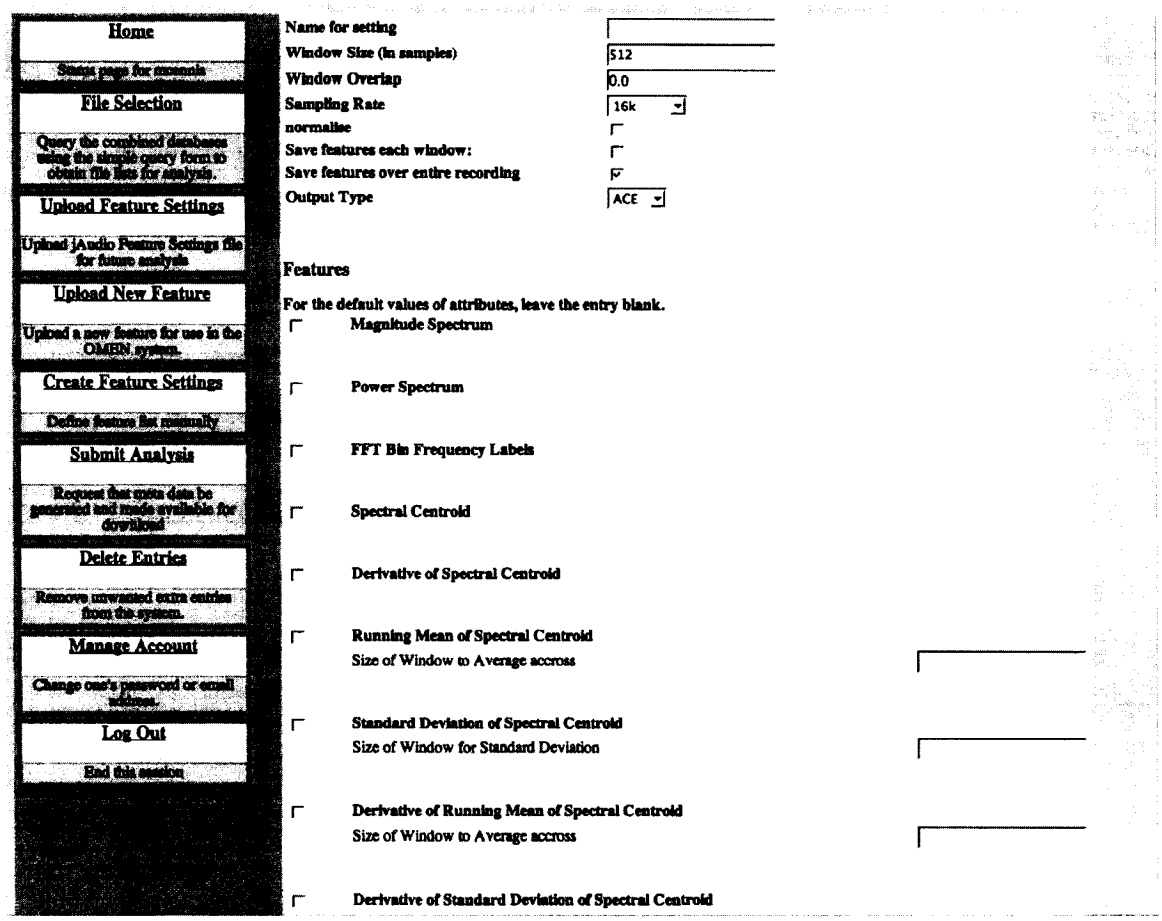


Figure 3: Researcher Create Settings

OMEN also provides an interface for submitting an experiment for execution. The first step is to choose which of the named settings and named queries to use for this experiment. The researcher also provides a name for the experiment and specifies whether or not the results should be made public. Once the experiment has been named,

the researcher is directed back to the researcher's homepage where the newly started experiment is now listed as an experiment in progress. By choosing to separate the definition of queries and settings from execution, researchers can reuse components, reducing the amount of information required to initiate an experiment. However, this makes initiating an experiment a three-step process.

Another aspect of the Researcher Interface is the *Delete Data* interface. In this window, researchers can select which of the queries, settings, and result sets are to be removed from the system. It should be noted that there are dependencies between the different components that prevent the deleted items from being permanently removed from the database. Queries and settings that are utilized by results cannot be fully deleted without deleting the results that reference them, even though they are removed from the users homepage. However, items that have been deleted but not removed will not count against the disk space usage of the researcher. Since metadata files can get quite large, a mechanism is needed to allow researchers to determine which data sets are most critical for their work. This will not necessarily eliminate the need for an administrator to delete files, but it will allow researchers a mechanism for avoiding potentially damaging intervention by an administrator.

Beyond deletion, a researcher can also view queries, settings, and results by following the hyperlinked names on the researcher's homepage (see Figure 4). Viewing queries shows the researcher the files in the query along with all the metadata of those files. Viewing settings shows the settings of the global options such as window size and window overlap and also shows all features that are active and the attribute settings for each of these features. Viewing results provides links for showing the query and settings

used to create the result set. In addition, OMEN provides links for downloading either the feature definition file and feature values file (if the output type is ACE) or a single ARFF file (if the output type is ARFF). One can also download the feature values file as a zipped file with one file per music file analyzed. This makes it easy for researchers to determine whether existing metadata or experiment components meet current needs that may eliminate the need for a request to occur at all.

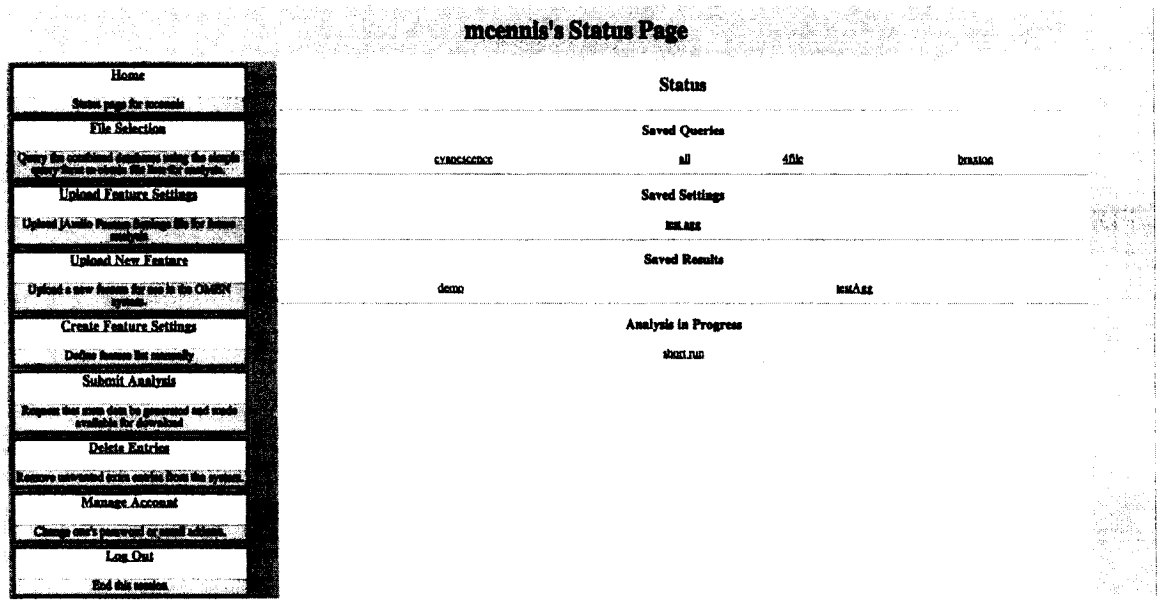


Figure 4: Researcher Status Page

The *Publish Feature* interface permits the researcher to make available for use a new feature not currently present in the analysis system. The feature implemented must follow either the FeatureExtractor interface (for general features) or the Aggregator interface (for aggregators) (see Chapter 4). The submitted Java file must also include Javadoc comments giving detailed explanation of the feature (including the purpose of the feature.) The Javadoc requirements are not enforced at submission, but changes must be approved by the administrator before they will be published for use. Providing a mechanism for adding new features to OMEN is critical. However, allowing researchers

to directly upload their code into the system is a serious security risk. One side effect of this system is that all researchers must provide source code for their submissions. While this will limit submissions, permitting secret feature extraction algorithms limits the ability of the MIR community to properly evaluate the uniqueness and correctness of these features.

Finally, a researcher can manage the details of their account. This includes changing their password, and changing their email address.

5.2 Librarian Interface

The Librarian Interface provides a web interface for a library to manage the files that it provides for analysis, for monitoring and managing the work queue, for managing usage policies, and for managing the library account with the master portal. This interface's primary design goal is to maximize librarians' ability to control the use of library resources while minimizing the expertise required to maintain the system. Librarians can be added or removed from the *Manage Accounts* section of the interface.

