Integrating multiple distributed data sources to achieve more relevant music search results

Damion Mitchell
University of Illinois

Huong Luu
University of Illinois

Joana Trindade
University of Illinois

1. Introduction

Currently, there is a proliferation of musical data available over the Web, and according to [1], "music" is the 4th most searched keyword in the internet. In addition, there are approximately 4,558,206 searches for "music" performed on a daily basis. Music is one of the most popular types of online information and there are now hundreds of music streaming and download services operating on the Web. Some of the music collections available are approaching the scale of ten million tracks and this has posed a major challenge for searching, retrieving, and organizing music content. One of the very important challenges is how to search these vast, global-scale musical resources to find relevant music information.

By viewing the Web as a global-scale musical resource, there are many issues and limitations to be addressed. By simply searching on meta-data (song title or artist names), problems relating to ambiguity in search queries oftentimes will arise. Examples range from numbers (*e.g.*, the hit "1979" from "Smashing Pumpkins") to verbs (*e.g.*, "Smile" by "Lily Allen"), and more ambiguous titles, such as "Music" from "Madonna". Also, existing techniques for enhanced search, such as entity annotation, do not apply directly to the music domain. One reason is that it is difficult to devise regular expressions and annotations that describe what music elements, such as a song title and artist or group name, should look like [2].

To address some of these issues, iMusic allows for the integration of data to produce efficient music queries (*i.e.*, album, artist, and songs) using multiple distributed data sources. This approach is based on the premise that we can enhance the quality of search results by using a number of available music resources to respond specifically to a user query.

1.1 Initial Goals of System

The main goal of the system was to serve as a component of iWisdm which would be able to handle "music" queries for a combination of keywords, and return a comprehensive corpus of information applicable to the query in question. There were several proposed goals for the iMusic system, which would assist in the realization of the integration of multiple distributed data sources to achieve more relevant music searches. Table 1 summarizes the initial goals of the system, as it pertains to album, track, and artistes search, with its proposed inputs and outputs.

Input(s)	**Output(s)**	
1. *Album* e.g. Erotica	a) Track list	b) Album art
	c) Pricing information (per album)	d) *Listener ratings and reviews
	e) Available free streaming	f) Similar albums recommendations
	g) Music video of album's hit song	
2. *Track* e.g. The Prayer	a) Lyric	b) Similar songs recommendations
	c) Pricing information (per song)	d) Available free streaming

3. *Artist* e.g. James Ingram	a) Biography	b) Discography
	c) Photos (from album art, or live concerts)	d) Similar artists recommendations
	e) *Upcoming events	f) *Collaborations
** optional features*		

Table 1-Summary of Initial Goals of System

The structure of this paper is as follows: Section 2 presents an overview of our initial prototype in Phase Zero, and Section 3 discusses lessons learned during this Phase. Section 4 discusses related work in query systems for music information retrieval. Section 5 presents iMusic at Phase One and our intended vision of the system. Finally, Section 6 concludes this report by giving a final summary of iMusic.

2. Phase Zero: An Initial Prototype

In Phase Zero, an initial prototype of iMusic was created, which was an end-to-end system that uses the integration of data to produce efficient music queries, via resources from several web sources. This phase allowed users to perform different music queries ranging from artist, album, or track searches. For artist queries, iMusic presented information about the artist's biography, discography, similar artists' suggestions, videos for hit songs, events, and news related to the artist. The prototype also provided album art, summary information, track list, and videos for album related queries; while for track queries, the lyrics and a video of the song were presented.

In this Section, we describe a high level overview of iMusic's architecture, as well as more details about its implementation.

2.1 System Description

It was paramount that the iMusic system not only enhance the music search experience of the user, but also present information in an integrated manner from several web sources. There was a need to present the users with more than just a track list, or video, but with a host of relevant information including lyrics, similar artists, and discography. eThe system therefore handles different music related queries (album, track, and artist), from which calls are made to several APIs and results returned ranging from albums, artistes, and tracks.

iMusic employs several APIs *(see table 2)* along with custom built queries to facilitate the integration of data. Since web pages are built using text-based mark-up languages (HTML and XHTML), and frequently contain a wealth of useful data in text form; performing data scraping is generally considered an ad-hoc, inelegant technique, often used only as a "last resort" when no other mechanism is available. There are instances when the requests made via an API is impossible or the API being used provides inadequate information, the system automatically uses custom built queries to scrape information directly from the source website, Table 3.

API	**Purpose**
Youtube	Return videos for selected album or track or artist hit songs
Last.fm	To get artist biography, similar artists, albums, album art, summary information
LyricsWikia	Return a url to lyrics of a song, given a song title and an artist name
Eventful	To get the upcoming events for a particular artist
Yahoo! Boss	To retrieve the latest news on an artist
EchoNest	To retrieve the official URLs and news for an artist.

Musicbrainz	To get discography information for an artist

Table 2 – APIs used and their purpose

Website	Purpose
LyricsWikia	To get content of lyrics of a song
Gracenote	To get biography, track list and discography information
Last.fm	Returns track list for a particular album

Table 3 – Websites that were scraped or elaborate queries created for, to extract data

The system design is based on a three tiers approach, which includes *presentation*, *application* and *storage*. The web browser is the first tier (presentation) from which the user specifies the search queries, an engine using some dynamic web content technology, in our case PHP represents the middle tier (application logic), and our database which is the web represents the third tier (storage). Figure 1 shows the architecture of the system, in which the user specifies a search term e.g. *"Shakira"* from the front-end GUI embedded as a web based application which is sent as a request to the middle tier, which services them by making queries against the database, the results are then formatted by the system and the information returned to the user in an integrated format.