The interface for managing files permits the librarian to view, add, or remove metadata about files from the main portal. This interface is divided into three components. The first component is used for editing metadata and removing files. The second component is the upload interface. Here, librarians can upload an iTunes XML file containing the metadata of files to be added to the archive. The third component of the interface is a web entry form where librarians can enter new metadata manually. This portion of the interface allows librarians to control the metadata published about their resources. In order to maximize control, each entry can be edited manually. To make the interface accessible and usable for non-technical users, the iTunes XML file was chosen

since it is a well-documented format that can be easily created using readily available software without requiring any specialist knowledge.

Each library maintains exclusive control over the extent to which computers in the library participate in metadata extraction. The *Manage Workers* interface shows all the computers at the library that are currently configured for processing (see Figure 5). This list contains each computer's id, URL for processing, name, and the *reset worker* button. The *reset worker* button sends a message to the worker computer to cancel all current analysis and to clear its cache of all music files. Canceled analysis is added back to the queue to be re-deployed. This portion of the interface has a conflict between control and simplicity. Librarians must understand relatively complicated URLs describing the location of worker computers. However, this information is essential for controlling the local computing grid. Ideally, computers configured to participate in the local grid should broadcast their identity to the controlling Library Node. However, this functionality would complicate both the client and server beyond what is feasible for the prototype.

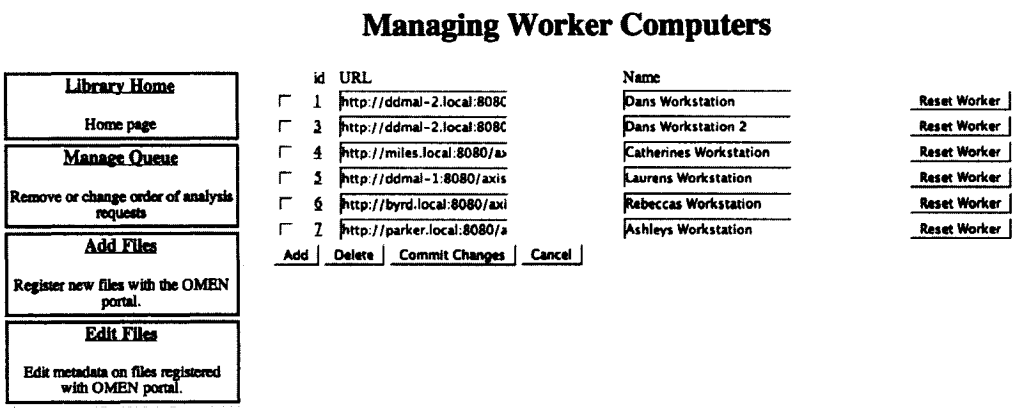


Figure 5: Managing Workers

By clicking on a worker in the list, the librarian can view the current settings for that particular worker. The features displayed in this component are the id of this worker and the values of each of the settings of this computer. The settings for workers are *Thread Priority, Max Cache Size*, and *On Idle Thread Priority* controls what priority should be used to execute analysis jobs (see Figure 6) *Max Cache Size* refers to the maximum amount of disk space that should be used to temporarily store the music files that are present in the request. *On Idle* designates whether or not this worker should work only when the computer is idle. This setting is not supported on all systems, so the receiving computer may silently ignore it. While these settings are required for complete control over the use of library resources, they require technical expertise to understand their purpose. Therefore, these settings are removed from the primary worker lists and placed in a separate settings interface that is accessible from the worker list for those librarians that have the expertise to understand the settings.

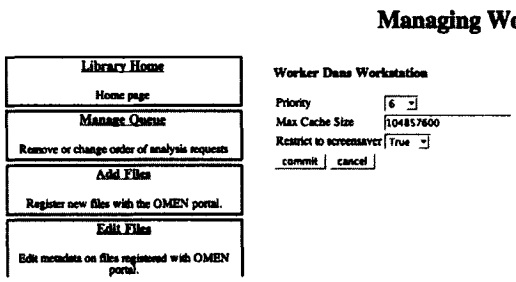


Figure 6: Worker Settings

In addition to configuring individual workers, the Library Interface provides an interface that allows librarians to globally set hours of operation for OMEN (i.e., when the library is closed) (see Figure 7). Any requests received after OMEN stops executing analysis are queued for execution when analysis resumes. While it may be useful to set different times of operation for different groups of computers, this would greatly

complicate the interface. A single, global control list provides sufficient functionality without placing undo burdens on the librarians.

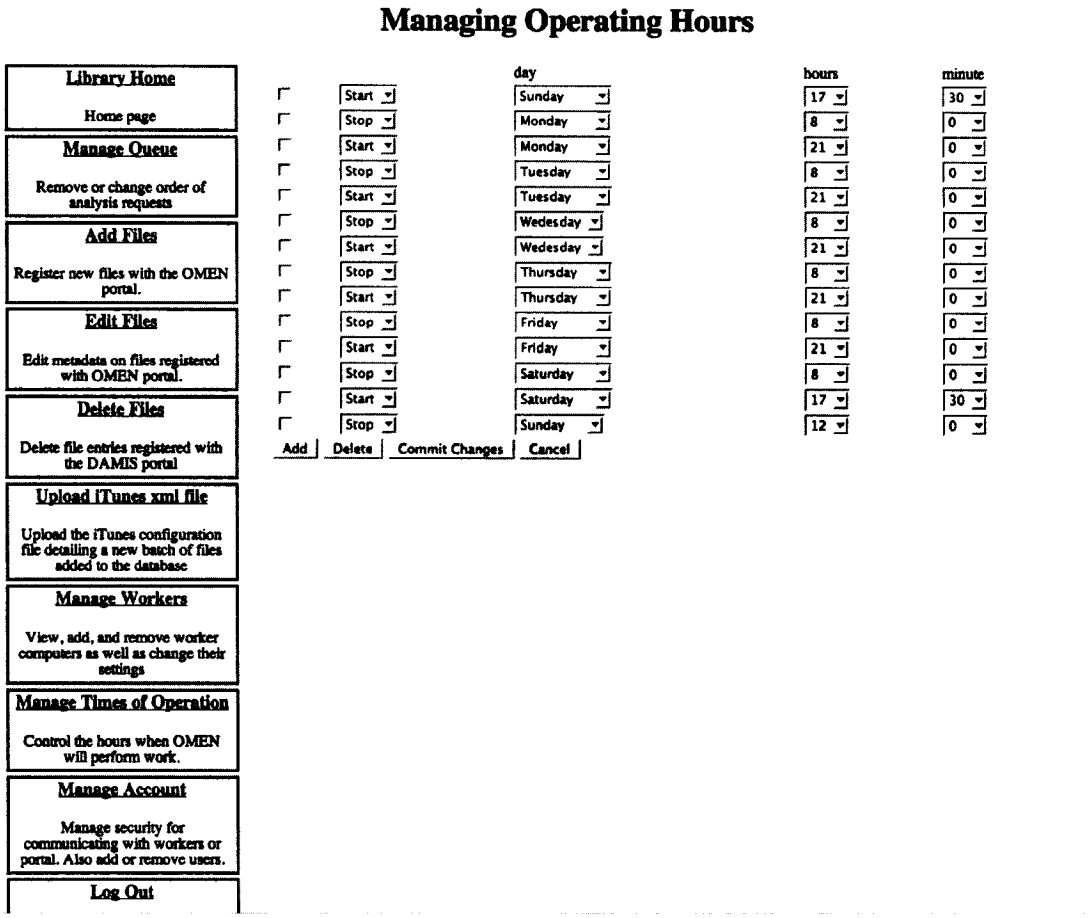


Figure 7: Operating Hours

The *Manage Queue* interface allows the librarian to alter the active queue of analysis requests that are currently pending as well as requests currently being processed (see Figure 8). In this interface, the librarian is able to cancel an analysis request and move a request up or down in the queue. Once a request has started executing, the request is now off the execution queue and can only be altered by canceling it. Since the queue is potentially changing extremely quickly, it is more important that librarians be able to request changes quickly as opposed to ease of use.