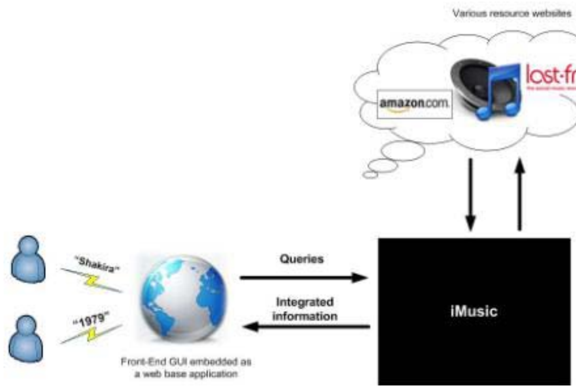
Various resource websites
amazon.com
last.fm
"Shakira"
"1979"
Queries
Integrated information
iMusic
Front-End GUI embedded as a web base application

Figure 1 – System Architecture

2.2 Implementation

The UML activity diagrams depicted in Figures 2, 3, 4, and 5 describe the workflow for different search requests in iMusic. Figure 2 describes an instance where an user enters a query, and the system searches Last.fm for related artist, albums, or tracks based on the query supplied. The system provides the user with a mechanism from which a specific category can be selected if more detailed information is needed or the user can reformulate the original query and resubmit it. If the user selects a category (track, artist, or album), the system will display comprehensive data on the category selected.

2.2.1 Artist Searches

When the system is issued a query that involves searching for artist information, several requests are made to our online sources (Tables 2 and 3), as depicted in Figure 3. Pertinent information such as artist biography, discography, events, news, similar artists, radio station, and videos are returned to the user.

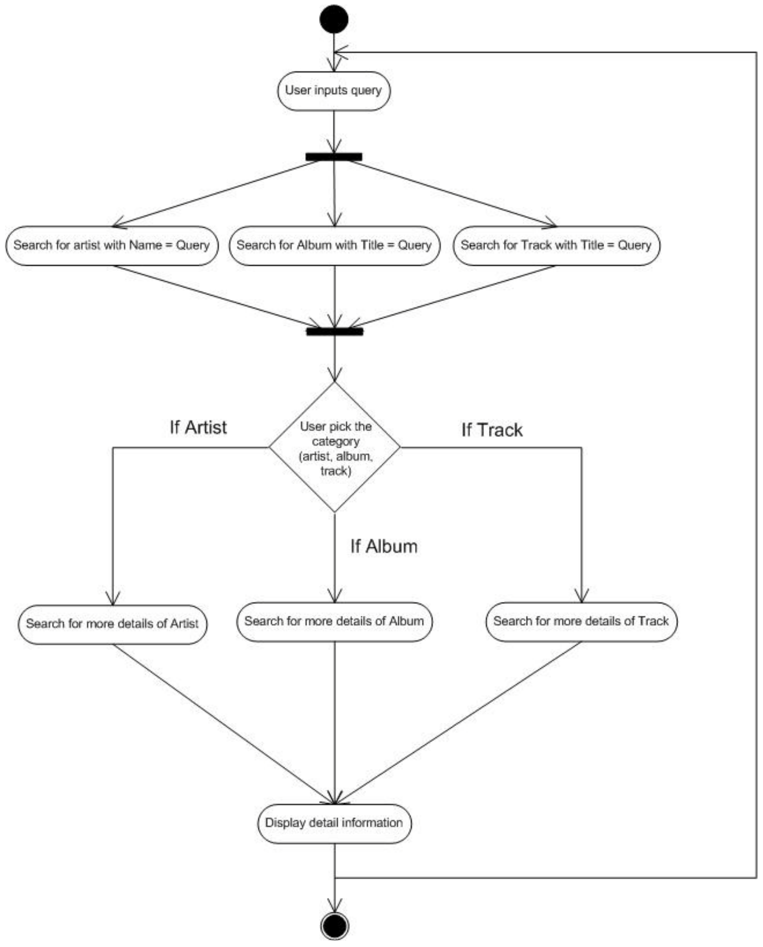
User inputs query
Search for artist with Name = Query
Search for Album with Title = Query
Search for Track with Title = Query
If Artist
User pick the category (artist, album, track)
If Track
If Album
Search for more details of Artist
Search for more details of Album
Search for more details of Track
Display detail information

Figure 2: System workflow

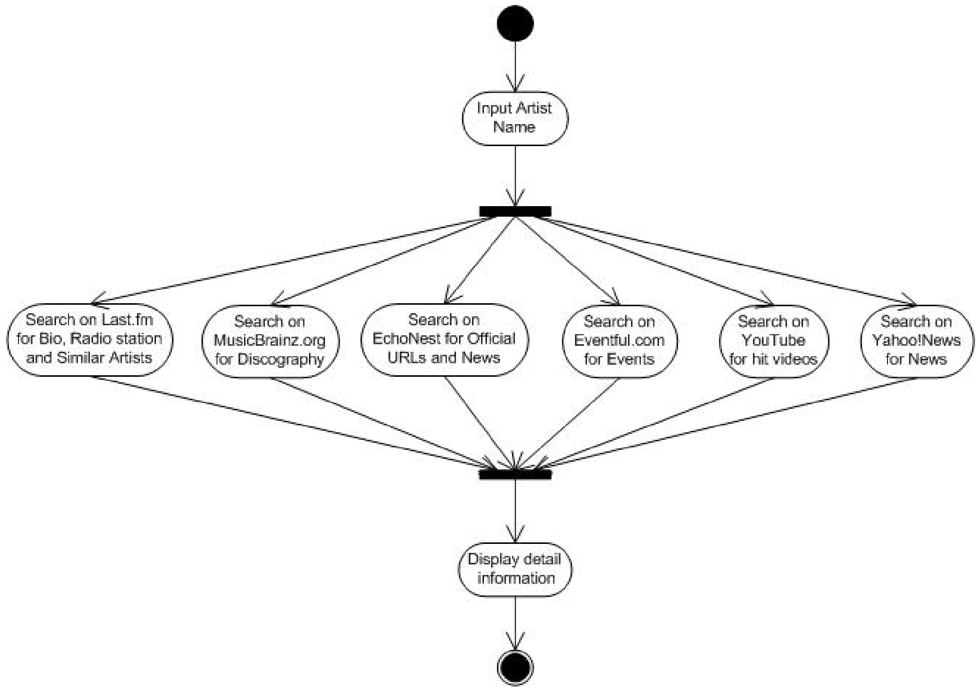
Input Artist Name
Search on Last.fm for Bio, Radio station and Similar Artists
Search on MusicBrainz.org for Discography
Search on EchoNest for Official URLs and News
Search on Eventful.com for Events
Search on YouTube for hit videos
Search on Yahoo!News for News
Display detail information

Figure 3: Searching for artist information

Biography

To search for artist biography, iMusic combines the information from Last.fm and Gracenote. Last.fm provides a short summary about an artist biography which is written and edited by users, and it can be retrieved using the `artist.getinfo` call on Last.fm API, which returns a XML tree containing the results. Once this XML response is returned, iMusic uses the *simplexml* [17] library for parsing XML in PHP.

A number of artists in Last.fm, however, lack a biography description as this is user submitted data. When iMusic cannot find an artist's biography in Last.fm, it resorts to Gracenote, which oftentimes has a more complete biography of an artist, though more verbose. However, Gracenote does not provide an API, thus we have implemented a scraping agent for that.

Similar Artists

In Phase Zero, iMusic retrieves a list of top-5 similar artists from Last.fm using the same call used to retrieve artist biography. If more similar artists were required, than iMusic would have to issue an extra request to Last.fm, only this time the `artist.getsimilar` api call. For our prototype purposes, we believe 5 artists are enough, and not having to issue a second request saves us processing time.

During Phase Zero, we have realized that a number of other online resources, such as EchoNest, Rhapsody and Yahoo! Music, also provide top-k similar artist rankings. Each of these websites, however, provides a different ranking for the same artist. We believe this is due to the differences in demographics and listening habits mined from their user base data, as most of these sources use collaborative filtering and/or co-occurrence coefficients to compute similarity [18]. This issue is addressed with more details in Section 5.

Official URLs

Each artist has several official sites in some famous social network sites or channels such as MySpace (e.g., http://www.myspace.com/madonna) and AOL music (e.g., http://music.aol.com/artist/madonna). These are reliable sources for news and events because they are official and their contents are provided by the artist's producers.

However, the url naming scheme is not always based on a regular expression, as one might think. For example, MTV's url for Britney Spears is (http://www.mtv.com/music/artist/spears_britney/artist.jhtml), while its url for Massive Attack is (http://www.mtv.com/music/artist/massive_attack/artist.jhtml). From the sources used by iMusic, only EchoNest provides the API for getting this information (official sites from iTunes, Amazon, Last.fm, AOL Music, Musicbrainz, Wikipedia, MySpace, Official website).

Discography

Most of the apis and websites from which iMusic gets information lack a straightforward way to display discography information. For example, Last.FM only displays top albums in the first page, and more albums if the user is willing to browse through the list. In addition, neither Last.FM, Gracenote, nor EchoNest has an `artist.getalbums` call (or similar) on their apis. Furthermore, the only website that displays a complete list of "releases" (e.g., albums, eps, singles, live, compilations), and offers an api call that returns a set of "releases" for an artist is MusicBrainz. Accordingly, this information has to either be scraped or collected from a MusicBrainz api call. Nevertheless, scraping from MusicBrainz has turned out to be a complex task, as the html code from the artist page is style-specific (e.g., it has a different css class for each row in the discography table) and has variable size, as well as dynamic content.

Therefore, the best option is to try and use the releases web service api call that receives as input an artist or an mbid (MusicBrainz id) of an artist. However, this call returns an unsorted set of releases associated to that artist or artist mbid without removing any duplicate entries. Hence, we had to implement a filter to remove duplicate releases from this list. For that, we use a hash table (i.e., an array in PHP) and we hash on the album title. While traversing the resulting xml tree for each release xml node (after de-serializing the xml tree with PHP's simplexml), we check if its title already exists on the hash table. If it does not, then we add the album title to the hash table together with other information about the album, such as release date and release type (i.e., album, single, ep, live, compilation).

News

In Phase Zero we have relied solely on Yahoo! BOSS API to retrieve news from Yahoo! News. However, we have concluded that the results returned are not always relevant because Yahoo! covers a broad range of topics, and we lose precision. Therefore, in Phase One we have decided to retrieve artist news through EchoNest API, and to combine both sources to try and increase the relevance of artist news displayed in iMusic. However, this leads to a number of problems related to duplicate information contained in both sources, which we address at Section 3.1 (Finding reliable data sources).

Videos

When it comes to online video sharing, the most popular site by far is YouTube, with billions of page views and hundreds of thousands of videos added daily. The service also allows users to tag videos with keywords, and view the most popular videos at any given time. The integration of music videos was done via the YouTube Data API. This API allows developers to access and search YouTube video data through a REST-based API, and integrate this data into their own XML-aware application.

This was accomplished by writing application-level code to send REST requests, parse and decode responses, and integrate the resulting data into the application interface. Using PHP, it allowed us to perform these tasks either through YouTube's PHP Client Library, or by manually parsing the XML responses based on the REST requests. To improve the relevance of the returned results from YouTube, we combined artist, album, and track in the album query search. For videos that pertain to a specific song, artist and track were combined. We have observed through experimental validation that results returned by YouTube are relevant for most of iMusic's use cases.

2.2.2 Album Searches

The goal of the system is to provide relevant integrated musical data based on a supplied query. It should be noted that there was no one online database that provided the wide repertoire of relevant information that the system needed to adequately address the needs of the user. Figure 4 shows that when the system is issued an album title, it retrieves data such as summary information, track list, hit videos, and album price by dispatching several queries to sources such as Last.fm, Gracenote, Youtube, and Amazon.

Track List

Accessing the track list for an album required implementing a scraping agent for track list information from Last.fm if the main source for that case (i.e., Gracenote or Musicbrainz) does not provide the track list for the album in question. Specifically, to get an album track list, if we have album mbid (MusicBrainz id, introduced in Section 2.2.1 Discography), we can quickly find it from musicbrainz.

However, not all albums have this information, so we need to find it from gracenote. We consider gracenote more reliable than Last.fm because information from Last.fm is user-uploaded and prone to errors. If we cannot find it from gracenote then we scrape it from Last.fm.

2.2.3 Track Searches

The user of the system can select a track from an album as described in Figure 5, from which the lyrics, video of the track, and the pricing information will be provided.

Lyrics

The first source we have considered to retrieve a song's lyrics is Lyricsfly.com. It has a complete api that allows users to search for lyrics by song, artist, album name or even by providing phrases from a song's lyrics. Accordingly, to provide access to lyrics from their API, Lyricsfly.com requires authentication through the use of API keys. This requirement is common practice among most of the APIs we have used during the development of iMusic.