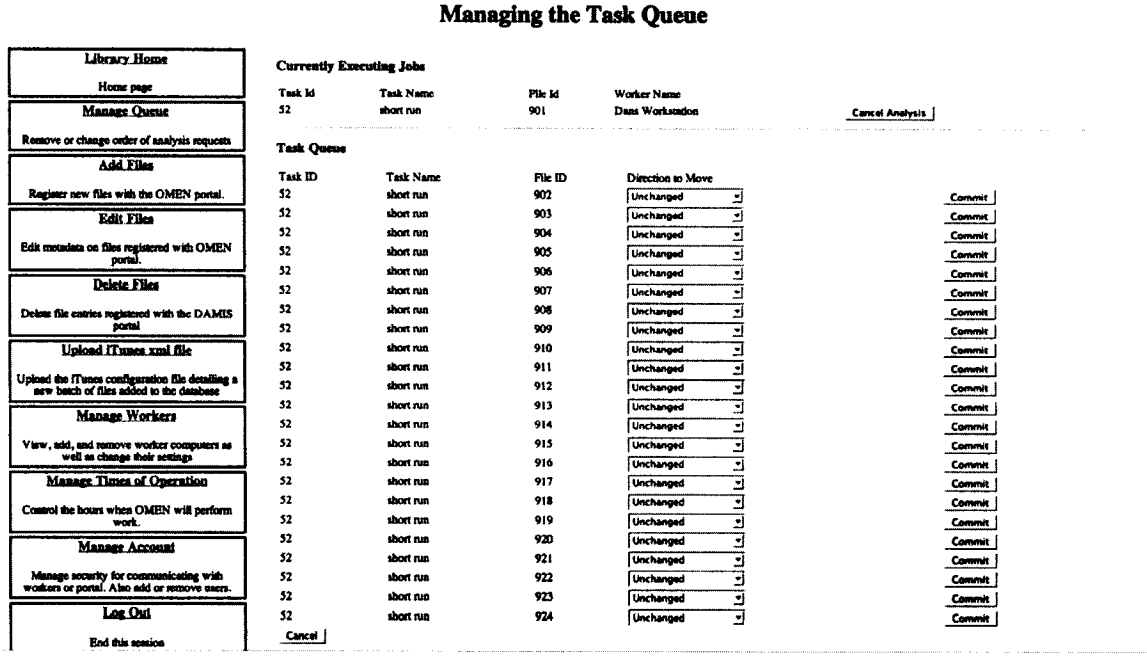


Figure 8: Managing the Queue

Finally, the Librarian Interface has a component for performing changes to the library's account. The librarian can change the list of librarians authorized to log in and reset the passwords of librarian accounts.

5.3 Administrator Interface

The administrator is responsible for maintaining the master site for both librarians and researchers. The Administrator Interface is based on the assumption that the administrator of OMEN is technically competent and wishes to complete tasks with the minimal number of steps. To accomplish this goal the administrator has five main components: *Manage Researchers* interface, *Add Libraries* interface, *Publish Features* interface, *Manage Public Data* interface, and *Manage Account* interface.

The *Manage Researcher* interface has components for viewing and editing researcher accounts (see Figure 9). The administrator can remove accounts, reset

passwords, and change email addresses. With the exception of removing accounts, the functionality of this interface is duplicated by the Researcher Interface *Manage Account*. This is important because some users will be unable or unwilling to manage their own account, making it imperative that the administrator also has access to this functionality. All accounts are available from the same page with all potential actions on an account listed on the same line as the account itself. This permits an administrator to find an edit an account in two steps—opening the account page and performing the needed edit.

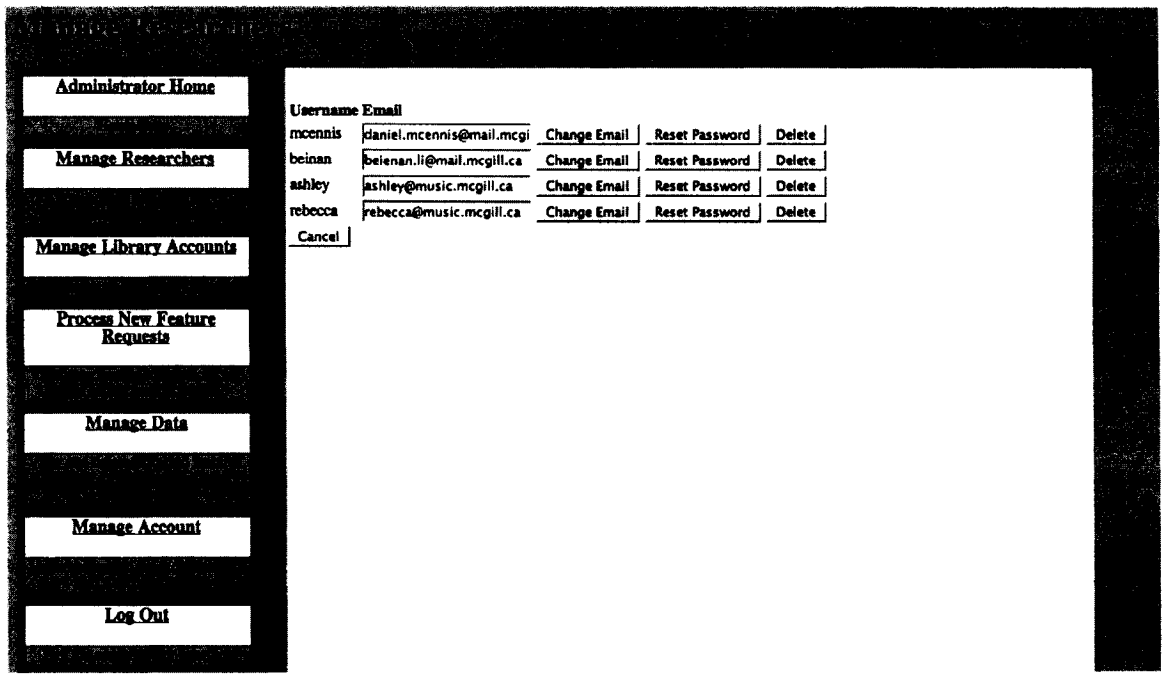


Figure 9: Managing Researchers

The *Add Libraries* interface allows the administrator to manage the libraries participating in OMEN. All existing libraries are listed in the interface along with the corresponding email addresses and the locations of their Library Node. The email address and locations can be altered or the libraries deleted entirely. Also, new libraries can be added to OMEN by supplying the name, email, and location of the Library Node (see Figure 10). Like with managing researchers, all tasks relating to managing libraries are

included on the same page, permitting administrators to add a library or perform an edit with a minimal number of steps.

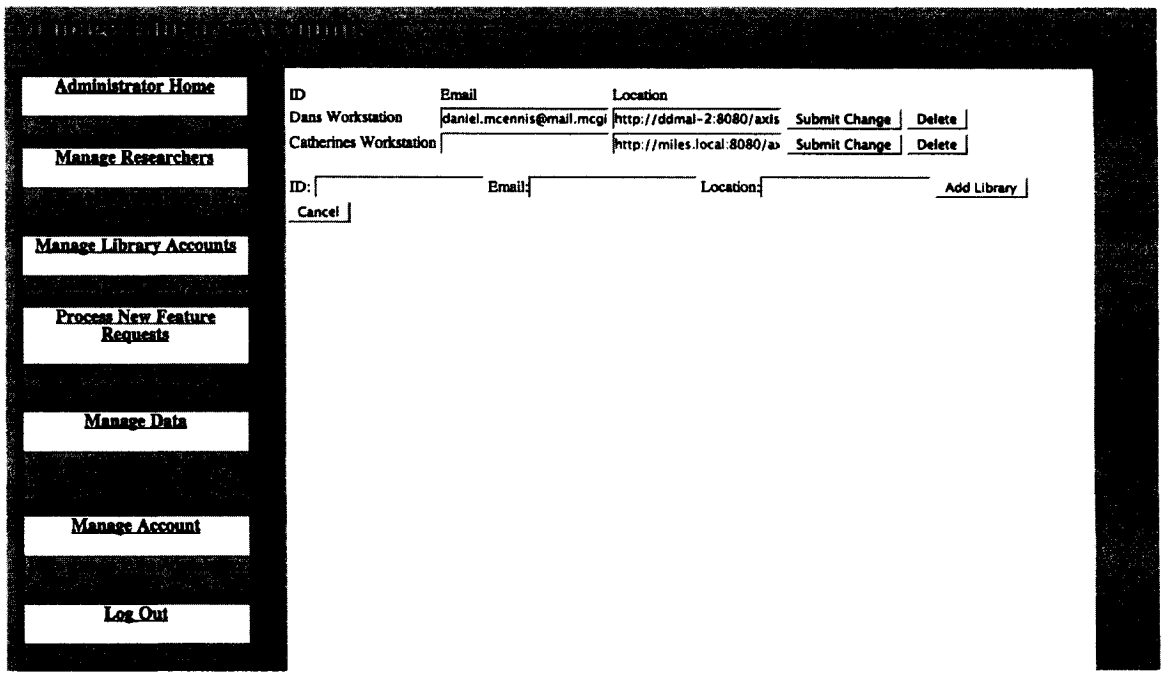


Figure 10: Managing Library Accounts

The *Publish Feature* interface shows all pending requests by researchers for adding features to OMEN (see Figure 11). Each request has buttons for *View Feature*, *Download Feature*, *Accept Feature*, and *Reject Feature*. The *View Feature* button shows the source code of the Java file. The *Download Feature* button permits the administrator to download the feature for testing against a local system. The *Accept Feature* button sends the administrator to a page to upload the compiled version of the feature. Then OMEN propagates the new feature by calling the NewFeature web service at all libraries. The *Reject Feature* button sends the administrator to a page where the administrator explains why the feature is rejected. This process has more steps than other tasks, however they are necessary to ensure that the system is not compromised by rogue code. Utilizing only the oversight of an administrator to safeguard the security of OMEN is

valid since many of the most difficult to detect attacks (i.e.,buffer overflows) are already caught by Java's built in error checking. The primary purpose of the check is to detect obvious attacks such as file or network access within a feature and to ensure that the features compile before distribution.