In Lyricsfly, however, API key distribution for non-commercial projects is limited to a temporary key that has to be renewed by the user once a week. When using a temporary key, only 30% of the lyrics are returned when a call is issued. This process incurs in an overhead, and we believe that returning only partial lyrics defeats the purpose of usability of the api. We have contacted the api developers, and by the time this report was written, the owners do not provide permanent keys to an educational project. Therefore, we have decided to retrieve lyrics from Lyrics.Wikia, which in turn provides an API that only allows searches by artist name and song title. We cannot query for parts of lyrics, but we believe this functionality is not required for our prototype.

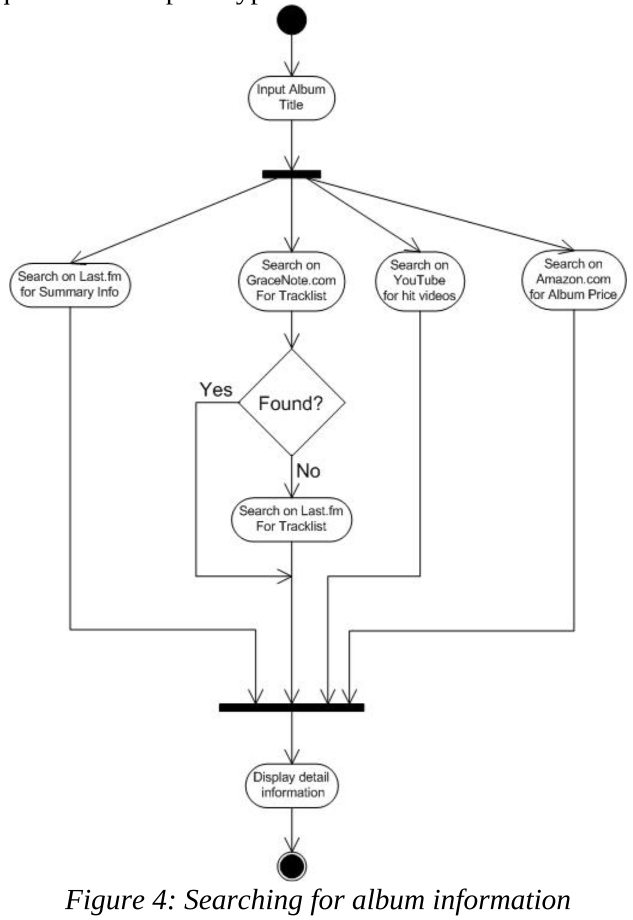
Input Album Title
Search on Last.fm for Summary Info
Search on GraceNote.com For Tracklist
Search on YouTube for hit videos
Search on Amazon.com for Album Price
Yes
Found?
No
Search on Last.fm For Tracklist
Display detail information

Figure 4: Searching for album information

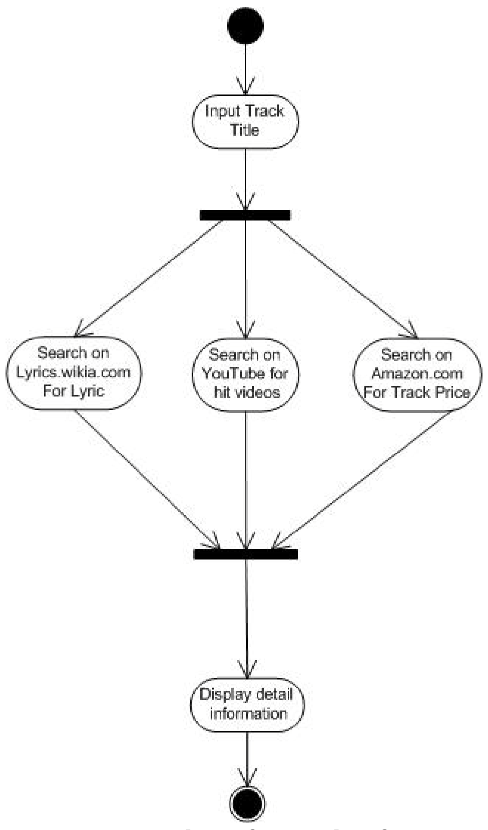
Input Track Title
Search on Lyrics.wikia.com For Lyric
Search on YouTube for hit videos
Search on Amazon.com For Track Price
Display detail information

Figure 5: Searching for track information

Price

Amazon provides a set of Web Services (AWS) where different functionalities are accessed over HTTP, using REST and SOAP protocols. Because Amazon's EWS api requires every request to be signed, and this signature computation adds to the already considerable overhead of making each request, it proved to be more beneficial to implement a scraping agent for the site to gather the required information.

There exist some limitations with using Amazon as a source for prices, since it does not have stored in their database all the information that pertains to certain tracks, e.g. tracks from a Vietnamese artist such as "ho quynh huong". Future work includes getting prices from other sources (e.g., iTunes Store, Walmart, Rhapsody) to make a comparison between all prices available in these sources, letting the user know which one is the cheapest.

3. Problems and Lessons learnt

The implementation of the system brought with it several challenges, from which we learnt many lessons. In this Section, we describe some of the problems we have faced, and how we solved them.

3.1 Finding reliable data sources

In Phase Zero, our prototype only retrieved news from Yahoo! News through its BOSS API. We have observed that this is not optimal, as Yahoo! News may return a number of non-relevant items. The reason is that Yahoo! News covers a broad range of topics including World news, Politics, Business, Health, Sports, Entertainment (including music), and so forth. Nonetheless, Yahoo! does not provide a means for querying only its music related news (neither through its API, nor through its web search feature). Therefore, the precion of the results we receive is lower, as Yahoo! will query all its news sources for our search string.

As a result, in addition to news we retrieve from Yahoo!, we have decided to use the EchoNest API. EchoNest scrapes data from different music related sources for news, such as gets data www.rollingstone.com, http://www.popjustice.com, and http://www.starpulse.com. One problem we have found in this scenario, however, is that some news articles retrieved from EchoNest are actually music reviews done by journalists that work for these sources. Ideally, we would like to eliminate these entries, as although they mention the artist in the search query, they are not examples of news. In addition, an approach we would like to employ here is to scrape news from the artist's official website, as well as from its profile at MySpace, AOL, MTV, and so on. The current version of iMusic provides links for these websites, and we leave the task for scraping news from these sources as future work.