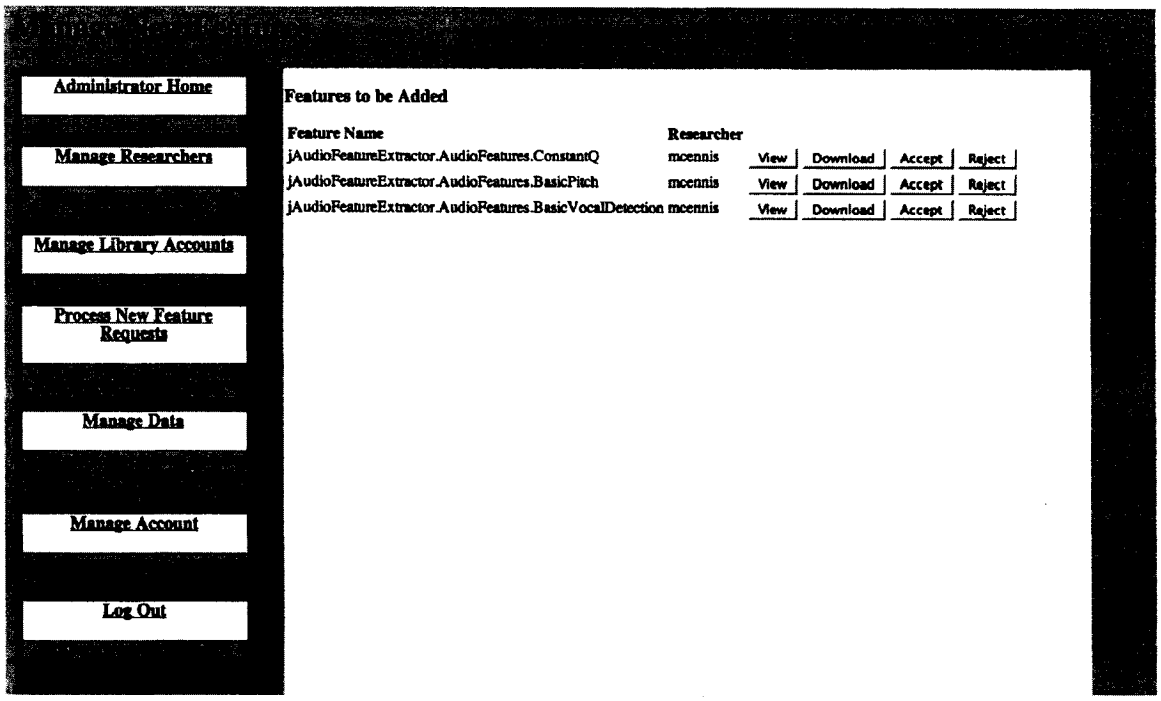


Figure 11: Managing New Features`

The *Manage Data* interface allows the administrator to eliminate cached data as needed. Data is grouped by type: file lists, settings, and results. Each entry is represented by its name. Unlike the Researcher Interface, data deleted is permanently deleted from the system. This permits administrators to cope with a potential lack of disk space when researchers have failed to police their own usage. All tasks and information related to managing data is placed within a single page to minimize the number of steps to accomplish the task.

The final component is the *Manage Account* interface. This interface permits the administrator to change the administrator password and the administrator's email address

as well as add new administrators and reset the passwords or change email addresses of other administrators.

6 Network Structure

In the previous chapter, OMEN was explained by describing its web interfaces. In this chapter OMEN is described by detailing each web service published by each class of computers (referred to as nodes) in the OMEN network. These services describe the communication between each of the different classes of computers for any given task. There are three types of nodes—Master Node, Library Node, and Worker Node. The Master Node is a central computer consisting of a single computer. The Library Node is the master computer for a given digital library. Worker Nodes are computers near a Library Node that perform on-demand metadata extraction on music files (see Figure 12).

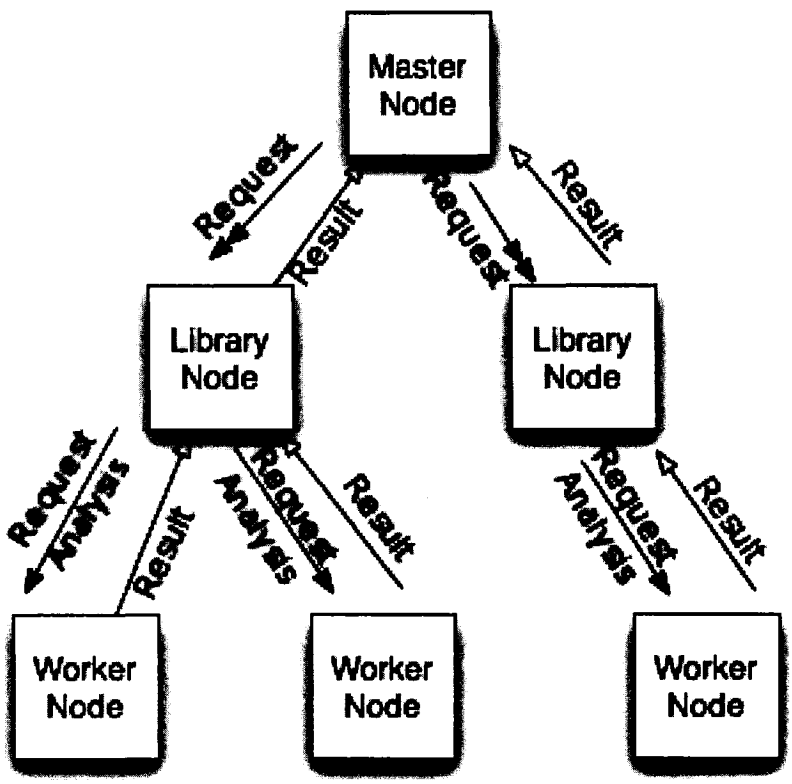


Figure 12: Network Topology

This structure was chosen for both its simplicity and for the control it provides. The distinction between Master Node and Library Node is needed to cope with copyright issues and provide control over participation to those institutions providing the computing and music resources. The split between Library Node and Worker Node is necessary to draw a distinction between the computer accessible to the rest of the network and those only accessible inside the library.

Inside a library, a server-client model is required in order to prevent those outside the library from knowing about the location of worker computers, interfering with the librarians' complete control over the use of library resources. While distribution of music files inside the network might be more efficient with P2P than client-server, a client-server model is much simpler and the impact of the bandwidth limitations of the Library Node is mitigated by caching.

A server-client model for communication between the Master Node and the Library Nodes has the advantage of simplicity and permits researchers to access the network using a web browser. A P2P solution could also work, but would require custom software to access and would require sophisticated consistency validation to ensure changes are fully propagated to all Library Node computers before they are applied.

6.1 Master Node

The Master Node is a monolithic central server that provides a central access point for accessing OMEN. It also provides a central point for monitoring analysis requests. This node hosts the researcher and administrator interfaces.

6.1.1 Web Services

The Master Node provides a substantial number of web services (see Glossary) utilized by Library Nodes. These are PublishResults, GetFileList, AddFiles, RemoveFiles, and NotifyAnalysisFailure.

The PublishResults service provides a mechanism for Library Nodes to report back to the Master Node the results of the analysis performed. This analysis can be in one of two forms: a single ARFF file or an ACE definitions file and an ACE values file. The PublishResult service receives the results from each Library Node and stores them in its database.

The GetFileList service provides a mechanism for Library Nodes to retrieve the file records that it has published to the Master Node. This is used to provide a means for a librarian to search its library's file list for errors to correct or files to remove.

The AddFiles service provides a mechanism for Library Nodes to add new file entries into the Master Node's database. Files are uploaded in a custom XML file passed by the Library Node.

The RemoveFile service provides a mechanism for allowing libraries to remove files from the master database. This service is also responsible for marking queries invalidated by the removal of files.

The NotifyAnalysisFailure permits a Library Node to notify the Master Node that analysis failed. A variety of different reasons can be encoded in the parameter including explicit termination by librarians, missing files, or errors caused by buggy feature implementations.

6.2 Library Node

Library Nodes are programs running on specific computers located at each participating library and are the master computers for their respective digital libraries. This node holds locally a copy of all files that are available for analysis. Furthermore, this node is responsible for handling all analysis requests, dispatching these requests to the Worker Nodes at the library. Furthermore, the Library Node is host to the library interface that provides access to settings specific to the library (see Chapter 6).

6.2.1 Web Services

The Library Node exports a number of services to create metadata or upgrade OMEN: ExecuteBatch, PublishResult, GetFile, AddFeature, and NotifyAnalysisFailure.

The ExecuteBatch service is called by the Master Node when a new analysis request is routed to the Library Node. This service is responsible for parsing this request into a number of requests for analysis of exactly one file. Each of these requests is then queued into a work queue. Finally, a scheduler runs to dispatch work to any idle workers in the network.

The PublishResult interface serves a similar purpose for the Library Node that the PublishResult service does for the Master Node. This service is responsible for receiving the results from each of the Worker Nodes. The results are then forwarded to the Master Node. The scheduler is then run to see if there is any work in the work queue for the now-idle worker.

The GetFile service is called by Worker Nodes when that node does not have the file locally that it needs to complete an analysis request. This service provides the file to the Worker Node.

The AddFeature service provides a mechanism for the Master Node to publish to Library Nodes new features for inclusion in the analysis subsystem. This service is responsible for replicating this call across all worker computers.

The NotifyFailure service provides a mechanism for Worker Nodes to notify the Library Node that it is unable to complete the analysis request that it was given. This service is responsible for reassigning the analysis request to another Worker Node by returning the request to the work queue. The type of failure is passed via an argument and includes not idle, missing file, invalid file, termination by Library Node, and problems from buggy feature implementations.

6.3 Worker Node

The Worker Node is the workhorse of OMEN. Worker Nodes are those computers already in place in the library for patron use that perform analysis. Since a Worker Node's primary purpose is providing search and web capabilities—not computation—a variety of techniques are needed to ensure that patron's use is not adversely affected. The techniques available in OMEN to accomplish this task can be activated or deactivated using the library web interface at the Library Node (see Chapter 5). One technique is that each Worker Node can be given hours of operation so that the computers are not executing analysis requests during hours that the library is open. Another technique available is to utilize 'on idle' computing which permits the application to execute if and only if the computer is currently running a screensaver. A third technique available is to set thread priority at the Worker Nodes. Setting thread priority informs the Worker Node how important the analysis task is. A low value will slow down analysis, but make it less likely that the analysis will interfere with patron use.

6.3.1 File Cache

Streaming music from the Library Node to the Worker Nodes is extremely bandwidth intensive—the time needed to transfer a file is comparable to the time needed to analyze it (Bray and Tzanetakis 2005). Fortunately, the same files will typically be analyzed repeatedly. By caching (keeping a local copy) of files that are analyzed, the Library Node can transfer the file to a Worker Node once, but analyze it many times, radically reducing bandwidth usage. While this uses significant disk space, the disk drives of the Worker Nodes are typically under-utilized.

6.3.2 Web Services

The Worker Node hosts a number of services that permit the Library Node to request analysis, publish settings and upgrades, and reset the worker. These services are: ExecuteBatch, AddFeature, Reset, ApplySettings, and CancelAnalysis.

The ExecuteBatch service allows the Library Node to communicate to the Worker Node the analysis it is to perform. This includes a reference to the file to perform the analysis on and the settings that the analysis should use. If the file is not present within the file cache, the Worker Node requests this file from its Library Node. Once the analysis has been performed, the results of the analysis are published to the Library Node using its PublishResults web service. If an error is encountered during execution, the Worker Node calls the NotifyFailure service instead.

The AddFeature service provides a mechanism for the Library Node to deliver new code for features to each of its workers. This service is responsible for placing the class file attachment into its class path and updating the file listing available features with the name of the new class.

The Reset service tells the Worker Node to erase its file cache and cancel any analysis requests it has received. This service is useful to deal with corrupted caches and as a last resort for terminating executing processes. This command will terminate any analysis currently running and call NotifyFailure at the Library Node with an argument that indicates that the analysis was canceled at the Library Node's request.

The ApplySettings service permits the Library Node to change how the Worker Node carries out its request. This includes whether the Worker Node should execute on idle only and the priority setting of the analysis threads.

The CancelAnalysis service provides a mechanism for the Library Node to cancel any analysis conducted on the Worker Node. When this message is received, the Worker Node cancels any running analysis and calls the Library Node's NotifyFailure with an argument that indicates that the analysis was canceled at the Library Node's request.

7 Legal Issues

One of the primary reasons that the OMEN project is needed is to cope with legal restrictions. These legal restrictions are part of a larger debate on the cultural issue of compensation for access to and use of music. The issue is further complicated by the vast disparities in copyright laws between countries and the indeterminate state of the laws within these countries. This discussion of legality is not a legal opinion and is not guaranteed to be accurate representations of legal responsibilities and/or rights.

7.1 Canadian Copyright Law

The copyright laws in Canada are remarkably flexible for the use of copyrighted works for the purpose of research. According to the Copyright Act "Fair dealing for the purpose of research or private study does not infringe copyright." (Copyright Board of Canada 2006). This would appear to permit any analysis system if the analysis system is used exclusively for the purpose of research.

7.2 USA's Copyright Law

Unfortunately, copyright law in other countries is not nearly so forgiving. Copyright in the USA is governed by Title 17 of the United States Code and is an extremely complicated 290-page document. While research is entitled to 'fair use' exemptions, the law is extremely vague as to what activities are considered acceptable research. In particular, one of the considerations when deciding fair use is "the effect of the use upon the potential market for or value of the copyrighted work." (Copyright Law of the United States 2003). By never copying the music outside the institution that purchased it, OMEN seeks to side step many of the issues of copyright under USA law.

While the legality of transferring music to different devices internally has never been tested in court, the consortium representing nearly all commercial music in the USA, the Recording Industry Association of America (RIAA), has stated before the Supreme Court in USA that "The record companies, my clients, have said, for some time now, and it's been on their Website for some time now, that it's perfectly lawful to take a CD that you've purchased, upload it onto your computer, put it onto your iPod. There is a very, very significant lawful commercial use for that device, going forward." This testimony would seem to provide participating institutions protection from litigation from the RIAA for uses internal to the organization.

7.3 Conclusion

Additional research is needed to determine the legal status of OMEN, particularly outside Canada and USA, before it could be adopted by the Music Information Retrieval community. However, given that OMEN is designed to accommodate relatively strict restrictions in the USA, it is probable that OMEN will pass legal muster elsewhere.

8 Conclusions and Future Work

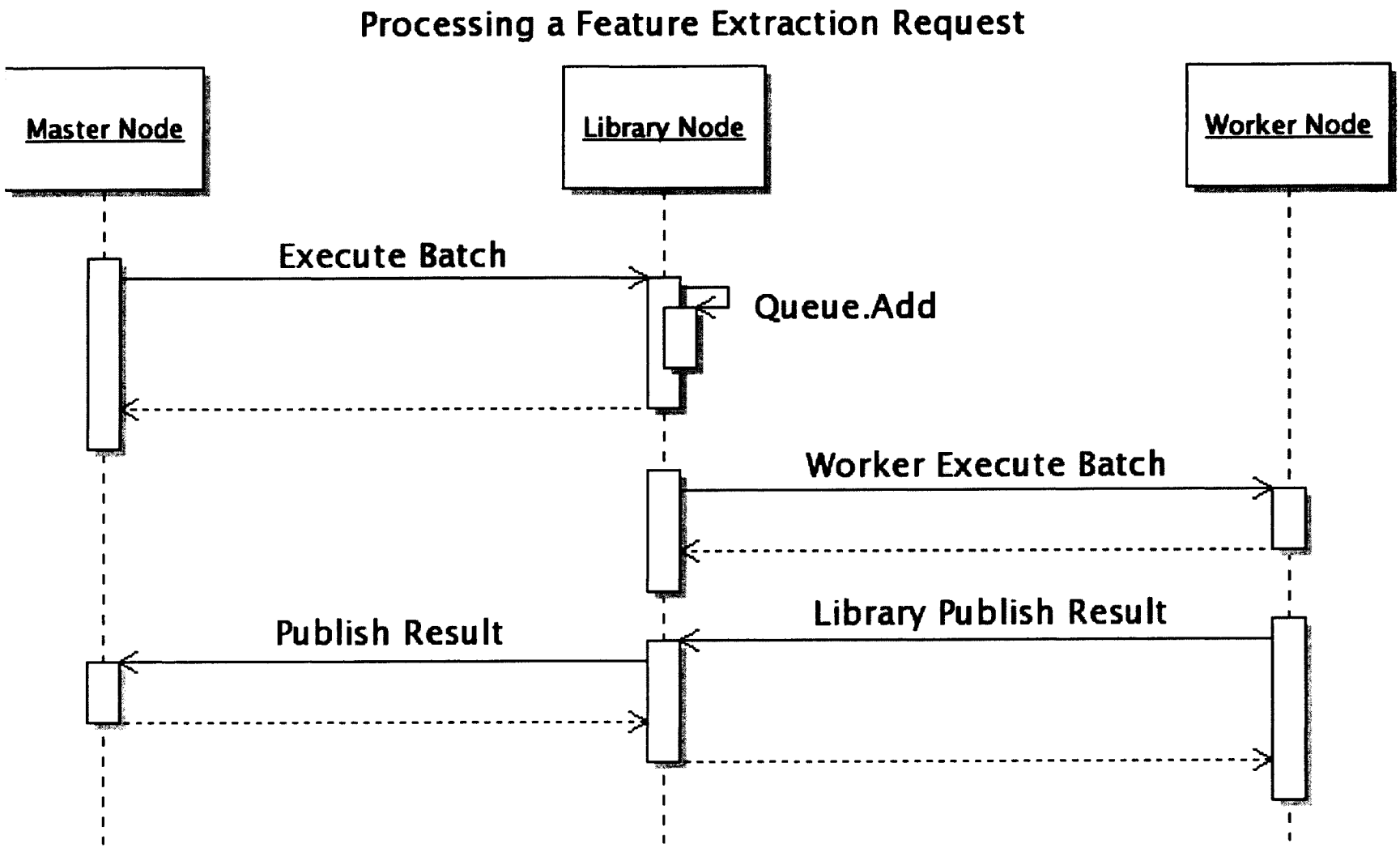
OMEN provides a mechanism for legally providing all Music Information Retrieval (MIR) researchers with access to a large dataset of music. This will permit researchers to compare results of MIR algorithms using the same music at any time, not just during specific competitions. It will also save researchers significant amounts of time by removing the need to construct their own datasets or resort to non-representative music in the public domain.