Similarly, we have observed that our sources for concerts and shows information are not always reliable. For example, a number of artist events posted in Last.fm are fake. Unfortunately, this type of events cannot be filtered out by regular spam detection techniques, as they do not present the same patterns as web spam. One of the fake events we have found for Madonna was actually an event of an artist that is famous for doing covers of Madonna in Brazil. The user that posted this event has tagged it as a Madonna concert, and a number of other users claimed they were going to attend it. Therefore, this particular item presents the characteristics of a real event. Therefore, for this Phase of iMusic, we have abandoned Last.fm as a source for artist events and rely on Eventful, which has proved to be more reliable in this context, even providing links for venues in which the artists will do their performance.

3.2 Filtering duplicate information

While getting data from multiple sites and even from the same source, it is usually the case that they contain duplicate information. Therefore, it is essential that duplicate entries are filtered out. Examples of duplicated information are those obtained more than once from different sources, such as news (e.g., Yahoo! News and EchoNest), and APIs that do not remove duplicate content (e.g., discography information obtained from MusicBrainz).

Our current solution for discography, where names of duplicates can be differentiated in a binary decision (i.e., they are either the same album, or they are different albums) is to use a hash table and hashing on the attribute that should be the same for the same items (i.e., album title).

In gray areas, such as news from different sources, where the news title can be slightly different and the article presents almost the same content, we cannot employ a hash table. Our idea here is to use string similarity scores, such as Levenshtein distance [20]. The complexity of this algorithm is $O(m * n)$, where m and n are the lengths of the two strings to be compared. Additionally, we have to perform this comparison in a $p * q$ fashion, where p is the number of news articles retrieved from Yahoo! News, and q is the number of articles retrieved from EchoNest. Therefore, the complexity of this filtering for ony two sources in the worst case tends to O(max(n, p, q)^3). Hence, we leave it as future work, as we would need to migrate iMusic's current implementation to a more powerful server.

3.3 Auto-complete: less usable than we expected

In order to reduce uncertainty and typing errors from user queries we planned to provide auto-complete (i.e., possible name suggestions) for all 3 types of queries: artist, album and track. However, a downside feature of auto-complete is that we have to update the suggested list every time the user types in a new character. Another issue is the integration of result list from 3 different data sources. For example, artist, album, and track have different xml schemas, and integrating thTese into a single schema (e.g., using xsl transformations) inside the PHP proxy that we implemented to make the requests requires additional processing overhead.

In addition, the number of requests generated by auto-complete, which is already considerable providing only artist support, is multiplied by 3 if iMusic is to provide support auto-complete for albums and songs as well. Because of network latency, this high number of requests slows down performance past the point where the feature can still be considered usable. This issue could be solved if we had a local index of all music related information, which seems unfeasible due to the massive amount of music information available.that would have to be crawled and indexed. Another approach would be to cache some of the query results in a database, but this still defeats the purpose of auto-complete for first time user queries. Therefore, we turn off the auto-complete feature in Phase One, while we consider other solutions, such as "did you mean" spelling suggestions.

3.4 Online searching or local indexing?

The goal of our system was to fit into vision proposed by iWisdm, hence our main database has been the Web. We have discovered that using the Web as our primary source of data has many advantages, but it also has a number of drawbacks. On the one hand, online searching presents a wider corpus of data that can be manipulated to provide users with a wider array of relevant information. On the other hand, it was clear to us that the use of online searching brought with it the disadvantage of network latency, that is the time delay experienced in the system, due to the number of HTTP requests that are made.

Nevertheless, local indexing does not solve all problems here. Crawling the web for pertinent data such as artist, albums, tracks, and other relevant information and having that stored in our own database seems unfeasible due to the massive amount of music information available that would have to be crawled and indexed.

Therefore, the approach here would have to be in terms of which searches can be parallelized (e.g., bio, discography, videos), and execute them in parallel threads. Another solution that can be combined with the use of parallel threads is to cache in a database some of the query results that do not change often (e.g., images, biography, discography), and to periodically flush data contained in the cache to ensure freshness.

3.5 Reliance on third-party APIs

APIs used throughout our project were not static, and some of them experienced different levels of evolution throughout the span of the system. These evolutions provided new improvements to the APIs, and thus served as a means of enhancing our system. For example, Last.FM did not provide a method for finding top artists and top tracks from a particular region (i.e., geo.gettopartists and geo.gettoptracks), but these were added during Phase One of iMusic. Similarly, API calls that return more information about an artist (e.g., artist.geturls) among others, were added to EchoNest API. We concluded that it is imperative that our system remains current with the latest improvements afforded us by these API, and where feasible to add these functionalities to the system to ease development.

Nevertheless, a drawback from relying in third-party APIs for some of our features is that if these companies stop providing their services, iMusic will suffer loss of functionality. In the future we would like to shift dependence from these sources by implementing support for user-provided data, and building our own database of music information. This process would take considerable amount of time, as well as marketing strategies, and thus we leave it as future work.

4. Related Work in Query Systems for Music Information Retrieval

Since the past couple of years there have been seen heighten interest in discovering novel ways through which the Music Information Retrieval (MIR) systems can be improved. Music Information Retrieval (MIR) is a multidisciplinary research endeavor that strives to develop innovative content-based searching schemes, novel interfaces, and evolving networked delivery mechanisms in an effort to make the world's vast store of music accessible to all. There are several services (e.g, Last.fm, Pandora.com) and research prototypes (e.g., [4, 5, 6]) which support music search and recommendation.

These systems rank the results of music search based on their relevance, quantified by a music similarity measure. In analogy to web search, it is useful to rank the results of music search by both importance, which reflects popularity, and by relevance, which reflects similarity [7]. The reason is that importance enables the retrieval of mainstream music (authoritative artists/songs), whereas similarity enables the discovery of new music (serendipity effect).

In recent work [4, 5], hybrid similarity measures were proposed, which combine audio extracted features with social media (playlists, tags) mined from the Web. Along the same lines, [6] proposed an audio crawler that mines mp3 blogs to find initial audio files relevant to a query, from which new music is then discovered by means of audio similarity. In the next paragraphs, we focus our discussion in query systems for music information retrieval. The issue of classification will be addressed later in Section 5.1.