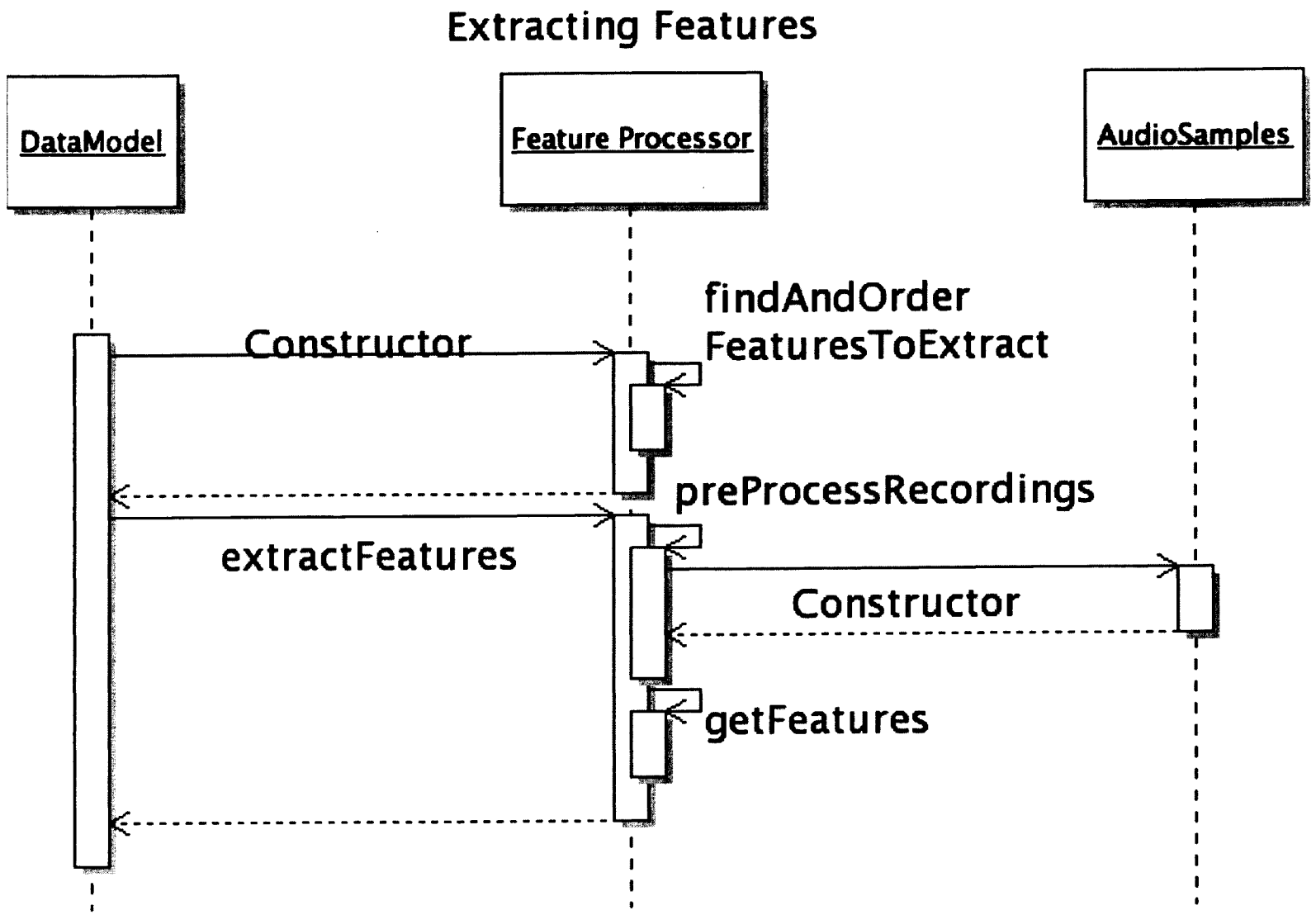
However, there is still work to be done. In particular, the system requires additional security to prevent unauthorized access of OMEN resources. This includes functionality such as digital signatures for authenticating communications between nodes.

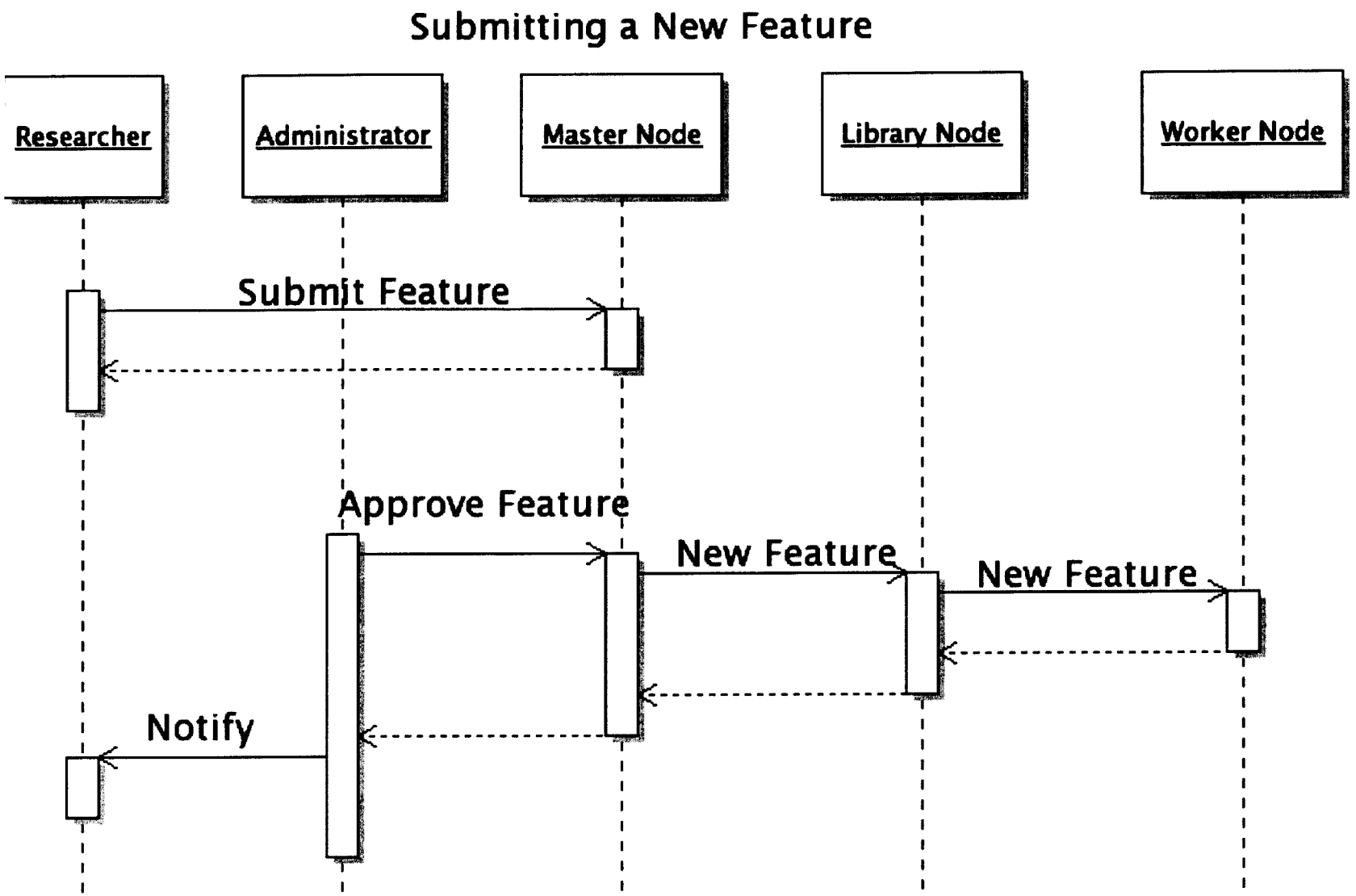
Another area for improvement is the scheduling algorithm for the Library Nodes. The required bandwidth could be greatly reduced if more care were taken to distribute feature extraction requests to Worker Nodes that already have needed files in their caches.