4.1 Query-by-text

To query for music information in a query-by-text system, an user submits a string of text describing what he or she is looking for, such as artist name, album title, song title, or pieces of lyrics. Retrieval can be done by comparing the query string to metadata associated to music items in its database (e.g., name, title, social tags) using text information retrieval techniques. The engine then has a choice between providing only exact matches, or suggesting approximate ones.

In query-by-text systems, however, the user is limited to what text can express. For example, an user cannot look for a specific melody he or she heard recently, unless the system provides a notation for representing musical notes, such as Parsons Code [12]. Similarly, an user cannot find songs that have acoustic guitar as one of their instruments, unless that information has been explicitly tagged in any matching songs in the system's database. Examples of query-by-text systems are Last.fm, Rhapsody, Google Music, and iMusic.

Although our system performs queries by text, we have sought to classify the results in terms of entities such as artist has albums, albums has tracks, and tracks have lyrics. We believe this classification should be extended to support retrieval of other music information, such as mood, genre, style, personality, and sound attributes. We address the open problem of music classification in Section 5.1.

4.2 Query-by-humming

In a query-by-humming system, an user submits its query as a hummed melody, and the system then looks for specific sound patterns (i.e., by means of sound analysis) to compute a signature for the query. This signature can then be compared in terms of similarity to signatures of songs in the database.

An advantage of this kind of query system is that it allows an user to search for songs when he does not know previous information about it, other than its melody. A drawback from this approach, however, is that the melody has to be unique in some way. If not, then signatures may not be distinct enough, and thus not differ considerably from each other, hence leading to false positives and/or negatives. Midomi [15], Tunebot [13], and Musipedia [14] are examples of systems that allow query-by-humming as part of their user interfaces.

4.3 Query-by-example

In this kind of query system, a user can submit an audio file (or samples of it) to try and find songs that are considered similar to the file. Aspects of similarity in query-by-example systems range from spectral analysis and timbre [16], to tonality and tempo.

A drawback from this approach, however, is the refinement process. For instance, if the initial results are not satisfactory for the user, then he or she has to submit other samples to aid the retrieval of new results that are more similar to what he or she is looking for. In addition, there is the additional burden in system storage and usability of uploading mp3 or midi files to the system's server.

As a result, this approach does not scale well, both for servers (because they need to handle multiple requests, each one uploading chunks of files), and for users (because they need to first obtain samples that are similar to what they are looking for). An examples of query-by-example systems is QUEBEX [20].

4.4 Relevance Feedback

Relevance feedback is not a query system per se, but rather a way that an user can guide the system in its retrieval process to obtain more relevant results. This can be integrated to essentially all query techniques mentioned in the above sections, and it is the case for systems such as Pandora, and Last.fm. Although Phase Zero of iMusic did not provide relevance feedback, we plan to provide support for it, as we believe it is essential for any music information retrieval systems to allow some kind of user feedback.

Nevertheless, we believe this should be done carefully, as relevance feedback from other users has a potential for damaging the user experience, because of the large amount subjectivity involved in music opinions. For example, different users may consider different artists similar in different levels, ranging from equal to completely opposite. A system that takes into consideration a user's preferences, and that recommends different artists and songs in a personalized fashion (as opposed to generic recommendations e.g., for all listeners of Madonna) may have potential for achieving more relevant results.

5. Phase One: Envisioning a Smarter Prototype

In this Section, we address some of the open research problems we envision for a Phase Two of iMusic. Additionally, we give an overview of what existing systems do with respect to these problems, and propose initial solutions for them.

5.1 Music Classification

In iMusic, we have sought to classify results in terms of entities, such as artist has albums, albums has tracks, and tracks have lyrics, to name a few relationships. However, iMusic's current classification has a number of limitations. For example, one has to know in advance what artist, album or song he is looking for, and then iMusic will display all pertinent data. We believe our classification should be extended to support retrieval of music information by providing other kind of queries as well, such as mood, genre, style, personality, and sound attributes. This information is not readily available on the Web, however, and to the extent of our knowledge, there are only a few websites that use mood [21, 22] or sound attributes [19] to classify songs in their databases.

As future work, we propose building an ontology of music information that takes into account the aforementioned attributes, as well as other attributes that we think may be useful in querying. The latter could be obtained by conducting a study of user habits when looking for music on the web. Then, we would classify a set of artists in terms of the attributes described in this ontology. For experimental validation of our classification, again we suggest conducting a study with users, as there is no available ground truth for what is the correct classification of music information in terms of the attributes we would like to use.

5.2 Song similarity

As in the artist similarity scenario, one of the main challenges in song similarity is that there is no commonly agreed definition of what similarity in this context is. As opposed to textual documents, a song is not just a bag of words, and textual information retrieval techniques cannot be directly applied. In particular, songs possess a number of characteristics, such as the ones in the non extensive list below:

- Genre (e.g., pop, rock, metal, classic)
- Mood (e.g., sad, happy, romantic)
- Tempo (e.g., adagio, allegro, largo)
- Instruments (e.g., acoustic guitar, synth, piano)
- Tonality (e.g., major, minor, chromatic)
- Lyrics (e.g., language, main message, mood)

In this section we outline some of the current approaches for computing song similarity, and what room for improvement we believe there is in this area.

A naive way to compute song similarity would be to just display all top-k tracks from any top-k similar artists. However, this approach is clearly far from optimal. For example, consider the top-3 songs from Madonna, as well as for the top-3 artists similar to Madonna, according to Last.FM:

Madonna	**Kylie Minogue**	**Janet Jackson**	**Cher**
1. Hung Up	1. Can't Get You Out of My Head	1. Feedback	1. Believe
2. Celebration	2. In My Arms	2. That's The Way Love Goes	2. If I Could Turn Back Time
3. Like a Prayer	3. Love at First Sight	3. Together Again	3. The Shoop Shoop Song

Table 4 – Top-3 most popular songs from top-3 most similar artist to Madonna, according to Last.fm

As can be seen in Table 4, Madonna's "Hung Up" would be considered similar to Kylie Minogue's "Can't Get You Out of My Head", as well as to Cher's "Believe". This similarity can be disputed, however, as these three songs present considerably different use of electronic instruments, rhythm, and vocal harmony, just to mention a few characteristics.