In addition, a mechanism is needed to identify duplicate files across libraries and more sophisticated mechanisms are needed for creating file lists and editing file metadata. Finally, OMEN should, in the future, provide a way to present to MIR researchers only those files that have ground truth (see Glossary) for particular MIR tasks.

Appendix A – Use Case Diagrams


Processing a Feature Extraction Request
Master Node
Library Node
Worker Node
Execute Batch
Queue.Add
Worker Execute Batch
Library Publish Result
Publish Result


Extracting Features
DataModel
Feature Processor
AudioSamples
findAndOrder
FeaturesToExtract
Constructor
preProcessRecordings
extractFeatures
Constructor
getFeatures


Submitting a New Feature
Researcher
Administrator
Master Node
Library Node
Worker Node
Submit Feature
Approve Feature
New Feature
New Feature
Notify

Appendix B – jAudio Design

It is frequently useful to know the underlying algorithms used to accomplish particular tasks. To this end, the overall structure, the key class involved in feature extraction, and the feature extraction algorithms are explained.

B.1Global Structure

jAudio was designed to using the Model View Controller design pattern (Buschmann et al. 2001). In this pattern, a program is divided into three parts—models, views, and controllers. The model part is where the data of the program is stored, views are the representations of the data that users see, and controllers are the mechanism by which a model is changed. In jAudio, the DataModel class comprises the model; the GUI classes comprise one view, and the jAudioFE class comprises the command-line view; and the Controller class is the controller part. When using jAudio as a library, OMEN implements its own views and controllers and uses the DataModel class as a model.

B.2 DataModel Class

The DataModel class provides the model of the model view controller. It stores the information needed to perform analysis including information about features and information about analysis settings.

The building of a DataModel object involves several parameters. The first is the callback stub for a view (called a ModelListener) that provides mechanisms for a DataModel object to report progress during analysis. The second is the location of the XML configuration file used to define what features and aggregators are available for use

by jAudio. At the start of the construction, DataModel constructs an array of available metafeature factories. These factories are hard coded into the constructor. Next, the DataModel object parses the XML file. The location of the plugin directory (the location where jAudio looks to find compiled Java class files implementing features and aggregators) is extracted from the configuration XML file. Each feature and aggregator in the configuration file is then dynamically loaded. For every combination of feature extracted from the configuration XML file and metafeature factory defined, a new feature is created. These new features are deactivated by default, but are added to the list of available features, completing construction of the DataModel object.

DataModel contains a number of variables that control how metadata extraction takes place. The first two variables are an array of features and an array of Boolean variable describing whether or not the corresponding feature is currently enabled. A variable of type ModelListener is provided to generate progress reports on progress of the analysis. There is also a variable that holds references to all files to be analyzed. There are also two variables for controlling the output of the jAudio. These two variables permit the output to be redirected to destinations other than a file (like a network connection).

B.3 Execution of Metadata Extraction

The extraction process can be divided into three phases—construction, dependency resolution, and extraction. The construction phase loads the parameters involved in carrying out metadata extraction (provided either by the GUI or by an XML file provided by the user). The dependency resolution phase establishes which features need to be extracted and in what order. Finally, the extraction phase is where the

individual audio files (chosen via the GUI, listed on the command-line, or provided in an XML file provided by the user) are analyzed.

B.3.1 Construction

The extraction process is initiated by calling the extract method of a DataModel object. It has a number of parameters used to control the execution of the extraction. The first two variables deal with the windowing of the input signal. The first variable gives how long the window should be in samples. The second variable determines the window overlap, which indicates what percent of the pervious window should be duplicated in the next window (ranging from 0 to 1). The next variable is the target sampling rate of the analysis in Hertz. (If audio files have different sampling rates, they will be automatically re-sampled to the target rate). The next variable is a Boolean value indicating whether or not the files analyzed should be normalized before extraction. The next two variables indicate whether metadata should be generated for every window, once for each file, or both. The following two variables give the names of the files where output should be stored. For ACE output, these two variables correspond to the feature definitions file (listing the features that have been extracted) and the feature values file (listing the values extracted for each file). For ARFF output, the first variable is ignored and output is sent solely to the second file. If these values are null, the internal variables for controlling output are used. Next is an array of the files to be analyzed. Finally, there is a variable indicating what format the output should be in—either ACE's data format of Weka's ARFF format.

B.3.2 Dependency Resolution

Initially, a helper object of type FeatureProcessor is created. This object's constructors has as parameters all of the variables in the DataModel and all the parameters to the execute method. FeatureProcessor performs dependency resolution for all features that have been chosen for extraction. This process has two distinct steps. The first to identify which features must be extracted (but not saved) in order to satisfy the dependencies of features that are to be extracted. The second step involves determining what order these features should be calculated in order that all features are extracted only after all the features they depend on have also been extracted.

jAudio begins automatic dependency resolution by making a list of all features whose output is to be saved (A). Another list (B) will be built that will eventually contain a list of all features to be extracted. These two lists can differ, as a user may not wish to save a given feature, but this feature may nonetheless need to be extracted in order to calculate another feature that does in fact need to be saved.

Initially B is set to be identical to A. jAudio then loops through each feature in B and adds each feature's dependencies to B if they are not already in B. The loop terminates when jAudio completes an iteration without adding any new features.

Once the features to be extracted have been identified, jAudio orders these features to ensure that every feature is calculated only after its dependencies have been calculated. To accomplish this, jAudio creates an ordered list of features to extract (C). It then cycles through all the features in B. If all dependencies of a given feature are in C, then it is added to C as well. This loop terminates once all features in B have been added to C.

B.3.3 Extraction

Once the FeatureProcessor object has been built and the features to extract have been determined and ordered, the executeFeatures method of FeatureProcessor is called once for each file to be analyzed. For each file, the audio data is transformed into samples via the preProcessRecordings method. Then the samples are broken up into windows of data, the features are extracted sequentially for each window, and summary statistics are calculated, if requested. Finally, output containing the extracted metadata is generated.

jAudio initially opens the given music file for preprocessing. This file is loaded into memory then, if needed, transformed into pulse code modulation (PCM) format (i.e., sample-based representation), and resampled to the sampling rate chosen by the user. If requested, the signal is normalized at this point. Finally, the signal is mixed down to one channel. The result is a description of the signal that is an array of double values between -1 and 1.

The array of samples is then broken down into windows of the appropriate size and overlap. If there is insufficient data to fill the last window, it is zero padded to full length. Each feature is extracted for each window by calling its extractFeature method using the dependencies calculated in the FeatureProcessor constructor. If it is not possible to calculate the output for a feature (due to, for example, not enough windows calculated yet to satisfy dependencies), then no output is produced for that feature for that window.

Once all the features have been calculated, the results are output in the appropriate output format (either Weka's ARFF or ACE's data format). No output is generated until all calculations have been completed.

Appendix C – Glossary

Axis

Axis is a dynamic-linked library that supports the deployment of web services written in the Java programming language. While it can be run independently, it is typically used within a servlet container program such as Tomcat. It is an open-source program published by the Apache foundation (Web Services 2006).

Cache

A cache is a temporary local copy of something. In the context of networking, a cache refers to a copy made of something transmitted over the network that is temporarily stored at the computer that receives it so, if it is needed again, the local copy can be used instead of requesting the original again.

Dynamic-linked Library

A dynamic-linked library is a collection of sub-programs that are designed to be used by other programs. When an outside program requests a sub-program in the dynamic-linked library, the library is read from the disk and the sub-program executed.

Ground Truth

A collection of metadata that describes what the 'correct' analysis is for a particular task. This is useful to test how well algorithms perform a given task.

Servlets and JavaServer Pages

Servlets and JavaServer pages are technologies that permit developers to write dynamic web pages using the Java programming language. In order to produce web using these

technologies, a special web server known as a servlet container is needed. Sometimes this container is integrated into a web server, but it is often a stand-alone program.

Tomcat

Tomcat is a servlet container that converts servlets and JavaServer pages into web pages. It is an open-source program published by the Apache organization (Tomcat 2006).

Web Services

Web services are web pages specifically designed to be accessed not by humans, but other computers. When a computer requests a web service, it sends as part of its request an XML file providing information to the computer supplying the web service much like a web browser submits information to a web server when submitting a form. The computer supplying the web service performs a task and then sends an XML file back to the computer that initiated the request. The primary benefit of using web services is that the communication is independent of any programming language or operating system, much like a web page can be viewed on different web browsers or different operating systems.

XML

XML is a technology for describing data in such a way that humans can also read the data and understand what it is representing and is easy for computers to read as well. Pieces of information are surrounded with tags that describe what the data inside represents. The format of these tags are fixed and the rules for nesting tags inside tags follow a strict syntax to make it easy for computers to use, but the names of the tags are chosen so as to

make it as clear as possible for humans to understand what the data inside the tags represents.

References

Amatrain, X., and Arumi, P. 2002. Clam: Yet another library for audio and music processing? In *Proceedings of Annual ACM Conference on Object Oriented Programming, Systems, and Applications.*

Anderson, D. 2004. BOINC: A system for public-resource computing and storage. In *Proceedings of IEEE/ACM International Workshop on Grid Computing.*

Apache Tomcat. 2006. [cited April 14 2006]. Available from http://tomcat.apache.org

Bainbridge, D., Don, K., Buchanan, G., Witten, I., Jones, S., Jones, M., and Barr, M. 2004. Dynamic digital library construction and configuration. In *Proceedings of European Conference on Digital Libraries.*

Berthold, H. 2002. *A federated multimedia database system.* Ph.D. Dissertation. Technischen Universität Dresden.

Bolger, B., Healy, M., and Tukey, J. 1963. The quefrency analysis of time series for echoes: Cepstrum, psuedoautocovariance, cross-cepstrum, and saphe-cracking. In *Proceedings of Symposium Time Series Analysis.*

Book Collector. 2006. [cited May 20 2006]. Available from http://www.portable-software.com/download/bc/Book%20Collection.htm.

Bray, S., and Tzanetakis, G. 2005. Distributed audio feature extraction for music. In *Proceedings of International Conference on Music Information Retrieval,* 434–7.

Bretschneider, T., and Kao, O. 2002. A retrieval system for remotely sensed imagery. In *Proceedings of Conference on Imaging Science, Systems, and Technology.*

Clough, P, Müller, H., and Sanderson, M. 2004. The CLEF 2004 cross-language image retrieval track. In *Proceedings of Cross-Language Evaluation Forum.*

Copyright Board of Canada: Copyright Act. 2005. [cited September 26, 2006]. Available from http://www.cb-cda.gc.ca/info/act-e.html.

Copyright Law of the United States. 2003. [cited September 26, 2006]. Available from http://www.copyright.gov/title17/preface.pdf.

Curzio, C., Lawson, J., Froment-Curtil, J., and Vender, J. 2006. *distributed.net.* [cited February 14 2006]. Available from http://distributed.net.

de Mallorca, P. 2004. *MPEG-7 overview.* [cited February 17 2006]. Available from http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm.

Downie, S., Ehmann, A., and Tcheng, D. 2005a. Music-to-knowledge (M2K): A prototyping and evaluation environment for music information retrieval research. In *Proceedings of Special Interest Group on Information Retrieval.*

Downie, S., Futrelle, J., and Tcheng, D. 2004. The international music information retrieval systems evaluation laboratory. In *Proceedings of International Conference on Music Information Retrieval.*

Downie, S., West, K., Ehmann, A., Vincent, E., 2005b. The 2005 music information retrieval evaluation exchange (MIREX 2005): Preliminary overview. In *Proceedings of International Conference on Music Information Retrieval,* 300–3.

Foster, I., Kesselman, C., Nick, J., and Tuecke, S. 2002. Grid services for distributed system integration. *IEEE Computer* 35 (6): 37–46.

Fujinaga, I. 1997. *Adaptive optical music recognition.* Ph.D. Dissertation. McGill University

Gunther, N, and Beretta, G. 2001. A benchmark for image information retrieval using distributed systems over the Internet: BIRDS-1. In *Proceedings of Internet Imaging.*

Gurman, J., Dimitoglou, G., Hourclé, J., Bogart, R., Tian, K., Hill, F., Suárez-Sola, I., Wampler, S., Martens, P., Yoshimura, K., and Davey, A. 2005. The virtual observatory: Still in a small box. In *Proceedings of Joint Assembly.*

Harte, M.-J., and Hoogerdijk, A. 2006. *Collectorz.* [cited May 20, 2006]. Available from http://www.colectorz.com.

Hulbert, N., Freeland, S., Shine, R., Bose, P., and Woodward, M. 2001. A prototype problem solving environment for living with a star data. In *Proceedings of Joint Assembly.*

Internet Archive. 2006. [cited February 12 2006]. Available from http://www.archive.org.

Jackson, L. 1999. *Digital Filters and Signal Processing.* 3rd ed. Norwell, MA: Kluwer Academic Publishers.

Klapuri, A. 1999. Sound onset detection by applying psychoacoustic knowledge. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing.*

Lawton, G. 2000. Distributed net applications create virtual supercomputers. *Computer* 33 (6): 16–20.

Li, X., Kim, Y., Govidan, R., and Hong, W. 2003. Multi-dimensional range queries in sensor networks. In *Proceedings of International Conference on Embedded Networked Sensor Systems*.

McAdams, S. 1999. Perspectives on the contribution of timbre to musical structure. *Computer Music Journal* 23: 85–102.

McClellan, J., Schafer, R., and Yoder, M. 1999. *DSP First: A Multimedia Approach.* Upper Saddle, New Jersey: Prentice Hall.

McEnnis, D., McKay, C., Fujinaga, I., and Depalle, P. 2005. jAudio: A feature extraction library. In *Proceedings of International Conference on Music Information Retrieval,* 600–3.

McKay, C., McEnnis, D., Fiebrink, R., and Fujinaga, I. 2005. ACE: A general purpose classification ensemble optimization framework. In *Proceedings of International Conference of Music Information Retrieval,* 161–4.

Messorotti, M., Coretti, I., Padovan, S., Zlobec, P., Antonucci, E., Cora, A., Volpicelli, A., Dimitoglou, G., Reardon, K., Tripicchio, A., and Severino, G. 2003. The Italian solar data archives: National and European perspectives. *Memorie della Societa Astronimica Italiana* 74: 391.

Moreira, J., Midkiff, S., Gupta, M., Artigas, P., Snir, M., and Lawrence, R. 2000. Java programming for high-performance numerical computing. IBM Systems Journal, 39(1): 18–24.

Ogle, M. 2005. *Streetprint.org.* University of Alberta, Faculty of Arts, Canada Research Chair Humanities Computing Studio. [cited October 2 2005]. Available from http://www.crcstudio.arts.ualberta.ca/streetprint.

Over, P., Kraaij, W., and Smeaton, A. 2005. TRECVID 2005: An introduction. In *Proceedings of TREC Video Retrieval Evaluation.*

Petkovic, M., and Jonker, W. 2004. Integrated use of different content derivation techniques within a multimedia database system. *Journal of Visual Communication and Image Representation* 15 (3): 303–29.

Quackenbush, S., and Lindsay, A. 2001. Overview of MPEG-7 audio. *IEEE Transactions on Circuits and Systems for Video Technology* 11 (6): 725–9.

Sikora, T. 2004. *MPEG-7 audio analyzer: Low level descriptors (LLD) extractor.* [cited January 27 2006]. Available from http://mpeg-7lld.nue.tu-berlin.de/.

Smeaton, A., and Over, P. 2002. The TREC-2002 video track report. In *Proceedings of Text Retrieval Conference.*

Spalding, T. 2006. *Library thing*. [cited May 20, 2006]. Available from http://librarything.com

Tritonus: Open source java sound. 2005. [cited May 12 2006]. Available from http://tritomus.org

Tzanetakis, G., and Cook, P. 2000. MARSYAS: A framework for audio analysis. *Organized Sound* 4 (3): 169–75.

Web Services: Axis. 2006. [cited April 14 2006]. Available from http://ws.apache.org/axis

Witten, I., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. 2nd ed. San Francisco: Morgan Kauffman.

Witten, I., McNab, R., Jones, S., Cunningham, S., and Bainbridge, D. 2000. Greenstone: A comprehensive open-source digital library software system. In *Proceedings of ACM Digital Libraries*.