Another approach to estimate similarity between songs is to compute the Jaccard's coefficient between the set of users that listen to that song, according to Formula 1:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}. \qquad (1)$$

Where A is the cardinality of the set of users that listen to song "a", and B is the cardinality of the set of users that listen to song "b". Nevertheless, this leads to a problem which can be illustrated in the following scenario. Supposed a band called "Jane Doe Band" is largely influenced by popular rock female singers, such as Pattie Smith and Blondie. In fact, Jane Doe is considered by all its 200 listeners to be the most similar artist to Pattie Smith in the world, and one of Jane's songs, "Unknown 1", sounds remarkably like Pattie's "Because the Night". However, as "Because the Night" has been listened literally a thousand times more than "Unknown 1", it probably will not be suggested as a similar song.

The reason for this is as follows. If a system uses collaboration filters (such as a co-occurrence coefficient like Jaccard's), to compute similarity between these two artists, "Unknown 1" will never be in the top-k similar songs to "Because the Night". This happens because 200 is a number too small if compared to 200.000, and thus cannot yield a good similarity score for Jane Doe. As a result, "the rich get richer", and only the most popular artists are recommended.

We believe the aforementioned problem is a serious one, as it inhibits new artists that are not yet popular to figure in the first pages of existing music recommendation systems. Unfortunately, most music recommendation engines available today use collaboration filters to compute similarity between songs [18]. One of the only exceptions is Pandora, which uses feature extraction to classify songs in terms of sound characteristics presented by them [19]. The weight of these attributes is obtained through sound analysis, and these values are used to form a vector representing that song. Similarity is then computed as the distance between these vectors. Still, Pandora tends to recommend only popular songs as similar, and we believe the reason is that its database is comprised of only artists that are part of major labels who have signed up to be included in Pandora's results.

5.3 Artist similarity

Before computing a similarity score between different artists, one should first define what similarity means in this context. One of the problems in this scenario, however, is that there is no commonly agreed definition of similarity among two or more artists. The most widely deployed technique in current music search engines is the use of collaborative filtering and computing similarity coefficients based on co-occurrence (e.g., Jaccard and Dice coeffiecients) of songs in the artist listeners' top-k list. Previous work in the literature that use co-occurrence coefficients include Knees et. al [9], who apply TF-IDF in web documents that pertain to an artist (e.g., homepages, artist profiles in music search engines) to weight terms that give an indication of genre and similarities with other artists.

Additionally, one of the problems in music information retrieval field is that there is no ground truth to evaluate the quality of the results. Similarly to web search, a user's notion of relevance of results is subjective. For example, consider the Table 5 below that depicts a top-10 list of artists similar to Madonna in four popular music search engines.

Last.FM	***Rhapsody***	***EchoNest***	***Yahoo! Music***
1. Kylie Minogue	1. Lady GaGa	1. Kylie Minogue	**1. Anastacia**
2. Janet Jackson	**2. Robyn**	2. Dannii Minogue	2. Kylie Minogue
3. Cher	3. Katy Perry	3. Gwen Stefani	**3. Robbie Williams**
4. Lady GaGa	4. Kylie Minogue	4. Britney Spears	**4. Mecano**
5. Britney Spears	**5. Christina Aguilera**	5. Janet Jackson	**5. Miguel Bosé**
6. Dannii Minogue	6. Celine Dion	6. Kylie Minogue & Jimmy Little	**6. Ella Baila Sola**

7. Cyndi Lauper	**7. Yelle**	7. Kylie Minogue Featuring MIMS	**7. Hombres G**
8. Spice Girls	8. Jennifer Lopez	8. Cyndi Lauper	**8. Gipsy Kings**
9. Sophie Ellis-Bextor	9. Gwen Stefani	**9. Hilary Duff**	**9. Michael Jackson**
10. Paula Abdul	**10. Mariah Carey**	10. Karaoke - Britney Spears	**10. Andy & Luca**

Table 5 - Madonna's similar artists according to four popular music search engines, as of December 2nd 2009

As can be seen above, the artists included in these four lists are not ranked in the same way by all search engines. In addition, all four lists have at least one artist that has not been mentioned in any of the other sources' ranks (highlighted in bold). Moreover, Yahoo! Music provides a list of artists that share no similarity to Madonna other than the fact that they can also be considered pop artists. More specifically, these are all mainstream artists popular in spanish speaking countries. We believe there are two main reasons for why these artists have appeared in Yahoo! Music's list of artists similar to Madonna:

1. Yahoo! Music's user base may be comprised of many hispanic people, as a number of bands shown in their ranking are popular in spanish-speaking countries.
2. The so-called "Harry Potter problem" [10], where a data mining engine associates a highly popular item (e.g., Madonna) to most items in its database. This is similar to words with high frequency in a document information retrieval engine, and can be diminished applying inverse document frequency to the weight of "Madonna" term.

During Phase One of iMusic we have come up with a solution for dealing with the disparities in similar artist rankings from multiple sources. The idea was computing a weighted sum of the rank for each artist provided by *n* sources, where r_i is the ranking given by source *i*, and w_i is the weight assigned for each source, as Formula 2 shows:

$$R = \sum_{i=1}^{n} w_i r_i \qquad (2)$$

However, there are two problems with our proposed solution. The first is that to get accurate weights for each source, we need a training phase, and there is no available ground truth to compare it with. If we then remove the weights, we are still relying on sources that may not necessarily give a good ranking to start with, as can be evidenced by the fact that they only show popular artists in their listings. This happens because these sources use collaborative filtering to compute their scores, and thus we only reinforce popularity of artists in the results.

Hence, we believe this is far from optimal, and similarly to the song problem, collaborative filtering does not do a good job for computing similarity between artists. In addition, again the "rich get richer". None of the sources we have investigated during this project suggests similar artists that are not already considerably popular. Thus, an user is lead always to only a small percentage of the massively available digital music content. Therefore, these search engines do not help the user to actually find music, as he or she can only explore the surface of this this massive collection of information.

5.4 Visualization

Visualization of music data is another research problem that would be interesting to tackle. Solutions currently available focus on visualization based on sound analysis (e.g., music players such as Windows Media Player, Winamp, and iTunes), and relationships between artists or songs such as similarity (e.g. MusicMap) or genre. Although outside of scope of our Phase Zero prototype, we are particularly interested in visualization approaches that explore other kinds of relationships between artists, albums, and songs. For example, it is particularly useful to study music patterns if dimensions such as time, level of collaboration degree, and social tags are available to be displayed.

As future work, we propose exploring an approach similar to DBLPVis [11], with modifications on the types of entities. More specifically, we could display Person as artists (instead of authors), social tags (instead of words), recording label or venue stream (instead of conference stream). In addition, we would like to extract information from lyrics to compute top-k words, and provide a notion of time by associating these with release year of songs and alubms. We would also like to conduct an experimental validation with of these visualizations with users.

However, DBLPVis has the advantage of dealing with a set of structured data from DBLP. We believe gathering trustworthy data to be able to infer the aforementioned relationships represents a significant challenge in the music field, as most data available in this domain is not structured. For example, current publicly available music search engines and databases do not have information about what genres and social tags are associated to a particular artist over time, as it is the case for authors in DBLP (i.e., keywords are extracted from papers, and papers have a year of publication associated to them).

6. Conclusion

In this report, we have described an approach that facilitates the integration of data to produce efficient music queries using multiple distributed data sources. Furthermore, we concluded that the quality of search results can be enhanced by using a number of available music resources to respond specifically to a user query. The main goal of iMusic was to serve as a component of iWisdm which would be able to handle "music" queries for a combination of keywords, and return a comprehensive corpus of information applicable to the query in question. Phase Zero of the system represented an en-to-end system that allowed users to perform different music searches ranging from artist, album, or track. The data integration was made possible through the use of several APIs, and through the implementation of specialized scraping agents.

The system's design was based on a three tiers approach, where the Web was used as our primary database. However, the problem of network latency has been shown to increase performance overhead as a function of the number of sources that iMusic had to be visit. One solution we propose to address this in iMusic is to cache in a database some of the query results that do not change often (e.g., images, biography, discography), and to periodically flush data contained in the cache to ensure freshness.

In addition, we have proposed a number of open research problems. We believe existing systems do not do a good job of helping find music information that is not already popular, and this is essential if we want to cope with the increasingly large amount of digital music that is becoming available today. Other challenges such as computing similarity between artists and songs, as well as coming up with efficient visualization of music relationships have been introduced. As exciting as these problems are, unfortunately due to time constraints for Phase One we have not been able to solve them. Therefore, we leave the door open to a Phase Two of iMusic that will address these issues.

7. References

[1] Keyword Ranking Popularity Report. Available online at

http://mostpopularkeywords.net/rank/?keyword=music

[2] Daniel Gruhl, Meena Nagarajan, Jan Pieper, Christine Robson, Amit Sheth. "Context and Domain Knowledge Enhanced Entity Spotting in Informal Text". *Proceedings of the 8th International Semantic Web Conference (ISWC) 2009*. Fairfax, Virginia, USA. October 2009.

[3] Don Byrd. Organization and search of musical information. Syllabus for a course available online at http://informatics.indiana.edu/donbyrd/Teach/I545Site-Spring08/SyllabusI545.html

[4] P. Knees, T. Pohle, M. Schedl, and G. Widmer, "A music search engine built upon audio-based and web-based similarity measures," in Proc. SIGIR, 2007, pp. 447–454.

[5] R. Stenzel and T. Kamps, "Improving content-based similarity measures by training a collaborative model," in Proc. ISMIR, 2005, pp. 264–271.

[6] O. Celma and P. Herrera, "Search sounds: An audio crawler focused on weblogs," in Proc. ISMIR, 2006, pp. 104–111.

[7] J. Kandola, J. Shawe-Taylor, and N. Cristianini, "Learning semantic similarity," in Proc. NIPS, 2002, pp. 657–664.

[8] Shao, B., Li, T., and Ogihara, M. 2008. Quantify music artist similarity based on style and mood. In *Proceeding of the 10th ACM Workshop on Web information and Data Management* (Napa Valley, California, USA, October 30 - 30, 2008). WIDM '08. ACM, New York, NY, 119-124.

[9] P. Knees, E. Pampalk, and G. Widmer. "Artist Classification with Web-based Data." In Proceedings of 5th International Conference on Music Information Retrieval (ISMIR '04), pages 517--524, Barcelona, Spain, October 2004.

[10] Lamere, P., and Volodkin A., "Help! My iPod thinks I'm emo – Part 1", Available online at Music Machinery, http://musicmachinery.com/2009/03/26/help-my-ipod-thinks-im-emo-part-1/

[11] DBLPVis, Available online at http://dblpvis.uni-trier.de/

[12] Parsons, Denys (2002). *The Directory of Tunes and Musical Themes*. Bohmeier.

[13] D. Little, D. Raffensperger, B. Pardo. *A Query by Humming System that Learns from Experience*. Proceedings of the 8th International Conference on Music Information Retrieval, Vienna, Austria, September 23-27, 2007.

[14] The Open Music Encyclopedia. Available online at http://www.musipedia.org/

[15] Finding Structure in Audio for Music Information Retrieval, article by Bryan Pardo, IEEE Signal Processing Magazine, vol. 49 (8), pp. 49-52, 2006

[16] W. Dixon Ward (1970). "Musical Perception". in Jerry V. Tobias. *Foundations of Modern Auditory Theory*. 1. Academic Press. pp. 409.

[17] Simplexml library for parsing XML in PHP. Available online at http://php.net/manual/en/book.simplexml.php

[18] Collaborative filtering in commercial systems. Available online at http://en.wikipedia.org/wiki/Collaborative_filtering#In_commercial_systems
[19] Music Genome Project. Available online at http://en.wikipedia.org/wiki/Music_Genome_Project
[20] B. Thoshkahna, K.R. Ramakrishnan, "Projekt Quebex: A Query by Example System for Audio Retrieval," *Multimedia and Expo, IEEE International Conference on*, pp. 265-268, 2005 IEEE International Conference on Multimedia and Expo, 2005.
[21] Experience Project. Available online at http://www.experienceproject.com/music_search.php
[22] Stereomood. Available online at http://www.stereomood.com/